INTERNATIONAL HELLENIC UNIVERSITY

Applications for Smart Devices

Andrea Dhimitri

SID: 3301100017

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in ICT Systems

SEPTEMBER 2011

THESSALONIKI – GREECE

INTERNATIONAL
HELLENIC
UNIVERSITY

Applications for Smart Devices

Andrea Dhimitri

SID: 3301100017

Supervisor: Prof. Ioannis Vlachavas

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in ICT Systems

SEPTEMBER 2011

THESSALONIKI – GREECE

DISCLAIMER

This dissertation is submitted in part candidacy for the degree of Master of Science in ICT Systems, from the School of Science and Technology of the International Hellenic University, Thessaloniki, Greece. The views expressed in the dissertation are those of the author entirely and no endorsement of these views is implied by the said University or its staff.

This work has not been submitted either in whole or in part, for any other degree at this or any other university.

Signed: ....

Name: ....

Date: ....

Abstract

This dissertation investigates and Ambient Intelligence system and proposes algorithms and techniques that can be implemented in a smart building environment. Implementing AmI environments, in regards to the connection infrastructure, essentially involves dealing with wireless networks Wireless Sensor Networks (WSN) or Wireless Sensor and Actuator Networks (WSAN). In addition the concept of Service Oriented Architecture (SOA) which is directly related to ambient systems is implemented using web services. Such a system is built for the purpose of a research project, namely Smart IHU. It involves the deployment of heterogeneous WSNs at the International Hellenic University, and the development of various web an Artificial Intelligence applications. A SOA compliant web service middleware, named aWESoME, unifies access to heterogeneous WSN systems, and exposes all their functions through web services (WS). In this work a Linux embedded computer was investigated, configured and adapted in order to replace existing gateways, which are simple PCs and consume far more energy. Experiments were conducted in different scenarios that measured the power dissipation and response time. Furthermore additional technologies and methods were investigated and used for the development of web services that control (e.g. shutdown, reboot, wake on LAN) and monitor parameters (e.g. CPU utilization) in distributed computers. These services utterly extend and enrich the functionality of the system's middleware, aWESoME.

Acknowledgements

This thesis was developed in the context of my post-graduate studies. The research was related with the Smart IHU project. For the completion of this dissertation, initially I would like to thank Professor Ioannis Vlachavas for the acceptance in selecting this dissertation topic and for providing the facilities both in the International Hellenic University and in the Aristotelian University of Thessaloniki. The development and completion of this dissertation topic was in collaboration with the Smart IHU Team, which is formed by the researchers Dr. Aggeliki Tsioliaridou, Mr. Athanasios Stavropoulos and the supervisors Dr. George Koutitas, Dr. Dimitris Vrakas. I would like to thank the Smart IHU Team for providing me the appropriate guidelines, helping me in addressing problems. I would like to thank Mr. Stavropoulos and Mr. Kostas Gottis for their precious help in the development of some of the Java Applications and Web Applications including Web Services. Especially I would like to thank Mr. Stavropoulos for providing me the appropriate guidelines, helping me into addressing and solving problems faced. Also I would like to thank Dr. Koutitas for his advices and support.

Last, but not least, I would like to thank my parents and my brother for supporting me in each step of my professional and academic career.

Andrea Dhimitri

25/09/2011

List of Contents

List of Figures

List of Tables

List of Charts

1 Introduction

The main objective of this work is to investigate concepts like ambient intelligence (AmI) and service oriented architecture (SOA). AmI is all about sensing, reasoning and acting. Initially the system senses the environment. Afterwards, based on predefined algorithms, makes decisions and finally acts. To implement such systems, the existence of a wireless sensor network is required. Concepts like Service Oriented Architecture, which can be implemented by Web Services (WS), are useful in order to connect applications. Therefore Chapter 2 includes the review of previous and present literature regarding Ambient Intelligence (AmI), Wireless Sensor Network (WSN), Service Oriented Architecture (SOA) and Web Services. The AmI concept will be presented providing additional examples of different approaches and applications. The second part of Chapter 2 includes a detailed presentation about Wireless Sensor Networks (WSN) and Wireless Sensor and Actuator Networks (WSAN). In this section some of the existing protocols in each layer, of the simplified OSI model, will be described. In addition IEEE communications standards and protocols (e.g. ZigBee) will be presented including WSN's applications. The third and last section of Chapter 2 presents the concept of Service Oriented Architecture (SOA). In this section the Web Services (WS) will be described including the protocols used (e.g. XML, SOAP, WSDL and UDDI) and available Java Platforms (e.g. J2EE, Java EE 5 and Java EE 6) for developing web services.

The first part of Chapter 4 describes the research project named Smart IHU. The overall architecture of the system is presented, along with the available systems (e.g. Plugwise, PrismaSense, OWL and CurrentCost) used in this project. The system consists of devices, including the software applications provided by their manufacturers. A middleware layer, named aWESoME (a WEb Service MiddlewarE), was developed in order to integrate the heterogeneous systems and to expose data and functions through web services.

After describing overall system of the Smart IHU project, evaluations were made and problems were found in combination will possible improvements. In the last section hardware and software tools were presented. These tools could possibly address existing problems and improve existing systems. Such tools are embedded devices which offer lower power dissipation and could replace the existing gateways, which were normal PCs. The software tools used were the Apache Tomcat 6.0, which can be installed on a Linux embedded single board computer (SBC) and used to deploy the existing middleware. Another important software tool is the NetBeans IDE 7.0 which is used to develop Java web applications and web services.

In general, Chapter 4 will describe the work performed in this report including experiments, addressing and solving problems. This chapter is formed in two main parts. The first consists configuring a SBC, named FoxBoard, to be able to serve as a gateway by running the middleware (i.e. aWESoME), aggregating data collected by the sensors and provide functions as web services. The second part is about the development of a Java Web Application which facilitates the remote monitoring and control of Computers. The functions of this Web Application will be provided as web services and finally extend the aWESoME middleware.

To conclude, in chapter 5, the conclusions retrieved by this report will be presented. In addition future development will be also presented in this chapter.

2 Literature Review

This chapter presents the basic concept used in this work, in regards to the previous and present literature. The main concepts are Ambient Intelligence and Service Oriented Architecture. Wireless Sensor Networks will be also presented since they provide interconnection between sensors, actuators, displays and processing devices.

2.1 Ambient Intelligence

Ambient Intelligence (AmI) is linked with electronic environments that are sensitive and responsive to humans. AmI is a vision developed in the late 1990s for a future digital system for the time frame 2010-2020. This vision has become very influential in the development of new concepts for the information processing combining interdisciplinary fields including electrical engineering, computer science, industrial design, user interfaces, and cognitive sciences (Aarts & Encarnacao, 2008). In an AmI world the devices operate collectively using information and intelligence found in the network connecting them. This will support people into their daily activities and will make their surrounding more flexible and adaptive.

The AmI vision is derived from the Ubiquitous or Pervasive Computing technological paradigms where users are surrounded by computing devices.

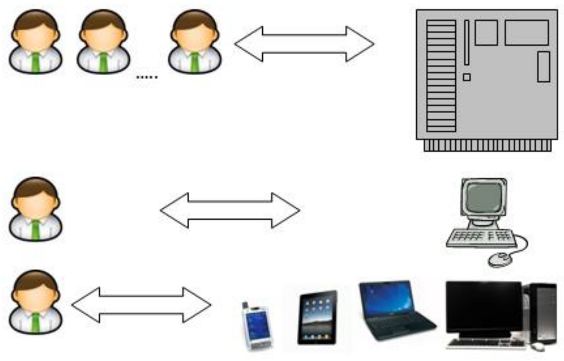


Figure 2.1: People and Computing power ratio

Initially computers were expensive and considered as a precious resource. Therefore a single mainframe computer was used by many users. The shift of the people and computer ratio is shown in Figure 2.1. In the 1980s the PC revolution has changed the computer per user ratio. Nowadays the industry progressed, the costs dropped and the ratio is many computing devices to a single user. As a result the computational resources available to each user have dramatically increased comparing to previous decades.

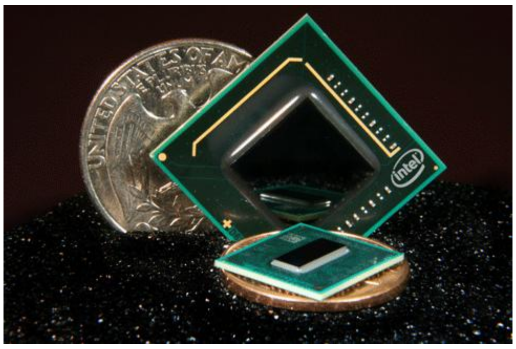


Figure 2.2: Indicative Size of Microprocessors used in portable Devices

Figure 2.3: Smart Home Appliances

Additionally, as shown in Figure 2.2, due to miniaturization of microprocessors, the computing power is often embedded in familiar devices like home appliances (e.g. in Figure 2.3 : refrigerators, washing machines etc.) and portable devices (PDA, GPS navigators etc.). These advances in technology in combination with the high user acceptance and experience, in these smart devices, have given a significant advantage in the development of AmI.

The basic idea behind Ambient Intelligence is that by enriching an environment with technology (e.g. sensors and devices interconnected through a network), a system can be built to act as an "electronic butler", which senses features of *the users and their environment, then reasons about the accumulated data, and finally selects an action to take that will benefit the users in the environment* (Cook, Augusto, & Jakkula, 2007, p. 3).

AmI Systems must be sensitive, responsive and adaptive (Cook, Augusto, & Jakkula, 2007, p. 3) and have a determinative relationship with many areas in computer science. Cook et al in 2007 organized the contributing technologies in five areas, as shown in Figure 2.4. These contributing technologies will be presented in the next sub sections.

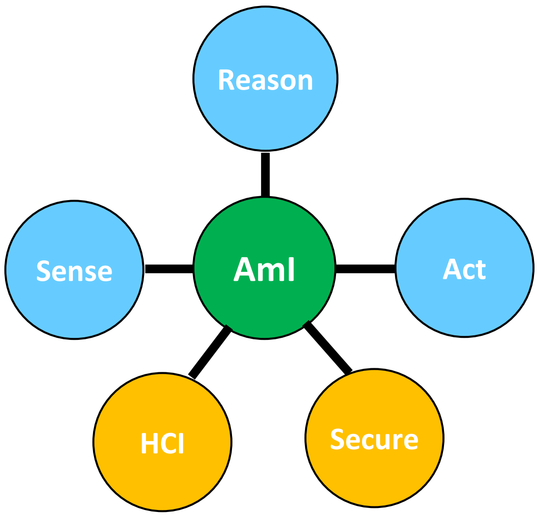


Figure 2.4: Relationship between AmI and contributing technologies

2.1.1 Sensing

Sensing the environment include the existence of sensors in order to perceive the physical environment. Many sensors have been designed to determine light (luminance), temperature, pressure, radiation, position, velocity, acceleration etc. These sensors usually have small dimensions and can be easily integrated in any AmI system. Wireless sensor networks have become very popular recently. A detailed presentation about sensors and sensor networks will be provided in section 2.2. Additionally RF-ID technology is used in order to spot objects (e.g. label the wallet or keys to be able to find them) (Cook, Augusto, & Jakkula, 2007, p. 18) or even control lights and temperature (Jabjone, Chatchaiyadej, & Chantavichean, 2009) . In Figure 2.5 an RFID bracelet is shown. This bracelet could be used by the system to detect an individual’s location.

Figure 2.5: RFID Bracelet

2.1.2 Reasoning

In between sensing and acting the need of intelligent algorithms is crucial. To make these algorithms adaptive and responsive the reasoning process must include modeling, activity prediction and recognition, decision making and spatial-temporal reasoning. Modeling consist the ability to model the user's behavior. Since this model can be build the AmI software could provide customized services to the user. If the model is accurate anomalies could be detected and changes performed in the user's patterns. In model building the data source, that is often used, is low level sensor information. Data mining techniques are used in order to spot patterns in these data and build a model corresponding to the user's behavior (Cook, Augusto, & Jakkula, 2007, p. 7). Prediction algorithms have been developed in order to predict the user's position or even actions. This allows the AmI system to predict the user's needs and alert or even perform the action itself (Helal, et al., 2003). Research has been focused in the decision making process in AmI systems (e.g. Argumentation-Based decision making (Neves, Santos, & Machado, 2007)). Another important reasoning type is spatial and temporal reasoning. An interesting example is provided, in a smart home environment, where the doorbell rang and the resident does not respond within 5 minutes. The AmI systems detects that the resident is at home and, based on rules, decides to alert the resident visually or by telephone (Cook, Augusto, & Jakkula, 2007, p. 10).

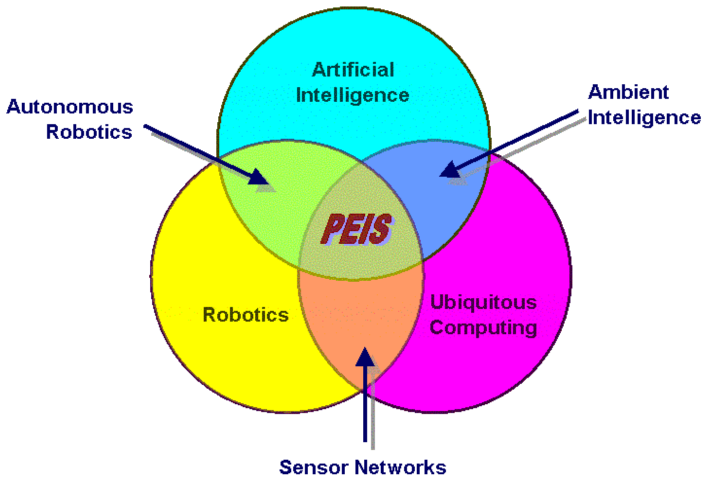


Figure 2.6: The study of PEIS-Ecologies lays at the intersection of several different fields

2.1.3 Acting

Sensing the physical environment and reasoning using intelligent devices which are a mechanism by which AmI systems can execute actions and affect the system users. Another mechanism is through robots (Cook, Augusto, & Jakkula, 2007, p. 11). Research in robotics has evolved and provides a wide range of assistive tasks and support to AmI. An interesting approach which ties AmI and Autonomous Robotics is Physically Embedded Intelligent Systems (PEIS). An illustration of the fields included in this approach is presented in Figure 2.6 (Saffiotti & Broxvall, 2005, p. 2).

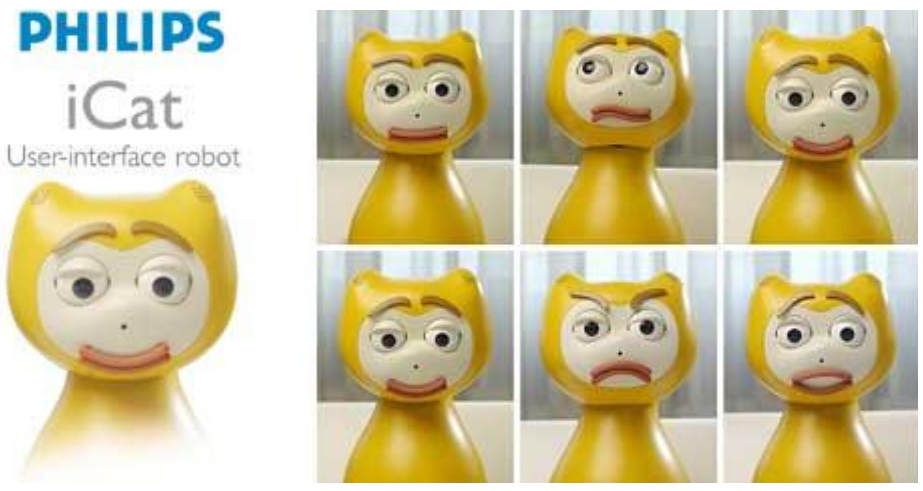


Figure 2.7: Philips iCat

Robots provide self-mobility and human-likeness to AmI systems. In recent years robots detect face expressions or even generate emotions or expressions. An example is PHILIPS iCat as shown in Figure 2.7. iCat is a 38cm tall robot containing 13 servos which control different parts of the face (eyebrows, eyes, eyelids, mouth and head position). As a result iCat can generate different expressions (happy, surprise, angry, sad). These expressions are needed to create social human-robot interaction dialogues (PHILIPS, 2005).

Another interesting example is the Dustbot system which consist a network of autonomous robots. These robots are part of an AmI system, are found in pedestrian areas in city centre and are designed to execute the following tasks: street cleaning; household garbage collection; air quality monitoring (Dario, Mazzolai, & Laschi, 2011). A robot of the Dustbot system is shown in Figure 2.8.

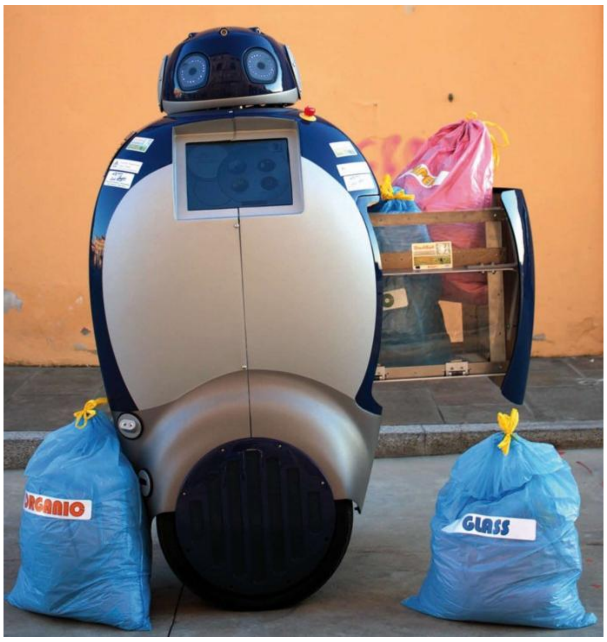


Figure 2.8: Autonomous Robot in Dustbot

2.1.4 Human-Computer Interaction

In regards to the Human-Computer interaction AmI should be easy to live with. Therefore the computer interfaces should be human-centric. These interfaces should be context aware and natural.

> *Context-aware systems are able to adapt their operations to the current context without explicit user intervention and thus aim at increasing usability and effectiveness by taking environmental context into account. Particularly when it comes to using mobile devices, it is desirable that programs*

and services react specifically to their current location, time and other environment attributes and adapt their behavior according to the changing circumstances as context data may change rapidly. (Baldauf, Dustdar, & Rosenberg, 2007, p. 263)

The other aspect of HCI deals with natural interfaces. The system should use its intelligence to analyze the situations and the user needs from previous activities and help when needed. There is a significant progress in AmI systems over the last few years. Even so a part of this progress will remain unused if the technologies are not natural and difficult to use by users (Cook, Augusto, & Jakkula, 2007, p. 12;13).

2.1.5 Privacy and Security

Since AmI systems, in most cases, share, process and store personal data regarding their users, privacy and security concerns are raised. At the sensor level, the sensor reliability, handling errors and installation errors can create security risks. To ensure security in AmI systems the designer should consider the combination of these risks with the sensor network communication channel reliability and security. In regards the privacy it differs regarding different users. In some cases privacy is more important than the benefits of the system. Therefore privacy preferences should be available to the user (Cook, Augusto, & Jakkula, 2007, p. 15).

2.1.6 AMI Applications

AmI systems, as shown in Figure 2.9, involve different devices of every-day use. As mentioned previously, these devices could be considered as “Smart Devices” since computing abilities have been added to them. A network interconnects these devices while processing can be done in a centralized or distributed way. There are many AmI applications in different fields. AmI systems are implemented in: Smart Homes; Hospitals; Transportations; Workplaces; Education etc.

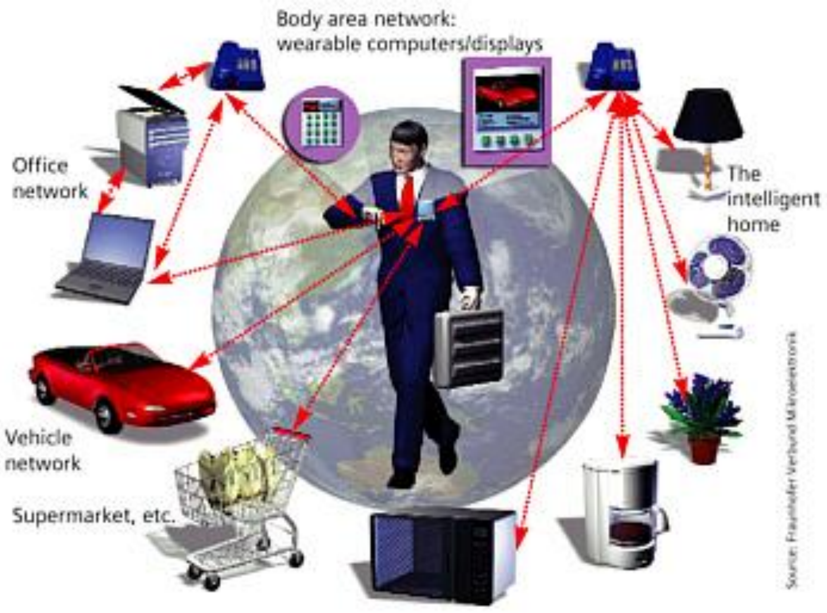


Figure 2.9: AmI Devices and users

Smart Homes

Smart Homes are usually equipped with sensors, actuators and processing devices. Mainly the sensors and the actuators are implemented into electro domestics (e.g. oven, refrigerator etc.), household items (e.g. beds, taps etc.) and temperature handling devices (e.g. radiators and air conditioners). These sensors collect data from the resident's activities, these data are mined and patterns are identified. Afterward, with the use of actuators, the system decides to act in order to facilitate the residents and benefit them in many ways. The benefits can be in: comfort (e.g. adjusting the temperature automatically), economy (e.g. reduce power consumption by controlling the lights or other unused devices), safety (e.g. after observing the lifestyle of the residents, the systems could detect possible harmful situations) (Cook, Augusto, & Jakkula, 2007, p. 16).

A remarkable example of a smart home is The Gator Tech Smart House, shown in Figure 2.10 . This project was developed by the University of Florida in the Mobile and Pervasive Computing Research directed by Dr. Sumi Helal.

Figure 2.10: THE Gator Tech Smart House

This house contains many interesting features. Some of these are:

- *Smart Mailbox:* The mailbox senses the arrival and alerts the resident
- *Smart Front Door:* A Front door that facilitate keyless entry using RFID technology. The front door is also equipped with a microphone, a Camera, an LCD display and an automatic door opener.
- *Smart Floor:* The floor senses the position of each habitant. The developers are also developing a feature to be able to detect if a habitant falls and report it to the emergency services.
- *Smart Plugs:* As shown in Figure 2.11 the system detects if a specific plug, of an electrical appliance, is connected in the power network using RFID technology.

A more detailed presentation of Gator Tech Smart House is provided by Helal et al (Helal, Mann, El-Zabadani, King, Kaddoura, & Jansen, 2005). Regarding the Smart Home there are also other examples available like: MavHome[1], iDorm[2], Aware Home[3], Domus Lab, etc.

[1] MavHome: http://ailab.wsu.edu/mavhome/
[2] iDorm: http://cswww.essex.ac.uk/iieg/idorm.htm
[3]Aware Home: http://awarehome.imtc.gatech.edu/

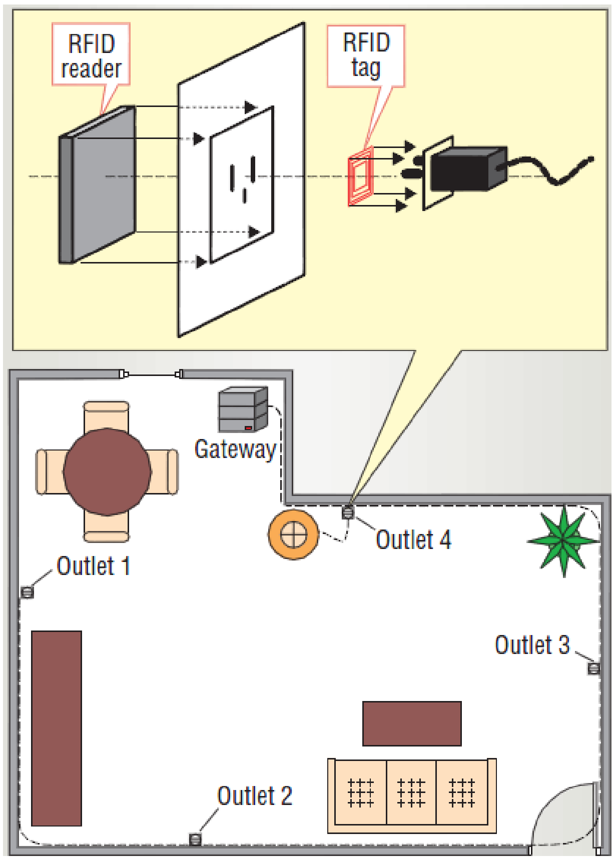


Figure 2.11: Smart Plugs

Transportations

Since a significant part of people's lives is spent traveling, AmI research is focused both in transportation's infrastructure management and in improving the user's experience. Train stations, airports, busses, cars could be equipped with sensors and provide information about how the system operates and how it is performing. These data could be processed in order to apply preventive actions regarding security and increase the experience of people using the system more effectively (Cook, Augusto, & Jakkula, 2007, p. 21). These AmI systems could use data from surveillance cameras or GPS systems and exploit the spatial information in order to assist the user and increase security. Intelligence systems can be also implemented in buses, cars, etc. to improve the user experience or even to increase safety.

An early example of an automatic system is the Electronic Stability Program (ESP). This system is already implemented in modern cars in order to assist the driver in difficult situations (e.g. executing tricky maneuvers).

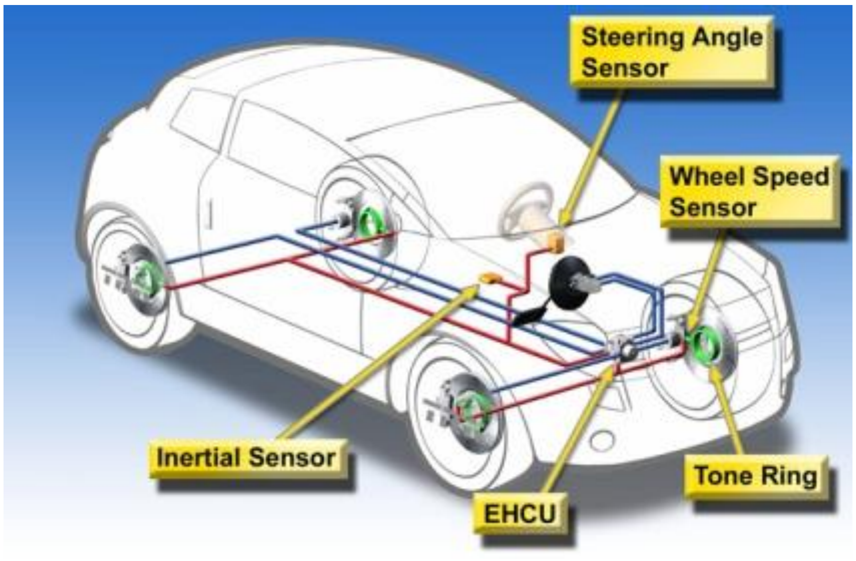


Figure 2.12: ESP Sensors

As shown in Figure 2.12 the electronic stability control systems monitors each wheel’s speed, the angle of the steering wheel, and the spatial acceleration using an accelerometer. The data provided by these sensors are processed and the system acts by braking separately the required wheel. In details the steering angle sensor provides information about the intentions of the driver and the desired direction. Afterwards the accelerometer in combination with the wheel speed sensors provides information about where the car is actually going. If the car is not responding to the driver’s intentions then the system brakes separately the required wheel in order to bring the car in the desired trajectory.

A more advanced and intelligent system should be able to process more parameters like the driver’s health information, the car’s condition, location (e.g. GPS) and the environmental conditions. A research of such a system that deals with the previously mentioned parameters is In-Vehicle Ambient Intelligent Transport System (I-VAITs) (Rakotonirainy & Tay, 2004). Research of an AmI system, in order to support in the management in the transport infrastructure is presented by De Amicis et al (De Amicis, Conti, Piffer, & Prandi, 2011). The architecture of such a system is shown in Figure 2.13. Another similar system is also implemented by the Organization of Urban Transportation of Thessaloniki. This system implements GPS technology in buses and their position, in combination with other data, is collected (using GPRS) and processed. The data retrieved are used to assist in the management of the system and also to facilitate individuals using the urban transport (e.g. showing the estimated time for the bus arrival).

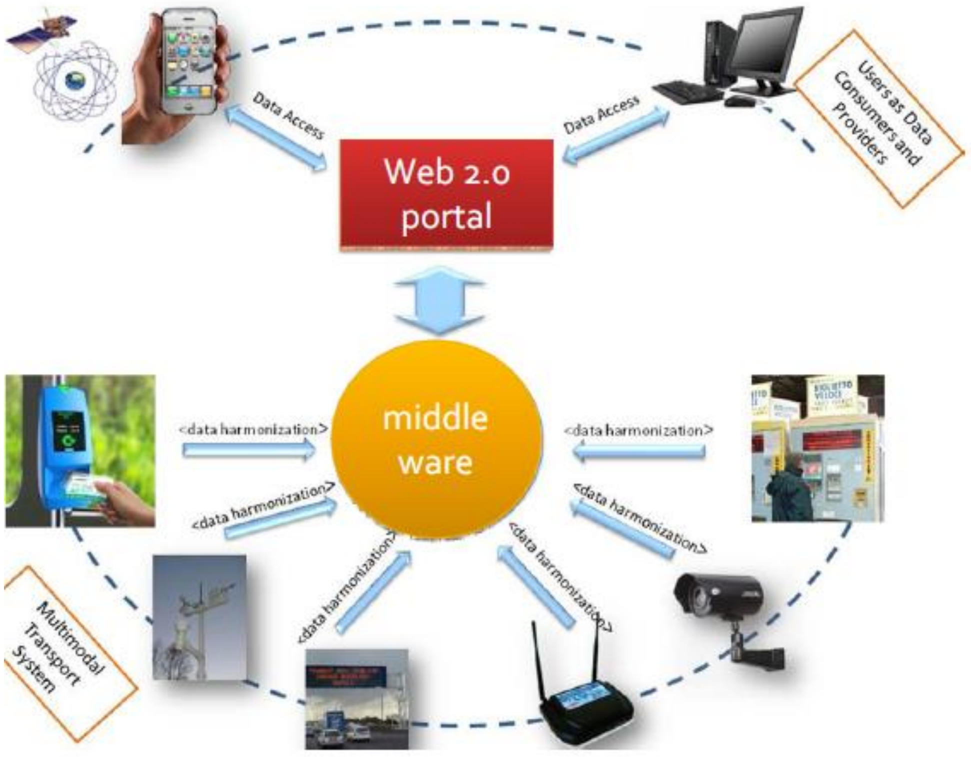


Figure 2.13: Architecture of an AmI system which supports management in transportation infrastructure

Education

AmI systems can improve the learning experience of students. Such systems include interactive computer interfaces which adapt according to the students and the studying material and additionally providing optic – acoustic material (i.e. video sound). Figure 2.14 indicates an interactive board. Many research projects have focused in this direction like: the Georgia Tech Classroom 2000, the intelligent classroom at Northwestern University etc. (Cook, Augusto, & Jakkula, 2007, p. 22).

Figure 2.14: Interactive board

2.2 Wireless Sensor Networks

The use of sensors in environments, structures, industry etc. provides useful information. These sensors provide data about physical quantities. In most of the cases these data need to be transferred in a safe and cost efficient way. Wired networks (i.e. cable or fiber optics) provide this type of connection but there are many disadvantages in using them. The main disadvantages are high installation and long term maintenance cost, breakage and connector failures. The installation cost could limit the number of sensors placed and reduce the quantity of information retrieved. Wireless sensor networks can eliminate these costs providing easier installation. Usually sensor networks are scalable; have low energy consumption; provide fast data acquisition; are reliable and accurate; and have low cost both for acquisition, installation and maintenance (Wilson, 2005, p. 439).

A generic view of many sensor networks interconnected with other networks is shown in Figure 2.15, (Wilson, 2005). Most of the existing applications include interconnection with other networks (e.g. internet) since the information, provided by sensor networks is needed to be accessed from different locations.

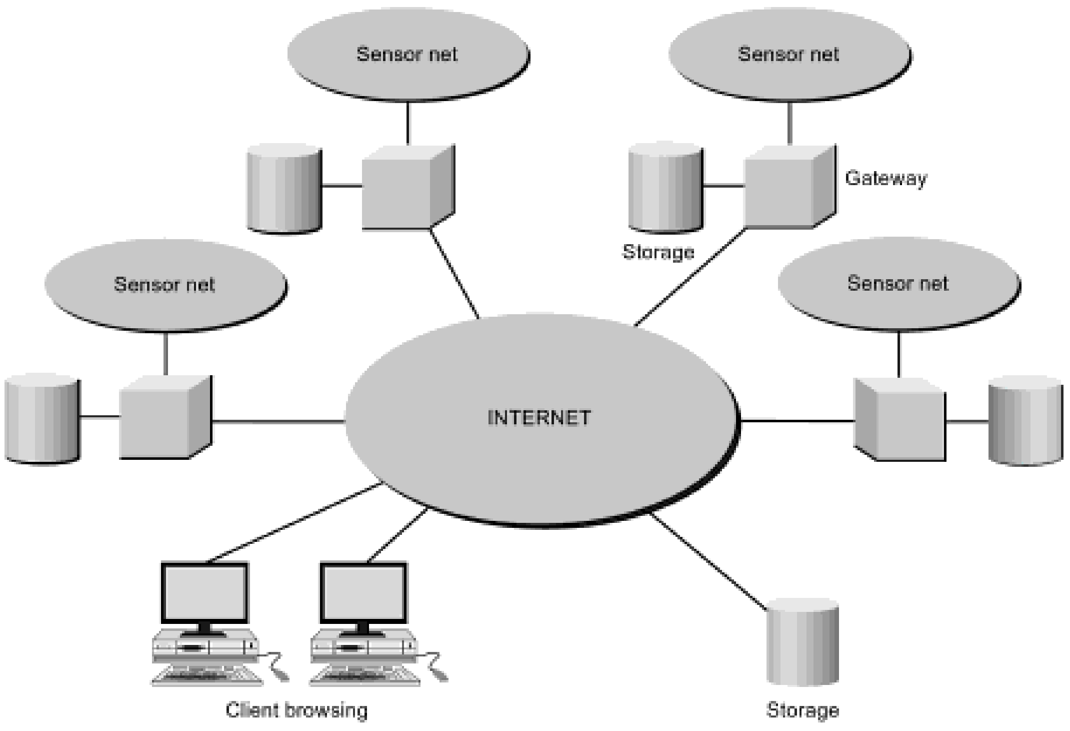


Figure 2.15: Sensor Networks Interconnected

Figure 2.16 indicates an example of a single sensor network. Often there are also actuators involved in sensor networks to be able to control devices included in the network.

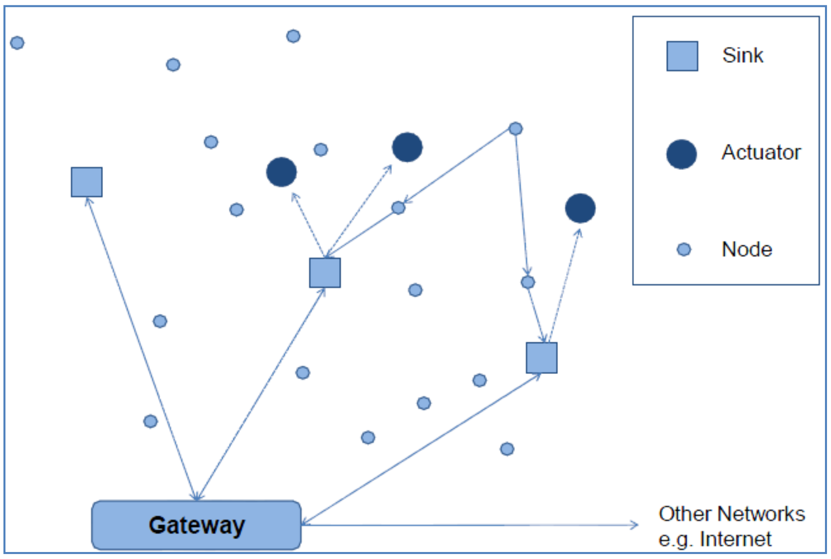


Figure 2.16: Example of a Sensor and Actuator Network

2.2.1 Wireless Sensor Nodes

Figure 2.17 indicates the block diagram of a wireless sensor node. An actual picture of a Zigbee wireless sensor node is shown in Figure 2.18.

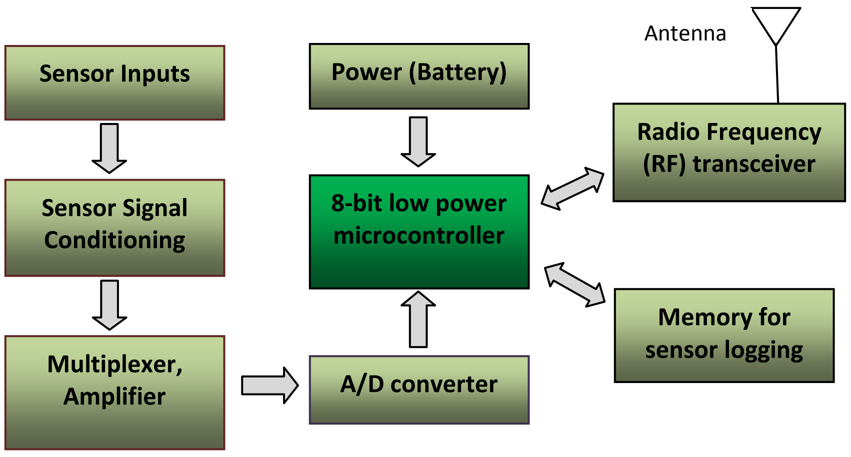

Antenna
Sensor Inputs
Power (Battery)
Radio Frequency (RF) transceiver
Sensor Signal Conditioning
8-bit low power microcontroller
Memory for sensor logging
Multiplexer, Amplifier
A/D converter

Figure 2.17: Sensor Node Block diagram

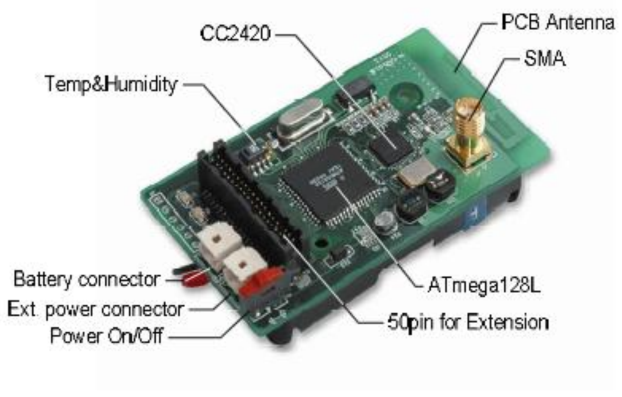


Figure 2.18: ZigBee Sensor node

The signal conditioning block contains electronic circuits which are responsible for transforming the output of the sensor (i.e. resistance, capacitance, voltage, current etc.) into an analog signal that meets the requirements of the next stage circuits. The sensor signal conditioning block can be programmed or replaced according to the sensor used. In case there are multiple sensors the analog signals retrieved are multiplexed. Afterward the cumulative analog signal is converted into a digital signal, using an analog to digital converter, and it is then processed by the microprocessor or even stored. The microprocessor is responsible for: managing the data collection from the sensors (e.g. transmit or store the data), performing power management functions, interfacing the sensor data to the physical radio layer, and managing the radio network protocol (Wilson, 2005, p. 440).

2.2.2 Sensors

Electronic sensors are devices which convert physical quantities into electrical signals (voltage or current). There are different kinds of sensors which sense temperature, humidity, pressure, light, radiation, smoke, movement etc. A brief description about sensors and their outputs is shown in Table 2.1 provided by (Weber, Vickery, & OECD, 2009, p. 8) .

Table 2.1: TYPES of Sensors and their outputs

Physical Property	Sensor	Output
Temperature	Thermocouple	Voltage
	Silicon	Voltage/Current
	Resistance temperature detector (RTD)	Resistance
	Thermistor	Resistance
Force/Pressure	Strain Gauge	Resistance
	Piezoelectric	Voltage
Acceleration	Accelerometer	Capacitance
Flow	Transducer	Voltage
	Transmitter	Voltage/Current
Position	Linear Variable Differential Transformers (LVDT)	AC Voltage
Light Intensity	Photodiode	Current

2.2.3 Network Topologies

Networks are installed in different environments, with different economic considerations and Quality of Service (QoS). Due to this diversity several basic network topologies have been developed. These basic topologies are shown in Figure 2.19 (LEWIS & Grant, 2004, p. 2).

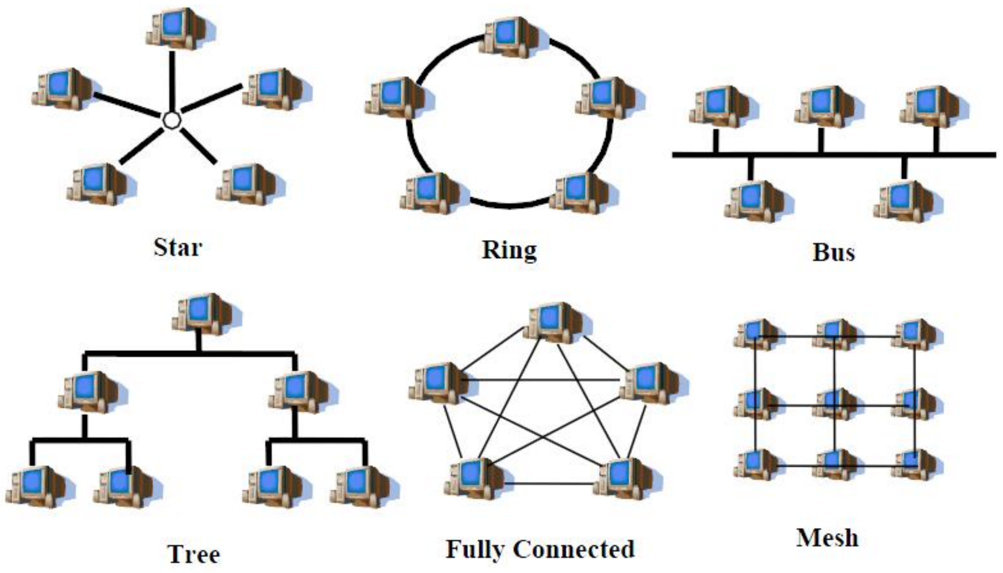


Figure 2.19: Basic Network Topologies

There are also hybrid combinations of these basic topologies. Sensor networks use these topologies depending on the IEEE standard used. In general networks are defined according to their coverage. Therefore the most commonly known are Personal Area Networks (PAN), Local Area Networks (LAN) and Wide Area Networks (WAN).

2.2.4 OSI Model

The simplified Open Systems Interconnection (OSI) model consists of five layers, the application, transport, network, data link, and physical layer. Figure 2.20 indicates the hierarchy of these layers with the application layer on top and the physical layer in the bottom. In this section the protocols used in these five layers will be briefly described in regards to wireless sensor networks.

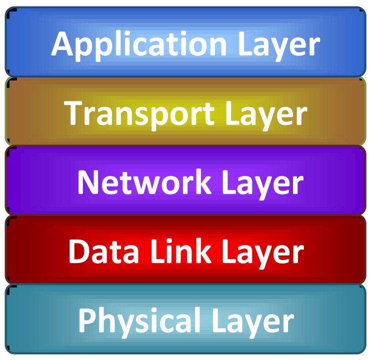


Figure 2.20: Simplified OSI model

Application Layer

Even though many there are many applications proposed and defined, application layer protocols remain largely unexplored. Three possible protocols are: Sensor Management Protocol (SMP), Task Assignment and Data Advertisement Protocol (TADAP), and Sensor Query and Data Dissemination Protocol (SQDDP) (Raghavendra, Sivalingam, & Znati, 2006, p. 25).

Sensor Management Protocol (SMP). SMP is used by system administrators in order to interact with sensor networks. Usually the nodes included in sensor networks do not have global identification. For this reason, SMP access the nodes using attribute-based naming (e.g. “the areas where the temperature is higher than 30^{o}C” is a more common query than “the temperature read by a certain node”) and location-based addressing (e.g. “the temperature in the basement” rather than “the temperature in a certain node”).

SMP is used to perform the following tasks (Raghavendra, Sivalingam, & Znati, 2006, p. 26)*:*

- *Defines the rules for data aggregation, attribute-based naming and clustering to the sensor nodes,*
- *Exchanging data related to the location finding algorithms,*
- *Time synchronization of the sensor nodes,*
- *Turning Sensor nodes on and off,*
- *Querying the sensor network configuration and the status of the nodes, and re-configuring the nodes*
- *Authentication, key distribution and security in data communications.*

Task Assignment and Data Advertisement Protocol: The users send their interest about a certain attribute, a phenomenon or an event to a sensor node, a subset of nodes or to the entire sensor network. This protocol is used by the nodes in order to advertise their available data to the users and the users query the data they are interested in (Raghavendra, Sivalingam, & Znati, 2006, p. 26).

Sensor Query and Data Dissemination Protocol: This protocol provides to the user the interfaces to perform queries respond to queries and collect the incoming data (Raghavendra, Sivalingam, & Znati, 2006, p. 26).

Transport Layer

The main objectives of the transport layer are: to bridge application and network layers by application multiplexing and demultiplexing; to provide data delivery between the source and the sink providing error control proportional to the requirements of the application layer; and regulate the ingress traffic on the network by implementing flow and congestion control mechanisms (Raghavendra, Sivalingam, & Znati, 2006, p. 27). These mechanisms must be modified in order to cover the unique characteristics of the WSN. Since there are hardware and power constrains it is difficult to use the same flow and congestion control mechanisms used by other protocols (e.g. TCP). The development of a transport protocol for WSN is also influenced by the fact that WSN are deployed in specific sensing applications, sensors and controlling actuators, in many sectors like health, environment, military etc. Depending on the application the congestion control and the security mechanism may differ (Raghavendra, Sivalingam, & Znati, 2006, p. 27;28).

To fulfill the main objectives, a transport control protocol for WSN must provide (Raghavendra, Sivalingam, & Znati, 2006, p. 28):

- *Reliable Transport: A*ssuring the proper function of the sensor network,
- *Congestion Control*: This is related to reliable transport since packet loss can affect the efficiency of the network and the available resources (e.g. power)
- *Self Configuration*: These protocols must be adaptive to dynamic topologies caused by node failure, temporary power down, mobility, etc.
- *Energy awareness*: The main objectives (e.g. congestion control) of the protocol must be achieved by consuming the minimum possible energy.
- *Biased Implementation*: The algorithms must run on the sink with minimum functionality to the sensor node since there are more available resources (i.e. power, processing power) to the sink in comparison to the sensor node.
- *Constrained Addressing/Routing*: It is preferable to have attribute-based naming and data-centric routing.

In order to have data integrity and retrieve reliable information from the sensors there is the need for a reliable connection between the sink and the physical event (i.e. the sensor). Therefore there is the need for retransmissions and acknowledgment mechanisms. Since the sensors operate on batteries, negative acknowledgements would be preferable because the energy consumed would be minimal. The transport protocol should be energy aware.

There are many transport control protocols developed for sensor networks. Some of them are: Congestion Detection and Avoidance (CODA), Event-to-Sink Reliable Transport (ESRT), Reliable Multi-Segment Transport (RMST), Pump Slowly Fetch Quickly (PSFQ), GARUDA, Tiny TCP/IP, Sensor TCP(STCP), SenTCP, Trickle, FUSION, Asymmetric and Reliable Transport (ART), Congestion Control and Fairness (CCF), Priority-based Congestion Control Protocol (PCCP), etc. A classification and description of these protocols is provided by Rahman et al. (Rahman, El Saddik, & Gueaieb, 2008).

Network Layer

In general, the network layer is mainly responsible for routing the packets choosing the route based on the routing algorithms. In sensor networks the routing protocol should be data-centric, this requires attribute-based naming. In the attribute based naming the user requests an attribute of a phenomenon, rather than querying and individual node. In a data-centric protocol data aggregation is important. Data aggregation could solve the overlap problems. An example of data aggregation is shown in Figure 2.21. In this example sensor node E aggregates the data from sensor node A, B and sensor node F from sensor node C, D. Afterwards the sensor node G aggregate the data aggregation of sensor node E and F. Last the data aggregation is sent to the sink. Data aggregation should be performed carefully because of the specifics of the data (e.g. the location of the sensor node or the type of data sent) (Raghavendra, Sivalingam, & Znati, 2006, p. 33). The network layer should also provide interconnection with other networks (e.g. internet). The sinks could be used as gateways for the sensor nodes.

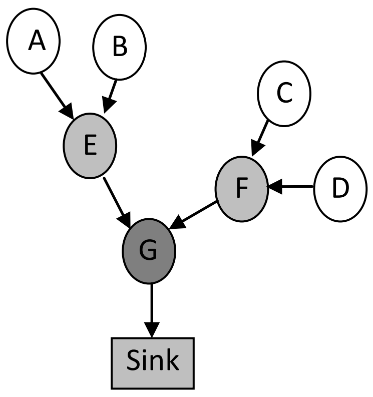


Figure 2.21: Example of Data Aggregation

Some of the principles when developing a routing protocol in wireless sensor networks are: power efficiency, data aggregation, data-centric networks, and interconnection with other networks. An example, shown in Figure 2.22, was provided by Raghavendra et al 2006 p,35 , this example helps describing the different approaches in order to choose a route in case there are more than one. The possible routes are shown in Table 2.2.

Table 2.2 Possible routes between the sink and a node T

Possible Routes	Description
🟥 Route1	Sink-A-B-T, total PA=4, total a=3
🟩 Route2	Sink-A-B-C-T, total PA=6, total a=6
🟦 Route3	Sink-D-T, total PA=3, total a=4
🟪 Route4	Sink-E-F-T, total PA=5, total a=6
PA is the available power, a_i is the energy consumed to transmit a data packet in link *i*	

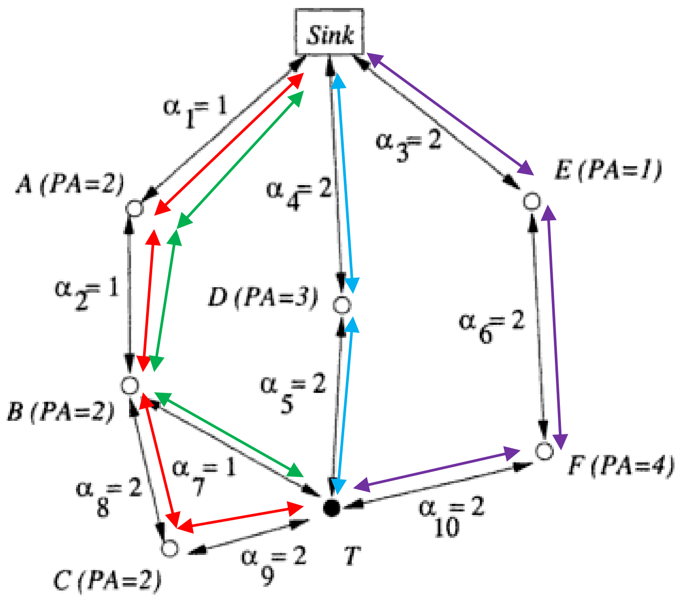


Figure 2.22: Route Selection example

Some approaches when choosing a route are (Raghavendra, Sivalingam, & Znati, 2006, p. 35) :

- *Maximum available power (PA) route*: The preferable route is the one with the maximum power available. The total PA is provided by the sum of the PA of each node. In the previous example Route2 has the higher PA, but since includes all the nodes in Route1 plus node C, this is not the more power efficient route because in C PA=2. Therefore Route2 is eliminated. The chosen route is Route4 from the remaining 3 routes.

- *Minimum energy (ME)* route: The route with minimum energy consumption is preferred. In Figure 2.22, Route1 is the route with minimum energy consumed (i.e. the total a=3). In case there are two routes with the same minimal energy consumption, the route with minimum number of hops will be chosen.
- *Minimum Hop (MH)* route. The route will the minimum number of hops is preferred. In the example Route3 will be selected.

There are many routing protocols in WSNs but most of them were not implemented and a part of them are in the developing stage. These protocols could be classified based on many parameters (e.g. Network Structure, Protocol Operation, Packet Destination etc.) (Dwivedi & Vyas, 2010, p. 31).

Network Structure

Regarding to the network structure the WSN protocols can be further classified in (Dwivedi & Vyas, 2010, p. 31):

- Flat-based or Data Centric routing. Some examples in this category are: Directed Diffusion, Minimum Cost Forwarding Algorithm, Coherent/No coherent Processing, Sensor Protocols for Information via Negotiation (SPIN), Rumor Routing, Stream Enable Routing etc.
- Hierarchical-based or Cluster based routing. Some examples are: Simple Hierarchical Routing Protocol, Low energy Adaptive Cluster Hierarchy, Power Efficient Gathering in Sensor Information System, Self-Organizing Protocol, Geographic Adaptive Fidelity etc.
- Location-based routing. Some examples are: Minimum Energy Communication Network, Geographic Adaptive Fidelity, Geographic and Energy Aware Routing etc.

Protocol Operation

Based on the protocol operation the following classification can be done (Dwivedi & Vyas, 2010, p. 32): Multipath-based routing, Query-based routing, Negotiation-based routing, QoS-based routing, Non-coherent & Coherent data-processing based routing etc..

Packet Destination

In regards to the packet destination the routing protocols can be classified as follows (Dwivedi & Vyas, 2010, p. 32): Gossiping and agent-based unicast forwarding, Energy-efficient unicast, Broadcast and multicast, Geographic routing, Mobile nodes.

Data Link Layer

The Data Link Layer receives a data stream from the network layer and is responsible for transmitting these data to the next hop. In this layer the data is divided into frames and a checksum is computed. The checksum is also computed by the receiver and if it does not correspond to the previous an error has occurred and the receiver sends an error report.

The Medium Access Control (MAC) layer is a sub layer of the Data Link Layer (DLL). Problems in medium access are influenced by attributes like Energy efficiency, Collision Avoidance, Scalability, Channel Utilization, Latency, Throughput and Fairness (Roy & Sarma, 2010, p. 2). A MAC protocol for WSNs must be able to manage power conservation, mobility and failure recovery strategies. Since most of the nodes operate on batteries, energy efficiency is a crucial factor in WSNs. MAC protocols must minimize the energy consumption due to radio operations (i.e. send receive and sense the channel) to the sensor nodes. Another fundamental task of a MAC protocol is to avoid collisions between interfering nodes transmitting in the same time. Therefore multiple access techniques are used. Some examples of these techniques are Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA), Code Division Multiple Access (CDMA), Carrier Sense Multiple Access (CSMA), Multiple Access with Collision Avoidance (MACA), etc. An interesting classification, shown in Figure 2.23, was provided by CONET in 2009 in their report about WSN standards.

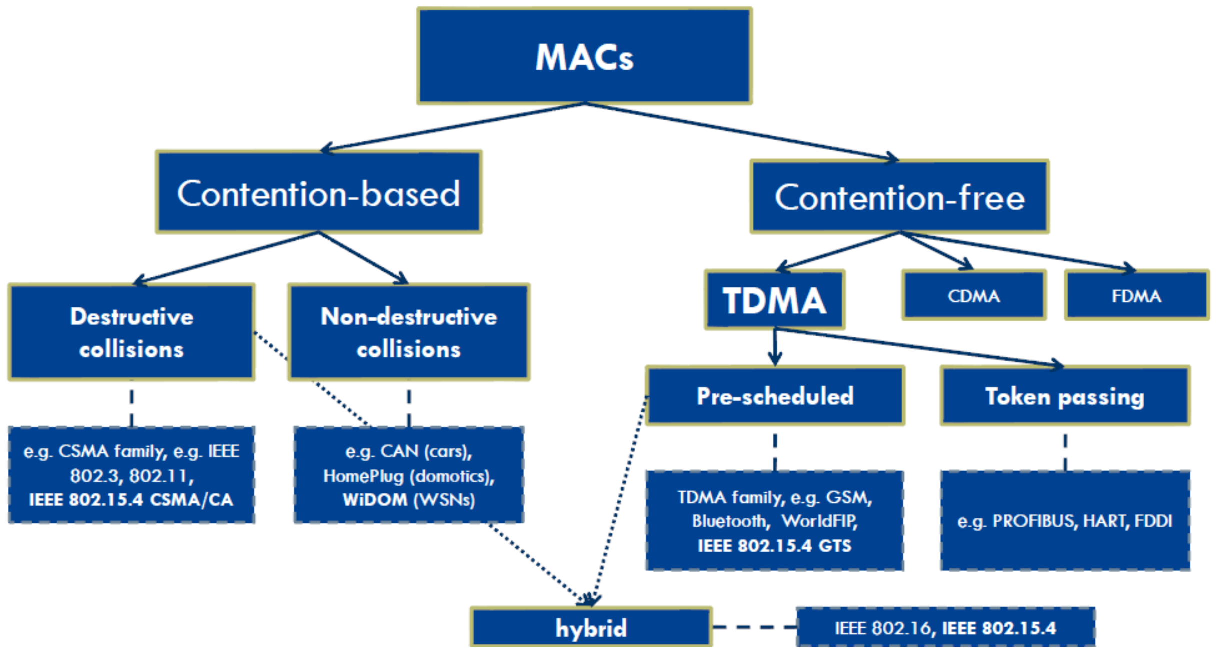

MACs
Contention-based
Contention-free
Destructive collisions
Non-destructive collisions
TDMA
CDMA
FDMA
Pre-scheduled
Token passing
e.g. CSMA family, e.g. IEEE 802.3, 802.11, IEEE 802.15.4 CSMA/CA
e.g. CAN (cars), HomePlug (domotics), WiDOM (WSNs)
TDMA family, e.g. GSM, Bluetooth, WorldFIP, IEEE 802.15.4 GTS
e.g. PROFIBUS, HART, FDDI
hybrid
IEEE 802.16, IEEE 802.15.4

Figure 2.23: MAC Classification

A qualitative overview of MAC protocols is shown in Table 2.3 (Raghavendra, Sivalingam, & Znati, 2006, p. 38)

Table 2.3: Qualitative overview of MAC protocols

MAC Protocol	**Channel Access**	**Features and Advantages**
SMACS	Fixed allocation of duplex time slots at fixed frequency	✓ Exploit large available bandwidth compared to sensor data rates. ✓ Random wake up during setup and turning radio off while idle.
Hybrid TDMA/FDMA	Centralized frequency and time division	✓ Optimum number of channels for minimum system energy. ✓ Hardware based approach for system energy minimization.
CSMA based	Contention based random access	✓ Application phase shift and pre-transmit delay. ✓ Constant listening time for energy efficiency.

Physical Layer

The physical layer consist the basic transmission technologies of the network. The data or the packets are sent in the physical medium (i.e. air). In this layer many parameters are defined (e.g. frequency selection, modulation/demodulation schemes,). In regards to the frequency selection, in the frequency spectrum, different frequency bands are allocated to different users (e.g. cellular phone communications, TV broadcasting, military communications etc.) according to their license. Bands which could be used without a license are the ISM (Industrial, Scientific, and Medicine) bands. The ISM frequency bands defined by the FCC and the transmission power limits are shown in Table 2.4.

Table 2.4: ISM Bands and transmission power limits

ISM Bands	Examples	Power Limit (Watts)
902 - 928 MHz	Cordless Phones	1W
	Microwave Ovens	750W
	Industrial Heaters	100W
	Military Radar	1000kW
2.4 - 2.4835 GHz	Bluetooth	100mW
	Wi-Fi 802.11b/g	1W
	Microwave Ovens	900W
5 GHz	Wi-Fi 802.11a/n	
5.15 - 5.25 GHz		200mW
5.25 - 5.35 GHz		1W
5.47 - 5.725 GHz		1W
5.725 - 5.825 GHz		4W
60GHz	57 - 64 GHz	

The modulation transforms the given signal, which transfers the data, in a high frequency signal using a carrier frequency. There are many available modulation techniques. Some digital modulation techniques are: Frequency Shift Keying (FSK), Amplitude Shift Keying (ASK) and Phase Shift Keying (PSK).

There are many difficulties related to the radio link characteristics (e.g. Link asymmetry, non-isotropic connectivity etc.), and to propagation phenomena (e.g. reflection, diffraction, scattering), etc.

2.2.5 Standards

Wireless Local Area Networks (WLAN) and Wireless Personal Area Networks (WPAN) are respectively based on the IEEE 802.11 and IEEE 802.15 standard families. The 802.11 based standards offer high data rates in the order of tens/hundreds Mbps and ranges in the order of tens/hundreds of meters. On the other hand the 802.15 based standards provide data rates in the order of hundreds Kbps up to several Mbps with ranges from few meters up to hundreds of meters. In order to provide high data rates and range the 802.11 based standards have higher energy consumption (Christin, Mogre, & Hollick, 2010, p. 98). Figure 2.24 indicates the 802 protocols in comparison to their range and data rate. In wireless sensor networks the sensor nodes are either powered by cables or batteries. In case the sensor nodes are powered with cables the 802.11 based standards could be exploited providing the previously mentioned advantages. In case the nodes are powered by batteries the energy must be used conservatively in order to avoid the frequent battery replacement/recharging. Therefore in these cases the use of the 802.15 based standards is more preferable, and specifically the 802.15.1-2 and 802.15.4 standard.

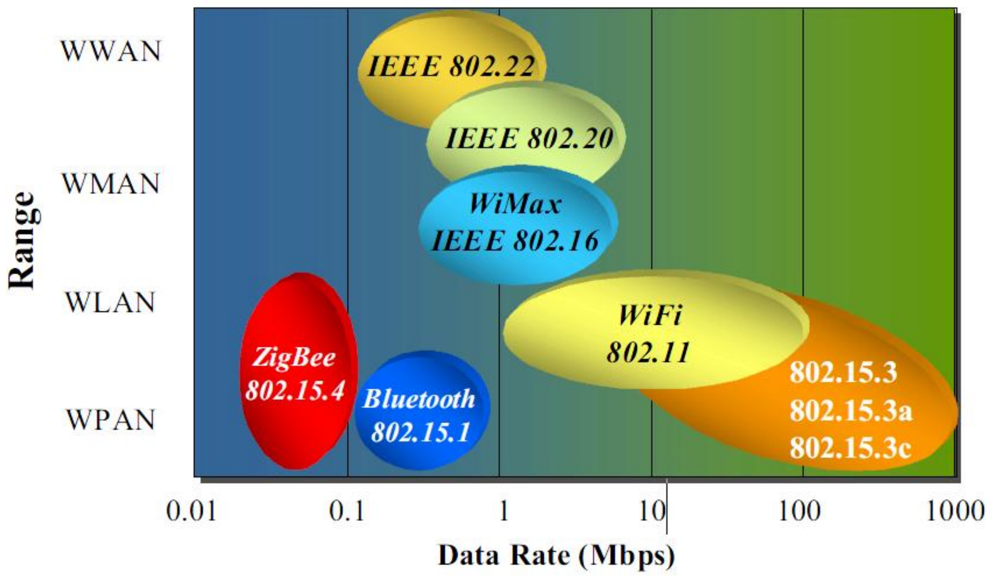


Figure 2.24: 802 Protocols Range and Data Rate

IEEE 802.15.1 Based Standards

The IEEE 802.15.1 standard also known as Bluetooth v1.0 could be classified in between the 802.11 and 802.15.4 regarding the energy consumption and data rates. Therefore this standard is partially suited for applications which require high data rates and strong real time requirements (e.g. factory automations) (Christin, Mogre, & Hollick, 2010, p. 99). The operation frequencies and the RF Channels in each region/country are shown in Table 2.5.

Table 2.5: Bluetooth, OPERATING frequency bands

Geography	**Regulatory Range**	**RF Channels**
USA	2.400-2.4835 GHz	f=2402+k MHz, k=0,…,78
Europe (except Spain and France)	2.400-2.4835 GHz	f=2402+k MHz, k=0,…,78
Spain	2.445-2.475 GHz	f=2449+k MHz, k=0,…,22
France	2.4465-2.4835 GHz	f=2454+k MHz, k=0,…,22
Japan	2.471-2.497 GHz	f=2473+k MHz, k=0,…22

Bluetooth v1.0 uses Gaussian Frequency-Shift Keying (GFSK) modulation and uses a radio technology called Frequency-Hopping Spread Spectrum. In USA and Europe (except France and Spain) Bluetooth uses 79 bands of 1MHz each from 2.400 to 2.4835 GHz (Bluetooth Specification Version 1.0A, Part A : Radio Specification, 1999). Bluetooth is packet based, has a star topology and supports up to seven nodes communication with a single base station. Even though some companies have implemented Bluetooth in some wireless sensor applications, they have not met with wide acceptance due to limitations. Some limitations of Bluetooth are (Wilson, 2005, p. 443):

- *Relatively high power for a short transmission range*
- *Nodes take long time to synchronize to network when returning from sleep mode. This increases the average system power.*
- *Low number of nodes per network (<=7)*
- *Medium access control (MAC) layer is overly complex when compared to that required for wireless sensor applications.*

IEEE 802.15.4 Based Standards

The 802.15.4 based standards have a lower data rates and energy consumption in comparison to Bluetooth. This standard is suitable for applications which have infrequent exchanges of small packets and energy consumption is an important issue (Christin, Mogre, & Hollick, 2010, p. 100). The physical layer in this standard operates in the 2.4 GHz frequency band as well as in the 868 MHz (Europe) and 915 MHz (North America) bands. The 2.4 GHz band is divided in 16 channels with a 250kbps maximum data rate and a 5 MHz gasp between the channels. In the 915 MHz band there are 10 channels with a 40 Kbps each with 2 MHz gasp between the channels. In the 868 MHZ band there is a single channel with a 20 Kbps data rate (Christin, Mogre, & Hollick, 2010, p. 100). The previously mentioned parameters are shown in Figure 2.25.

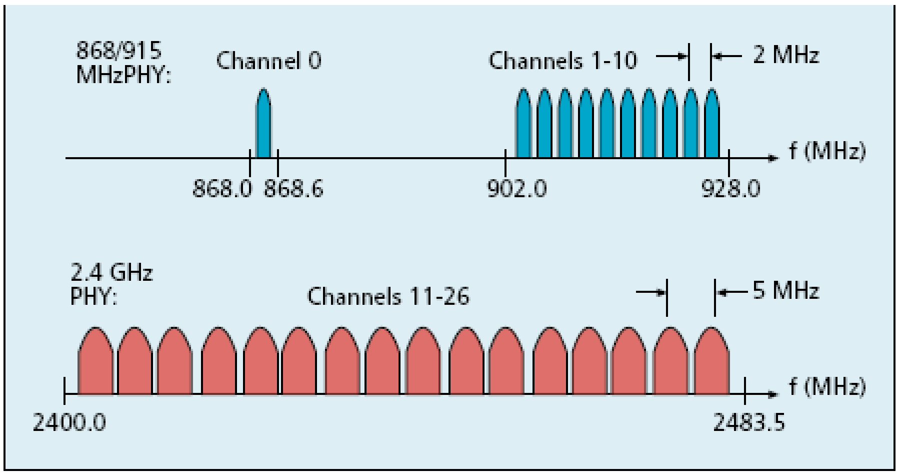


Figure 2.25: IEEE 802.15.4 channel Structure

The 802.15.4 standard specifies the following four physical layers (IEEE Computer Society, 2006) :

- 868/915 MHz direct sequence spread spectrum (DSSS) PHY employing binary phase-shift keying (BPSK) modulation
- 868/915 MHz DSSS PHY employing offset quadrature phase-shift keying (O-QPSK) modulation
- 868/915 MHz parallel sequence spread spectrum (PSSS) PHY employing BPSK and amplitude shift keying (ASK) modulation
- 2450 MHz DSSS PHY employing offset quadrature phase shift keying (O-QPSK) modulation

Table 2.6: 802.15.4 PHY Data and Spreading Parameters

	Frequency Band	**Data Parameters**			**Spreading Parameters**	
		Bit Rate (Kbps)	**Symbol Rate (Ksymbols/s)**	**Symbols**	**Chip Rate (Mchips/s)**	**Modulation**
868 MHz	868.0-868.6 MHz	20	20	binary	0.3	BPSK
915 MHz	902.0 – 928.0 MHz	40	40	binary	0.6	BPSK
2.4 GHz	2.4 – 2.4835 GHz	250	62.5	16-ary ortho-gonal	2.0	O-QPSK

Table 2.6 provides a more detailed description of this standard. Some examples of IEEE 802.15.4 based standards are: ZigBee, ZigBee Pro, 802.15.4e, WirelessHART, ISA100.11a etc.

ZigBee

ZigBee was developed by the ZigBee Alliance and was originally designed for home automation. ZigBee Pro was released in 2007 in order to cover the industrial automation's requirements. ZigBee Pro provides frequency agility in order to scan the available channels and choose the channel with less interference (Christin, Mogre, & Hollick, 2010, p. 105). The following paragraphs will describe the common parameters of ZigBee and ZigBee Pro. ZigBee can be used in large deployments since it can support hundreds of devices. The topologies used are star, tree and mesh. This standard is based in two device classes including Full-Function Device (FFD) and Reduced-Function Device (RFD). This standard proposes three different types of devices: ZigBee coordinator, ZigBee router and ZigBee end devices shown in Figure 2.26 (Christin, Mogre, & Hollick, 2010, p. 106).

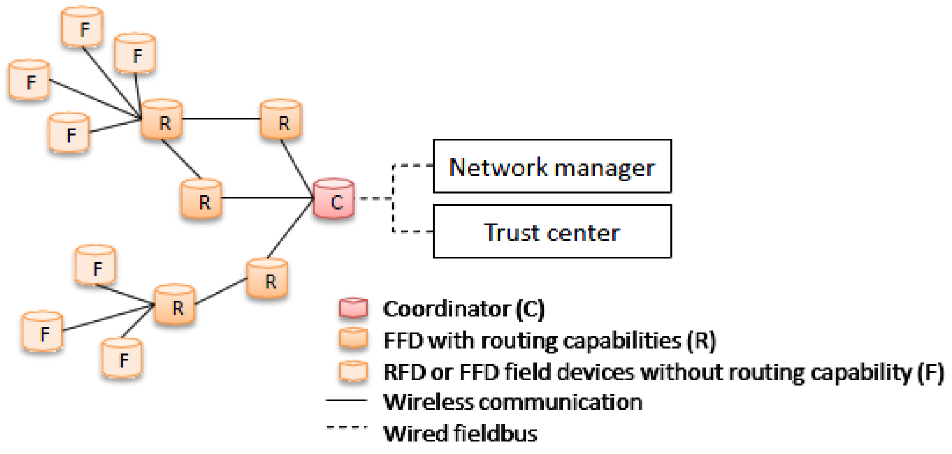


Figure 2.26: ZigBee network elements

As shown in Figure 2.26 the end devices can be FFD or RFD. FFD with routing elements are responsible for linking group of devices and supporting multi hop communication. A single FFD manages the network by supervising its formation, storing information and bridging it with other ZigBee networks (Christin, Mogre, & Hollick, 2010, p. 105). The ZigBee stack is based on the physical, DLL and MAC layer of the IEEE 802.15.4 standard, shown in Figure 2.27 (Wilson, 2005, p. 444). The Network and Application layers are specified by the ZigBee Alliance. Initially a common frequency for all the devices is selected, after this data transfers between the ZigBee devices are possible. ZigBee networks provide two types of data transmission mechanisms, with or without beacon.

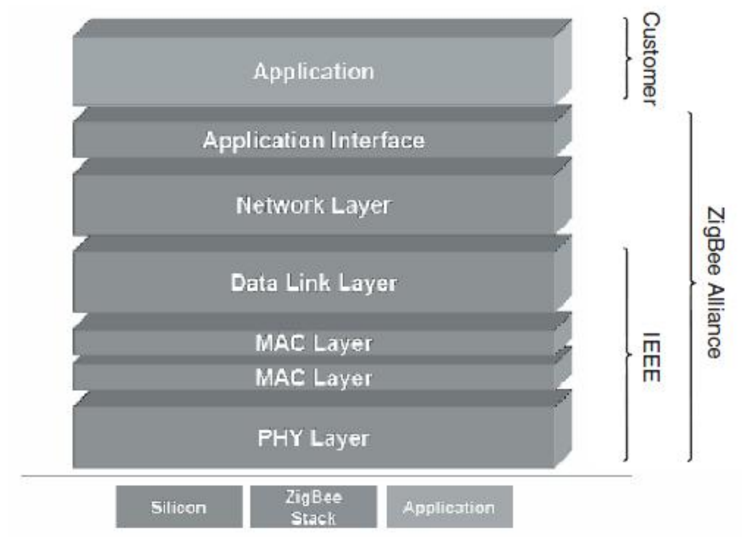


Figure 2.27: ZigBee Stack

With beacon: In this mode the FFD manages the communication from the end devices to the FFD by sending a first beacon to synchronize the sleeping phases of all the RFDs and announcing the superframe structure. The first part of the superframe is slotted and slotted CSMA/CA is used to access the channel, the second part is composed of slots which are reserved by the network coordinator for specific nodes. Initially the FFD announces the data transfer in the beacon to transfer data from FFD to RFD. Afterwards the RFD that has data to transmit sends a request to the FFD to begin transmission. In case the communication is between two FFD the mechanism is similar since one FFD acts as RFD (Christin, Mogre, & Hollick, 2010, p. 106).

Without beacon: In this mode there is no beacon and superframe transmitted. The channel is accessed using unslotted CSMA/CA. Every FFD is always active to receive data from the end devices. The RFD sends a request to the FFD to receive data from the FFD. Additionally the MAC layer partially supports the admission process of new devices. This process starts with the scan procedure. During this procedure the RFDs listen for beacon requests send by the FFD. To complete the admission process, request and acceptance notifications are exchanged at the MAC layer. The acceptance depends on the security mechanisms supported by upper layers. In case of acceptance the new device receives a 16bit address (Christin, Mogre, & Hollick, 2010, p. 107).

The network layer is responsible for network formation, address assignment and routing in the ZigBee network. This layer initiates the network discovery mechanism to detect other ZigBee networks. The application layer select the network, the network layer select a parent to attach the joining device and assigns to the MAC layer to begin the association. The network layer provides a 16bit address and employs the Ad hoc On Demand Distance Vector (AODV) routing algorithm. This algorithm is used in route discovery in mesh networks (Christin, Mogre, & Hollick, 2010, p. 107).

The application layer is composed of 240 Application Objects (APO). The APOs are software units controlling dedicated device hardware and are distributed over the network devices. Each APO consist a set of variables and provides the ability to set, read, or report changes in these values. An APO local number exceeds to the device address and it is used to access these functions. There are also applications profiles which define formats and protocols which provide intra APO communication (Christin, Mogre, & Hollick, 2010, p. 107).

2.2.6 Wireless Sensor Network Applications

There are many WSN applications in different sectors like industrial automations, automotive, home applications, agriculture, military, health, etc. Figure 2.28 indicates and example where sensor nodes could be deployed in a field where corps (e.g. grapes) are developing. These sensor nodes could sense temperature, humidity, ground humidity etc. and form a wireless sensor network in a mesh topology. After-wards the data is aggregated and send to the manager using a satellite link or a GPRS/3G connection. The previous scenario could assist in the agriculture increasing the production effectiveness.

Another scenario is shown in Figure 2.29 where a sensor network is deployed in a forest in order to be able to detect fire in specific areas and prevent major forest de-structions.

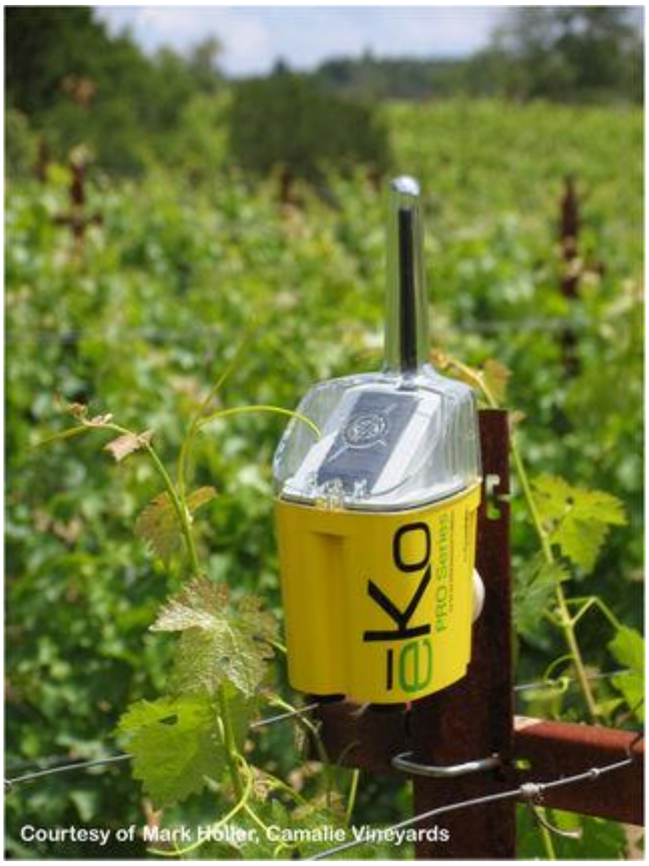


Figure 2.28: WSN in agriculture

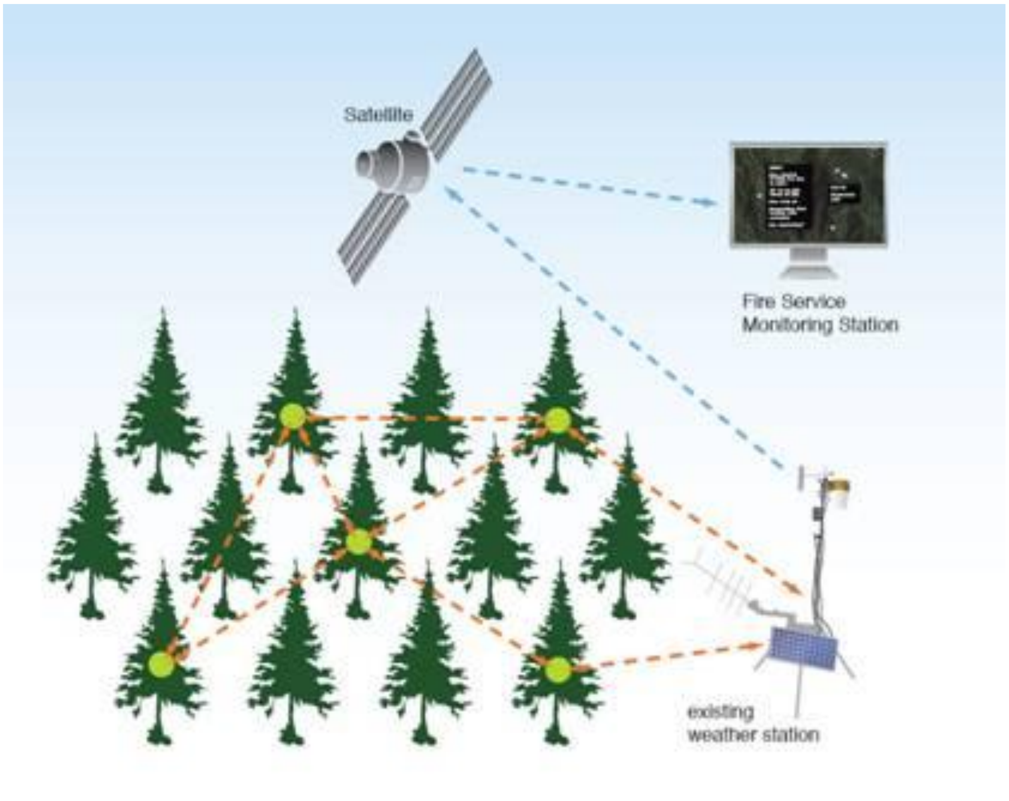


Figure 2.29: WSN in a forest

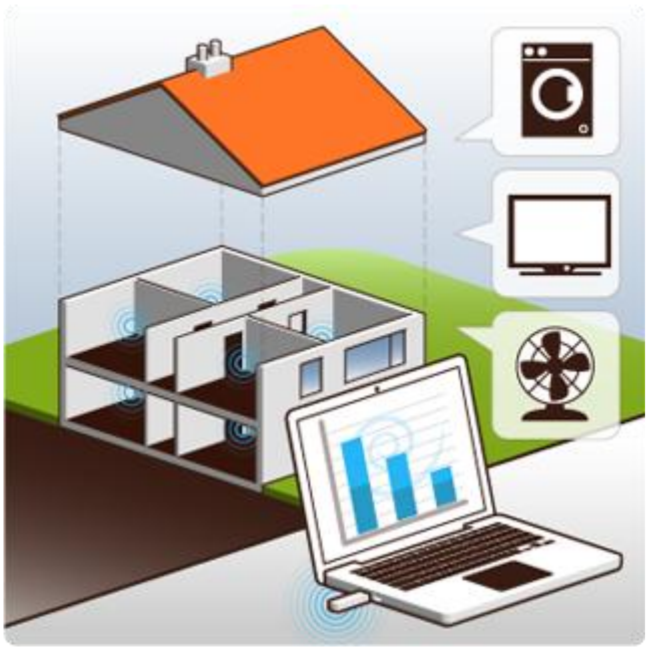


Figure 2.30: WSAN in a smart home application

Wireless Sensor and Actuator Networks (WSAN) can be deployed in smart home applications, shown in Figure 2.30. In home applications WSAN can monitor the energy consumption, temperature, motion etc. Additionally the actuators can provide to the manager/user the remote control (e.g. ON-OFF) off different devices.

2.3 Service Oriented Architecture

Service Oriented Architecture is closely related to AmI systems since it contributes in the reuse of available functions and the remote control between Enterprises. SOA is implemented by web services. In this section Web Services and the related protocols will be presented.

2.3.1 Overview

A service oriented architecture is an information technology which the applications use the services available in the network (e.g. some applications use the World Wide Web service). The implementation of a service-oriented architecture includes developing applications which rely on services and make the available applications services, to be used by other applications (Ort, 2005, p. 3). A service provides a function, in most cases a business function. SOA is an approach that connects applications and provides intercommunication between them. The service oriented architectures have been used for years. SOA differs from other architectures because of the loose relationship between the service and the client. This loose relationship means that the client is independent of the service. The client communicates with the service using a well defined interface and the service performs the processing. In case the service implementation changes, the client will communicate with it in the same way as before. This relationship makes services document-oriented.

> *A document-oriented service accepts a document as input, as opposed to something more granular like a numeric value or Java object. The client does not know or care what business function in the service will process the document. It's up to the service to determine what business function (or functions) to apply based on the content of the document*
>
> (Ort, 2005)

The main reasons for using SOA based approaches are (Ort, 2005, p. 4;5;6):

- *Reusability:* SOA allows the reuse of existing assets in this way new services can be created by existing applications.
- *Interoperability:* In SOA users and services can interact between them even though they may run on different platforms.
- *Scalability:* These services are more scalable since the user and the service are loosely coupled.
- *Flexibility:* SOA applications are flexible and easy to evolve with changing requirements.
- *Cost Efficiency.*

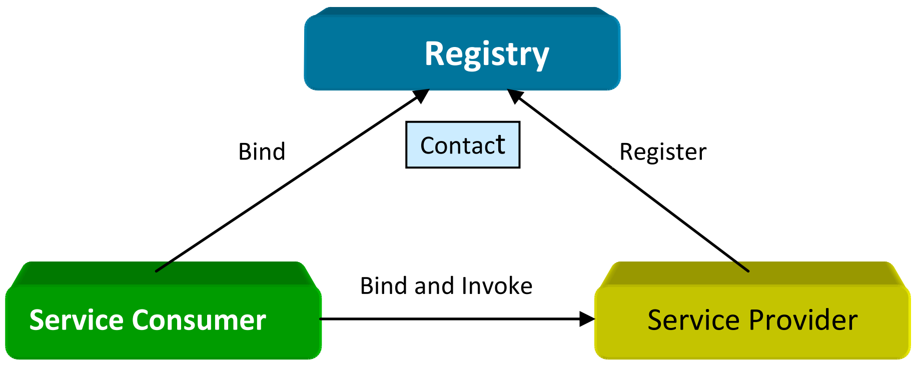


Figure 2.31: SOA find-bind-execute paradigm

SOA uses the find-bind-execute paradigm shown in Figure 2.31. In this paradigm the Service Providers register their services in the registry. The Service Consumer searches the registry to find services that match specific criteria. If the registry has such a service, it provides to the consumer a contact and an endpoint address for that specific service (Mahmoud, 2005)[4]. Afterwards the consumer can bind and invoke the service from the provider.

2.3.2 Web Services

A Web Service is a software system which facilitates the connection between two electronic devices over a network. This software system provides the means for interoperability between different application software.

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards

W3C WORKING GROUP NOTE 11 FEBRUARY 2004

Web services and SOA are two different things, but web services are the standards-based way to realize SOA (Mahmoud, 2005).

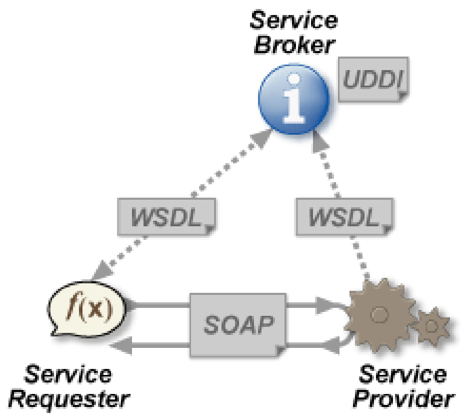


Figure 2.32: Web Services Architecture (by H. Voormann)

In comparison to the example in Figure 2.31, Figure 2.32 provides an example of the architecture of a web service. In this example the service provider publish a service using an XML document, called WSDL. This is a machine-processable document which contains a description of the web service's interface. The service requester typically

[4] http://www.oracle.com/technetwork/articles/javase/soa-142870.html

searches for the WSDL document of a service in a UDDI registry or ebXML registry/repository.

2.3.3 Related Protocols

As mentioned in the previous sections the main protocols used: to discover a service is UDDI; to describe a service WSDL; for XML messaging SOAP, XML and XML-RPC; to transport services the HTTP, FTP, SMTP, BEEP.

Service Discovery Protocol	UDDI
Service Description Protocol	WSDL
XML Messaging Protocol	SOAP, XML, XML-RPC
Service Transport Protocol	HTTP, FTP, SMTP, BEEP

Figure 2.33: Web Services protocol Stack

Figure 2.33 indicates the protocol stack in web services and the functionality and hierarchy of each protocol. These protocols will be described in the following paragraphs.

eXtensible Markup Language (XML)

eXtensible Markup Language (XML) is a markup language which is widely used to describe data being exchanged on the web. XML uses tags to describe the content of a document. The XML tags identify the information in a document and also the structure of the information. An example is shown in Figure 2.34 where the information “Bibliography” is identified by XML. Additionally the structure of `Bibliography` is described. The `Bibliography` contains one subordinate item described as `book`. The `book` has four subordinate items identified as `title`, `author`, `year` and `isbn`.

```
<Bibliography>
      <book>
            <title>Understanding Digital Signal Processing</title>
            <author>Richard G. Lyons</author>
            <year>2010</year>
            <isbn>0137027419</isbn>
      </book>
</Bibliography>
```

Figure 2.34: XML Example

The information inside the tags has a meaning only if people associate a particular meaning with a particular tag. When people agree on a meaning of a tag (e.g. <book> is used to identify a book and <title> <author> <year> and <isbn> are used to identify the title, author, year and ISBN respectively) and use those tags consistently, this provides a way to exchange data. An XML document is typically associated with a schema which specifies which tags are allowed in the document, the structure of those tags, and the rules about the tags (e.g. what type of data is expected in a tag). Valid XML documents are well formed and conform to the associated schema. This makes it relatively easy to process XML documents. Therefore XML has been generally used as a data language in web services (Ort, 2005, p. 6;7).

Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is an XML based protocol used to exchange information in a distributed environment. SOAP provides a common format for messages when exchanging information. It defines the structure of the message and this facilitates the applications to be able to interpret or send the data. As shown in Figure 2.35, a SOAP message is formed by a SOAP Envelope which is mandatory; a SOAP Header which is optional; and a SOAP Body which is also mandatory. The SOAP Envelope is used to specify an XML namespace and an encoding style. The former specifies the names that can be used in the SOAP message and the later identifies the data types recognized by the SOAP message.

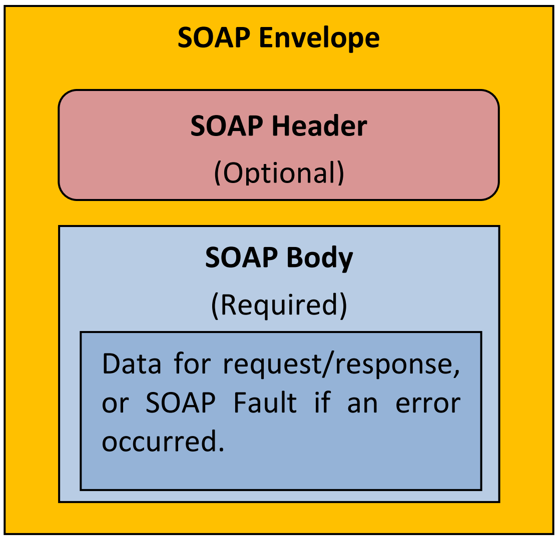
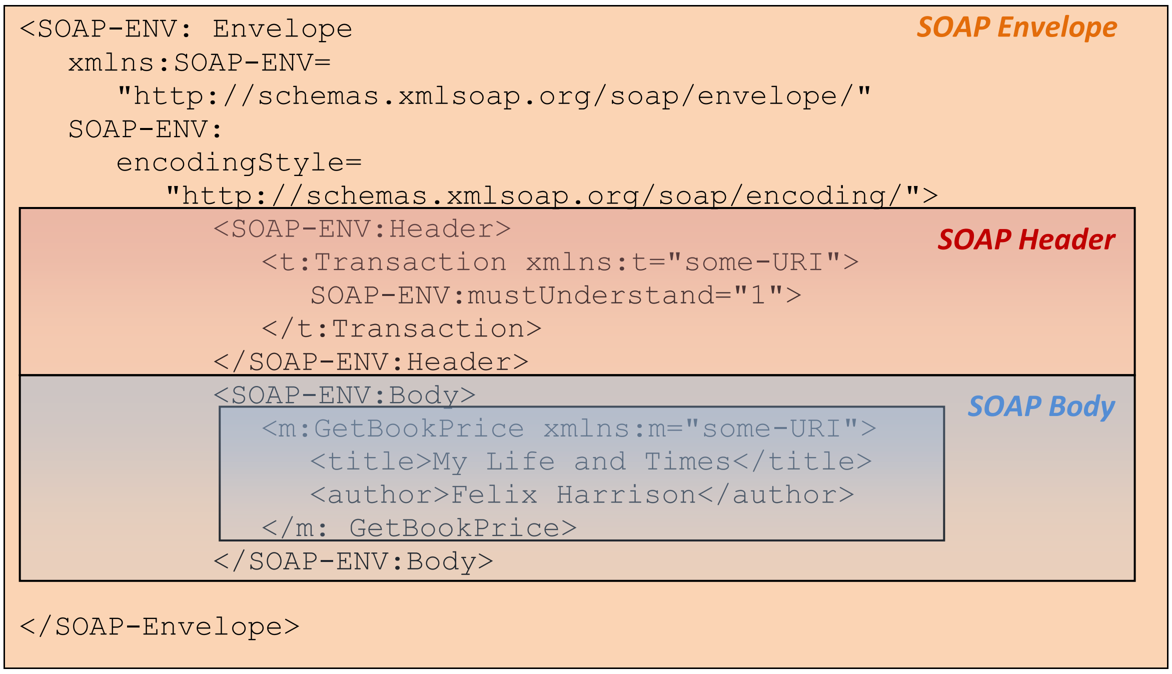

SOAP Envelope
SOAP Header
(Optional)
SOAP Body
(Required)
Data for request/response, or SOAP Fault if an error occurred.

Figure 2.35: Conceptual SOAP message structure

In most of the cases a SOAP message passes through intermediate nodes when traveling from a client to a service. The intermediate nodes are applications which receive and forward SOAP messages. These intermediate nodes can provide additional services (e.g. perform security operations or transform the data in the message.).



```
<SOAP-ENV: Envelope
   xmlns:SOAP-ENV=
      "http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:
      encodingStyle=
         "http://schemas.xmlsoap.org/soap/encoding/">
            <SOAP-ENV:Header>
               <t:Transaction xmlns:t="some-URI">
                  SOAP-ENV:mustUnderstand="1">
               </t:Transaction>
            </SOAP-ENV:Header>
            <SOAP-ENV:Body>
               <m:GetBookPrice xmlns:m="some-URI">
                  <title>My Life and Times</title>
                  <author>Felix Harrison</author>
               </m: GetBookPrice>
            </SOAP-ENV:Body>

</SOAP-Envelope>
```

Figure 2.36: An actual example of a SOAP message

The SOAP Header can be used to indicate additional processing in the intermediate nodes, independent of the processing done at the final destination. Typically the header contains information processed by infrastructure within a Web Server (e.g. a SOAP header can be used to provide routing information for the SOAP message). Last the SOAP Body contains the information (i.e. “payload”) intended for the final recipient of the SOAP message. In case there is a problem the SOAP Body will contain error information in the form of a SOAP Fault. A SOAP Fault is an XML structure which describes the error. An actual example of a SOAP Message is shown in Figure 2.36. This example was provided by Ort in 2005 (Ort, 2005, p. 8) and is a SOAP message used to retrieve the price of a book. The elements <SOAP-ENV: Envelope>, <SOAP-ENV: Header> and <SOAP-ENV: Body> are used to markup the Envelope, Header and Body of the message.

SOAP messages can be transported using communications protocols like HTTP or SMTP. These messages are independent of the operating system or the platform. The main specifications of the World Wide Web Consortium (W3C) are SOAP 1.1 in May 2000 and SOAP 1.2 in April 2007. There are some differences between SOAP 1.1 and 1.2 and it is possible that a server that understands SOAP 1.1 may not be able to accept SOAP 1.2 messages.

Web Service Description Language (WSDL)

WSDL is a specification used to describe web services.

> *WSDL describes four critical pieces of data:*
>
> - *Interface information describing all publicly available functions*
> - *Data type information for all message requests and message responses*
> - *Binding information about the transport protocol to be used*
> - *Address information for locating the specified service*
>
> (Cerami, 2002, p. 103)

Conceptually, WSDL files represent a contract between the service requestor and the service provider. WSDL is independent of the platform and programming language. By using WSDL a client can locate a web service and invoke its available functions. WSDL is a common language for describing web services and can provide a platform for the automatic integration of web services (Cerami, 2002, p. 103).

The WSDL is an XML-based language which consists of six major elements, shown in Figure 2.37 (Cerami, 2002, p. 104): definitions, types, message, portType, binding and service.

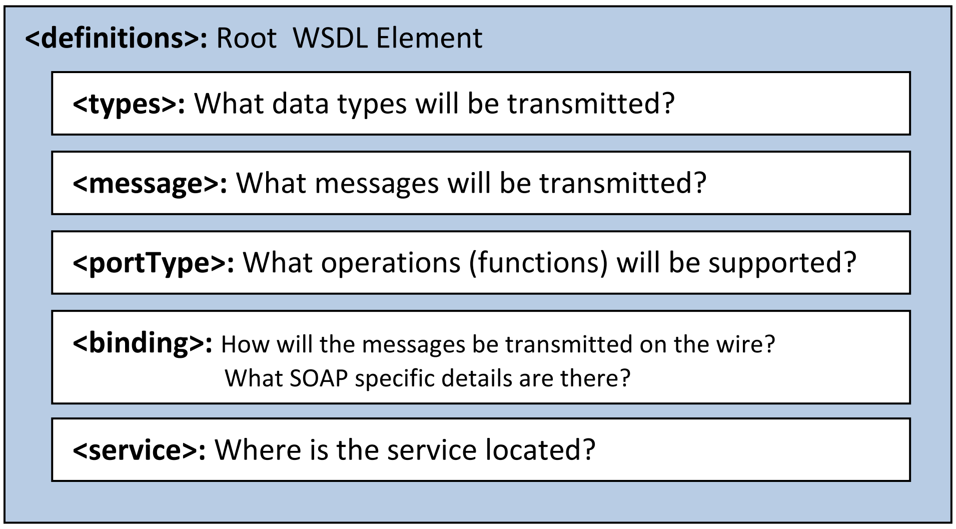

<definitions>: Root WSDL Element
<types>: What data types will be transmitted?
<message>: What messages will be transmitted?
<portType>: What operations (functions) will be supported?
<binding>: How will the messages be transmitted on the wire? What SOAP specific details are there?
<service>: Where is the service located?

Figure 2.37: WSDL major elements

The <definitions> is the root element and contains the name of the web service, declares multiple namespaces, and contains all the service elements described. The <types> element describes the data types used between the client and the server. WSDL does not use only one specific typing system but by default the W3C XML Schema is used. The <message> element describes a single message, a request or a response. It contains the message's name and zero or more <part> elements which refer on the message's parameters or return values. The <portType> element combines more than one message to form a complete operation (e.g. combines a request and a response in a single request/response function. The <binding> element contains SOAP specific information. Last the <service> element contains the address for invoking the web service, usually a URL. Additionally there are also two utility elements the documentation and the import element. The former is use to provide human readable documentation and the later to import other WSDL documents or XML Schemas (Cerami, 2002, p. 104;105).

Even though, WSDL is a fundamental requirement for the implementation of web services, there are some disadvantages. WSDL does not provide some information such as:

- The provider of the service.
- The type of business that provides the service.
- Other Services available from the same provider.
- The quality of service that the provider offers.
- The cost for using the service (e.g. free or fee based)

The standard that provides this kind of information is UDDI (Endrei, et al., 2004, p. 125).Upon now the W2C has released three specifications of WSDL, the WSDL 1.1 in March 2001, the WSDL 1.2 in June 2003, and WSDL 2.0 which was published as a W3C recommendation in June 2007. The WSDL 1.1 and WSDL 2.0 have some significant differences between them.

WSDL 2.0 has three specifications:

- *Core, which explains the abstract interfaces independent of protocol and encoding;*
- *Message Exchange Patterns (MEP) with predefined types of interactions; and*
- *Bindings pertaining to SOAP and HTTP.*

(Padmanabhuni, Chaudhari, Bharti, & Kumar, 2007)

Universal Description Discovery and Integration (UDDI)

UDDI is an XML based mechanism used to list and locate web services. It is considered as a directory for storing information about web services. A UDDI registry manages information about the service provider, service implementation, and service metadata. Service providers can use UDDI to advertise the services they offer and service consumers use UDDI to find the services that fulfill their requirements and retrieve the services metadata in order to consume the service (OASIS). The interfaces of these web services are described by WSDL. UDDI communicates using SOAP and uses protocols like HTTP, DNS and XML. The UDDI specifications define a UDDI Schema and a UDDI API. The UDDI schema identifies the types of XML data structures of a service in the registry. The API describes the SOAP messages used to publish or discover an entry in the registry. A UDDI registry provides information such as the name of the service, a brief description about what the service does, an address where the service can be accessed, and a description of the interface to access the service (Ort, 2005, p. 10).

A business can register three type of information in a UDDI registry (Chappell & Jewell, 2002, p. 96;97):

- Contact information and identifiers about the company (e.g. business name, address, unique identifiers like tax IDs). This can facilitate the service consumers finding the registered web service based on the business identification.
- Information that describes the web service based on different taxonomies. Allowing service consumers to find the web services based on categorizations (e.g. manufacturing business).
- Technical information which describes the supported functions and the behavior of the web service. This information provides also the location of the web service.

2.3.4 Web Services in Java

Web services are designed to be language and platform neutral. To develop a web service or an application that uses web services, platforms and programming languages are needed. Java is a language that can be used in developing such services and applications. In present, the commonly used Java platforms in the development of web services are: the Java 2 Enterprise Edition 1.4 (J2EE), the Java Enterprise Edition 5 (Java EE 5), and the Java Enterprise Edition 6 (Java EE 6). These platforms consist of Java technologies which are designed for use with XML, and conform to web services standards like SOAP, WSDL and UDDI.

The J2EE 1.4 was developed under the Java Specifications Request (JSR) 151 and the final release was in November 2003. The Java EE 5, which is an evolution of J2EE 1.4, was developed under the JSR 244 with the final release in May 2006. Last the Java EE 6 under the JSR 316 with the final release in December 2009. The specifications of each platform are provided in the Java Community Process web site[5].

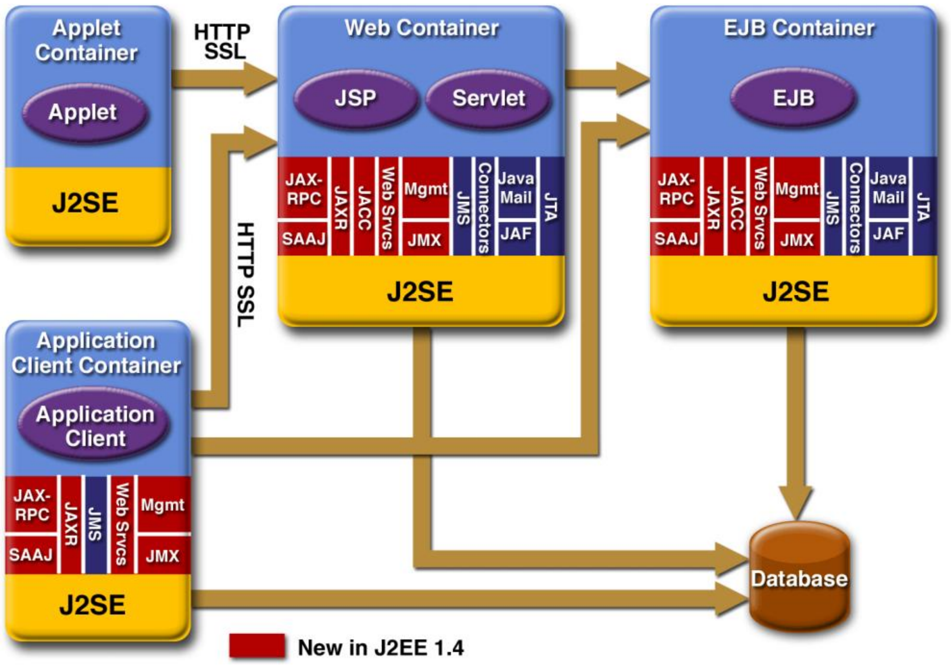


Figure 2.38: Architecture and APIs of the J2EE 1.4

Figure 2.38, provided in the J2EE 1.4 Tutorial in 2007 by Armstrong et al p.60 Figure 1-7, indicates the architecture of the J2EE 1.4 platform. The new added APIs and specifications, in comparison to the version 1.3, are illustrated in red.

[5] http://jcp.org/en/home/index

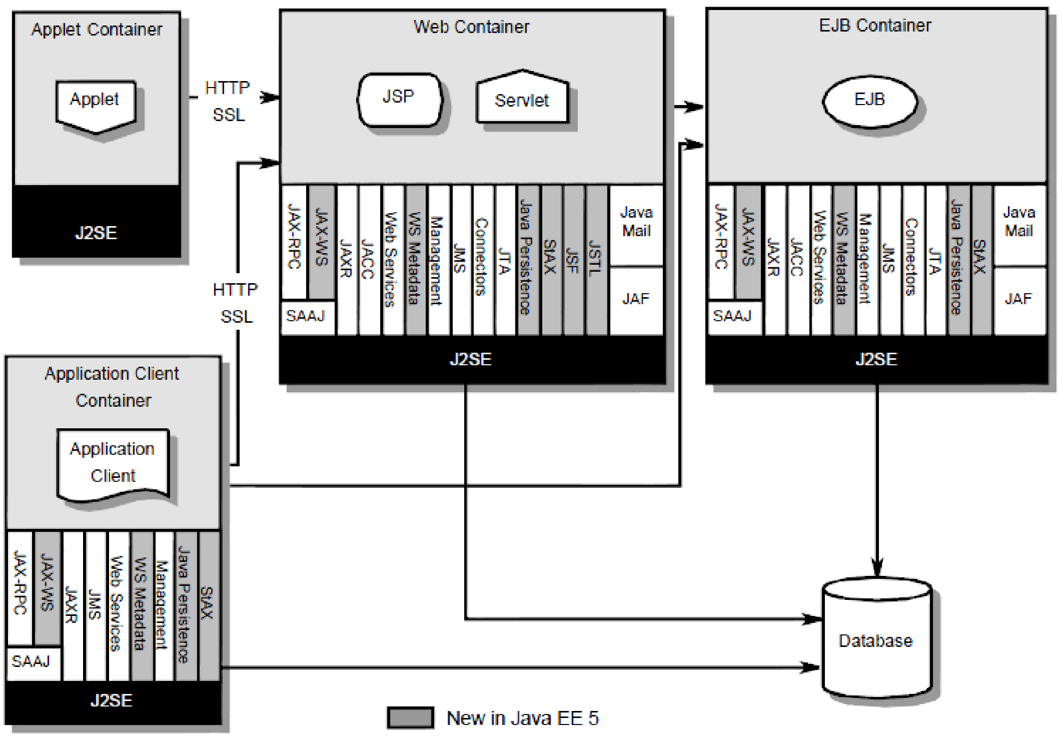


Figure 2.39: Java EE 5 architecture diagram

Figure 2.39, provided by Sun Microsystems in the Java EE 5 specification in 2006 p.6 Figure EE.2-1, demonstrate the architecture of the Java platform EE 5. Again the new added APIs and specifications are marked in dark grey in comparison to the others.

Respectively, Figure 2.40, provided by Sun Microsystems in the Java EE 6 specification in 2009 p.22 Figure EE.2-1, illustrates the architecture of Java EE 6 and the APIs. As in Figure 2.39, the new specifications are also marked in dark grey.

More detailed specifications about these platforms are provided by the Java Community Process web site (jcp.org) in the JSR 151, 244 and 316. In each platform specification, detailed information and descriptions about each API or technology can be found.

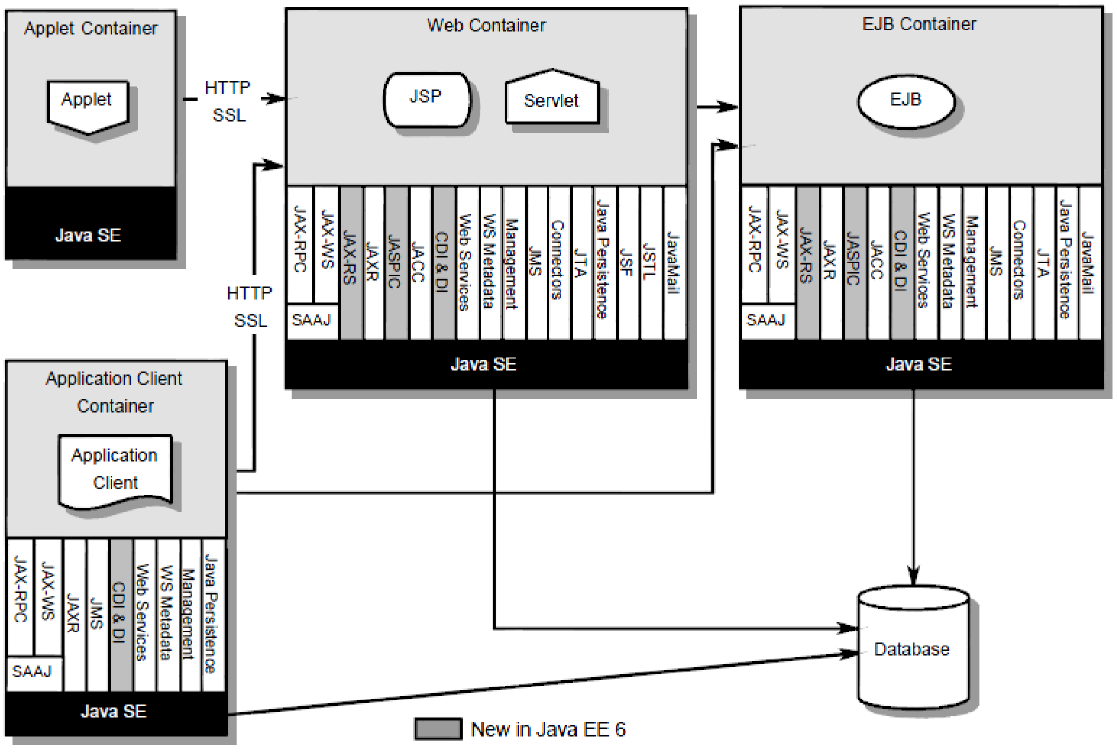


Figure 2.40: Java EE 6 architecture diagram

3 Problem Definition

Initially, in this chapter, the Smart IHU project will be investigated and the existing devices and applications will be presented and described. Even though the existing project includes many pioneer ideas and innovative functionalities, there are still some parts that need to be improved and problems to be solved. The next step is to detect, these problems and requirements, and afterwards to address them in order to be able to provide possible improvements and solutions. Last, there will be a presentation about available hardware and software tools, that may solve these problems or improve the existing system.

3.1 The Smart IHU Project

The Smart International Hellenic University (i.e. Smart IHU) is a research project in the field of Information and Communications Technologies for sustainable growth and energy efficiency. This project is based on the cooperation of the ICT and Energy departments of the School of Science and Technologies of the International Hellenic University (IHU). The main objective of this research project is to transform IHU into a "Smart" University with automated processes, enabling Smart Building Smart Grid technologies, remote monitoring and management, energy efficiency, and providing support for educational activities (IHU School of Science and Technology, 2010).

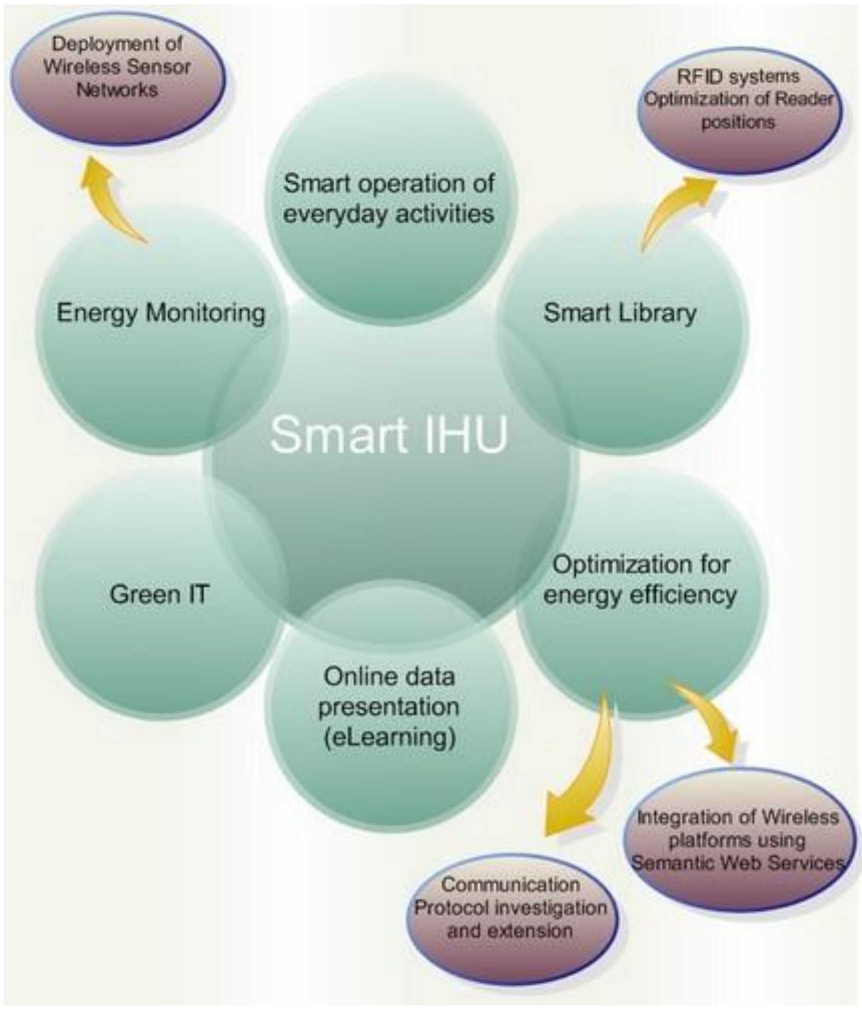

Deployment of Wireless Sensor Networks
RFID systems Optimization of Reader positions
Smart operation of everyday activities
Energy Monitoring
Smart Library
Smart IHU
Green IT
Optimization for energy efficiency
Online data presentation (eLearning)
Integration of Wireless platforms using Semantic Web Services
Communication Protocol investigation and extension
Figure 3.1: Smart IHU research directions

Figure 3.1, retrieved from the *Research and Development web page of School of Science and Technology regarding Smart IHU*[6], indicates the research directions of this project. The main research directions include:

- The design and deployment of WSNs in order to monitor energy consumption of the building.
- Integrating the wireless platforms using Semantic Web Services (WS).
- Designing energy aware algorithms to reduce energy consumption in WSNs.
- Developing algorithms to control, schedule and optimize power tasks of smart appliances.
- Optimizing and evaluating the placement and range of RFID tags and readers in the IHU library.

[6] http://rad.ihu.edu.gr/index.php?id=si

3.1.1 System Architecture

The Smart IHU is a complex system which incorporates various WSN technologies and integration platforms.

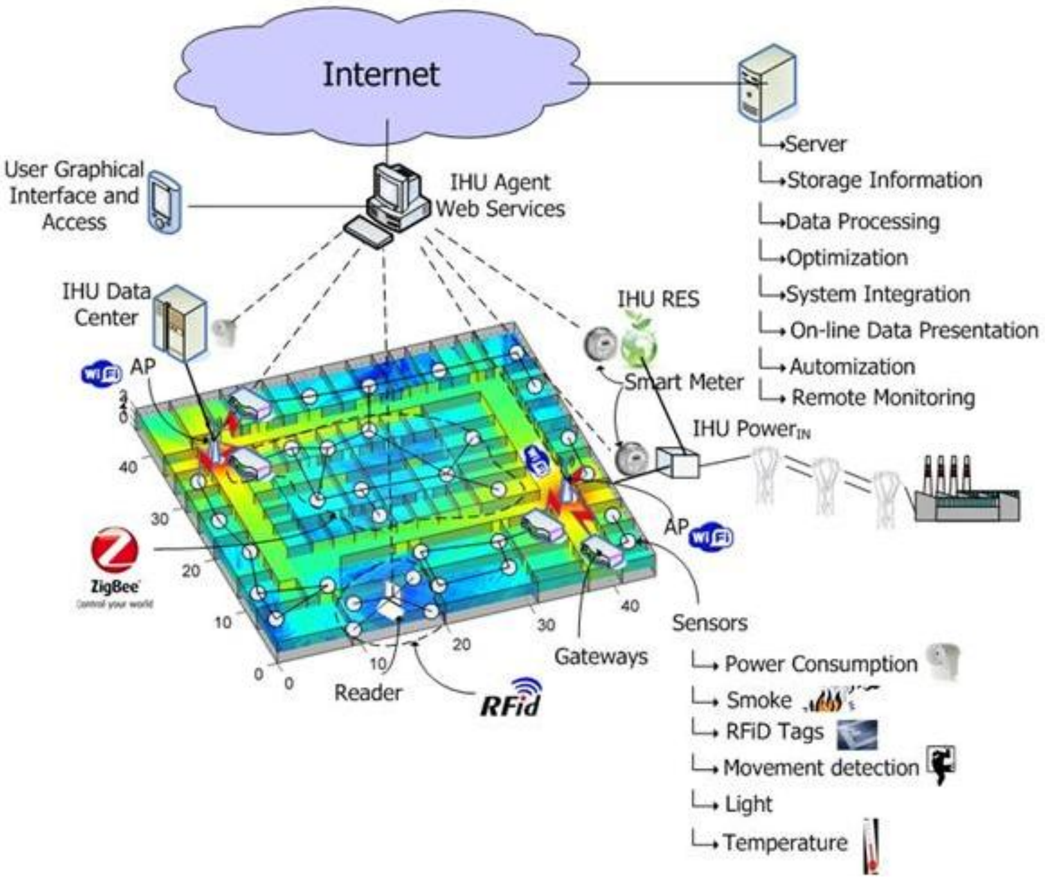


Figure 3.2: Smart IHU System

Figure 3.2, provided in the Smart IHU presentation web page (IHU School of Science and Technology, 2010), indicate the Smart IHU system. The main elements involved in this system are: WiFi Network; ZigBee Sensors and Network; RFID tags and readers; PC agent; Semantic Web Services Middleware for system integration; Smart Meters; Server; User Interface and PDA data presentation; Data Center for Green IT computations; Gateways; Renewable Energy Sources (RES), mainly solar (IHU School of Science and Technology, 2010).

The operational layers of the system are shown in Figure 3.3. Additionally this Figure indicates the technologies, the devices, and the operations, found in each layer.

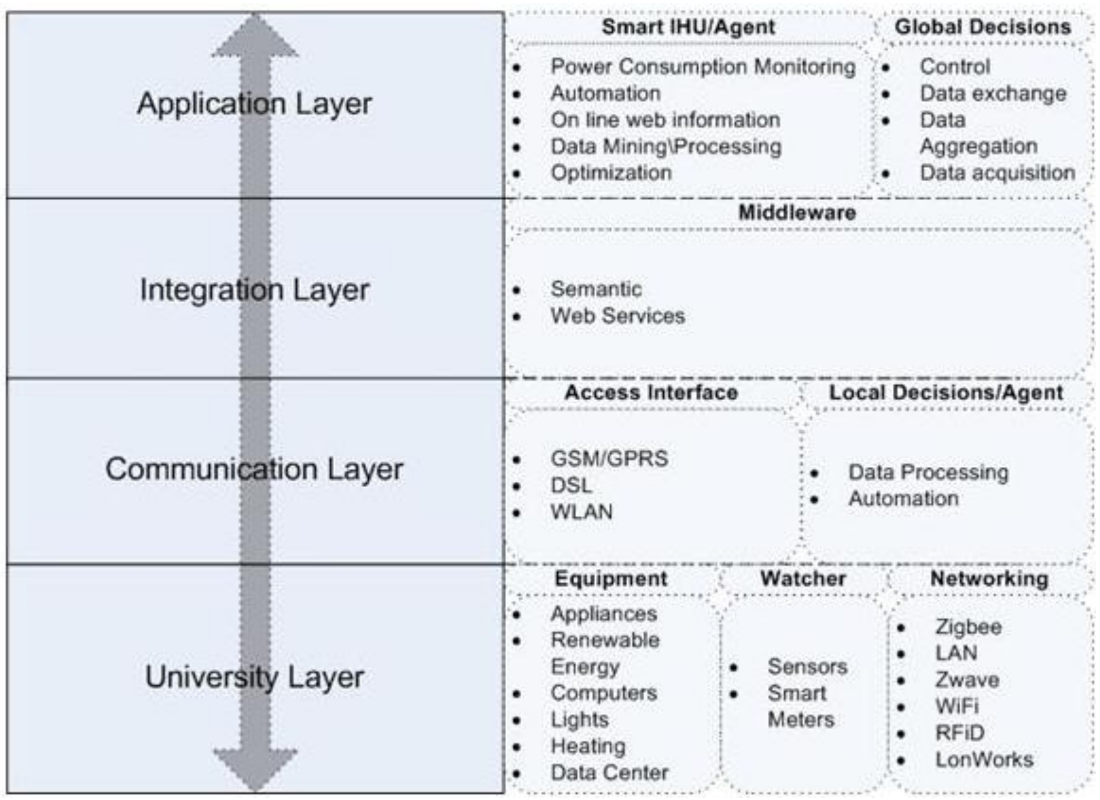


Figure 3.3: Smart IHU operational layers, technologies and devices.

Figure 3.3 is found in the Smart IHU presentation web page (IHU School of Science and Technology, 2010).

3.1.2 Devices

The scope of this sub section is to describe the devices used in the Smart IHU project to monitor the energy consumption and environmental parameters. There are different devices of different manufacturers. Specifically the manufacturers are: Plug-wise, Prisma Electronics SA, OWL and CurrentCost.

Plugwise

Plugwise is a Dutch company which has developed and produced wireless systems for energy management and appliance control since 2006. The company provides a wide range of wireless devices but the main devices used in the Smart IHU project are: the smart plugs, Circle and Circle+; and the Stick. The software that is used to receive power measurement data and manage these data is called Source.

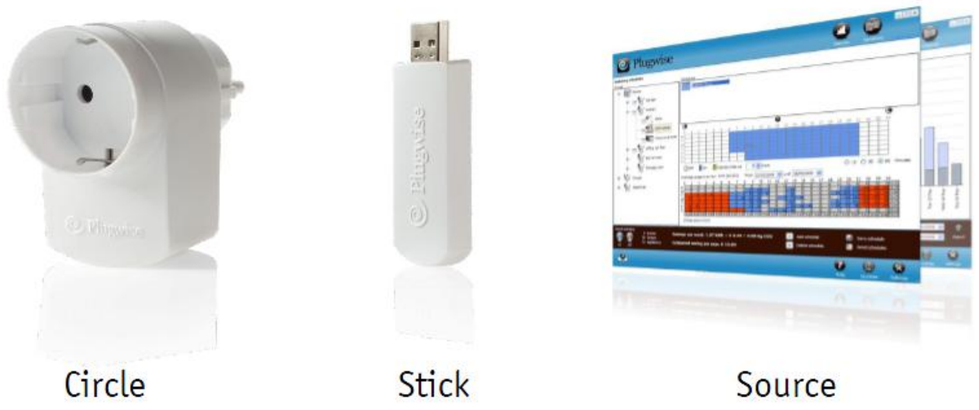


Figure 3.4: Plugwise – power management components

The components of the power monitoring and management system developed and produced by Plugwise are shown in Figure 3.4, provided by Plugwise[7].

Circle and Circle+

The smart plugs, called Circle and Circle+, are the sensor nodes which measure energy and switch appliances wirelessly. The relay, shown in Figure 3.5 provided by Plugwise, facilitates the power control of the appliance connected in the Circle (i.e. the appliance can be switched on or off using a computer). Circle is equipped with a stand-by killer which powers off the device when it enters in standby mode. The user can restart the device by unplugging and plugging back Circle.

Circle is equipped with a power supply unit which converts the Alternative Current (AC), provided by the power grid, to Direct Current. DC is used to power the various electronic circuits of Circle. Therefore the sensor nodes in a Plugwise WSN are powered directly from the power grid. Circle+ is equipped with a real time clock and a battery, shown in Figure 3.5, and periodically synchronizes time with other Circles. When the network is formed the Circle+ can act as a regular Circle. Additionally a flash memory is implemented with a storage capacity of 512Kbyte.

[7] http://www.plugwise.com

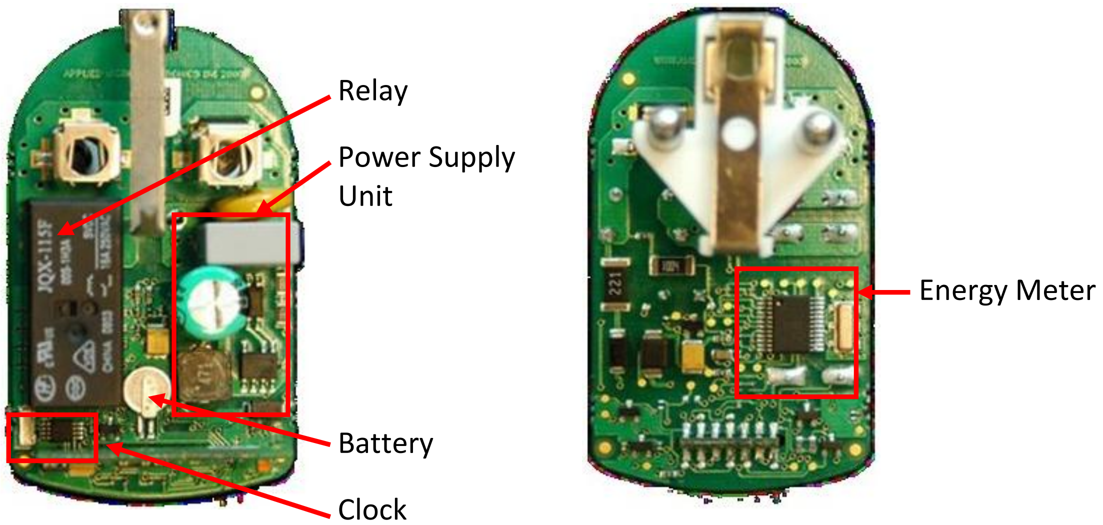


Figure 3.5: Circle+ Electronic Components and Circuits

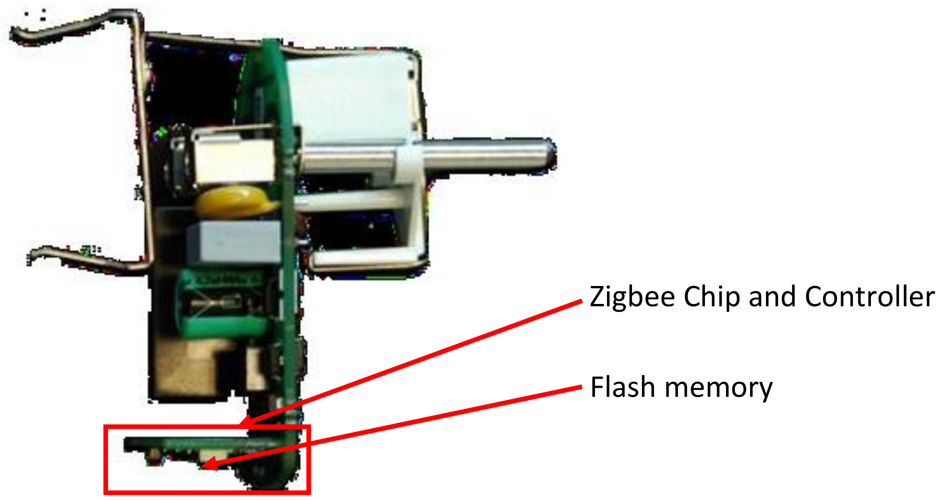


Figure 3.6: Circle+ Zigbee circuit and Flash memory

In regards the wireless communication of these devices, both Circle and Circle+ have an integrated Zigbee Chip and Controller, shown in Figure 3.6. The standard used is Zigbee Pro with a manufacturing specific profile but accessible with key to other non Plugwise systems. Plugwise's profile has a highly secure authentication using a 128 bit AES encryption. Each device has its own unique MAC address and can be individually identified. The transmission frequency varies from 2.4 to 2.4835GHz in 16 channels with a nominal receiving sensitivity of -97dbm and a nominal transmission power of 3dbm. As shown in Figure 3.7 provided by Plugwise, the smart plugs can support a full dynamic MESH network where each module can act as a router. This network can easily be extended adding more smart plugs or other devices like Stealth[8] etc.

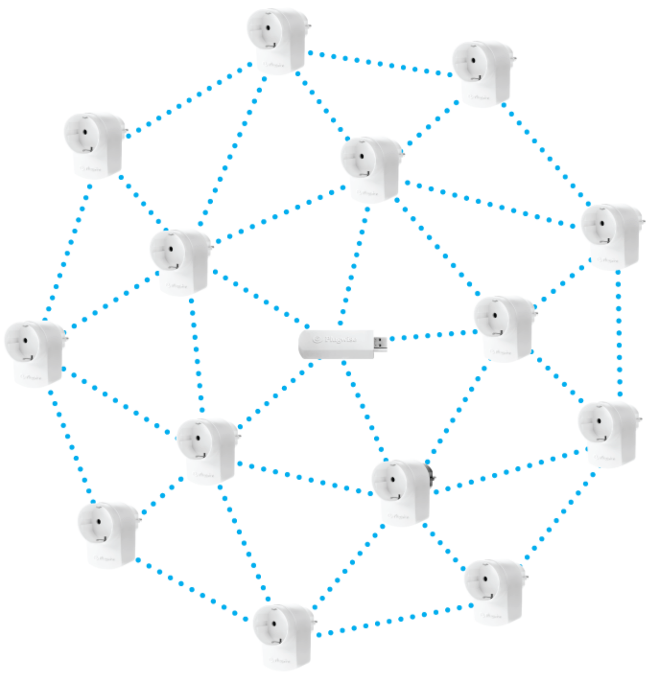


Figure 3.7: Plugwise Mesh network

Both these devices operate in a supply voltage from 83V to 253Volts AC with a 50/60Hz frequency. The power dissipation of each device varies from 0.3 to 1.1Watts (nominal 0.55Watt) under the following conditions: Input Voltage 230V AC; frequency 50Hz; ambient temperature 25^{o}C. The maximum output current is 16Ampere when cosphi=1. This means that the maximum output power of Circle and Circle+, when cosphi=1 and Vin=Vout=230V AC, is P=Voutmax * Ioutmax * cosphi = 230V*16A = *3680Watts*. Therefore these devices cannot be used in high power devices like ovens and boilers. The power measurement accuracy of Circle and Circle+ is 5% (when 230V, 50Hz, 23^{o}C, output range: 1035-3680Watts) and 1% in a one hour cumulative data. In regards the environmental conditions these devices can operate in a maximum temperature of 60^{o}C with a 95% RH maximum humidity.

Stick

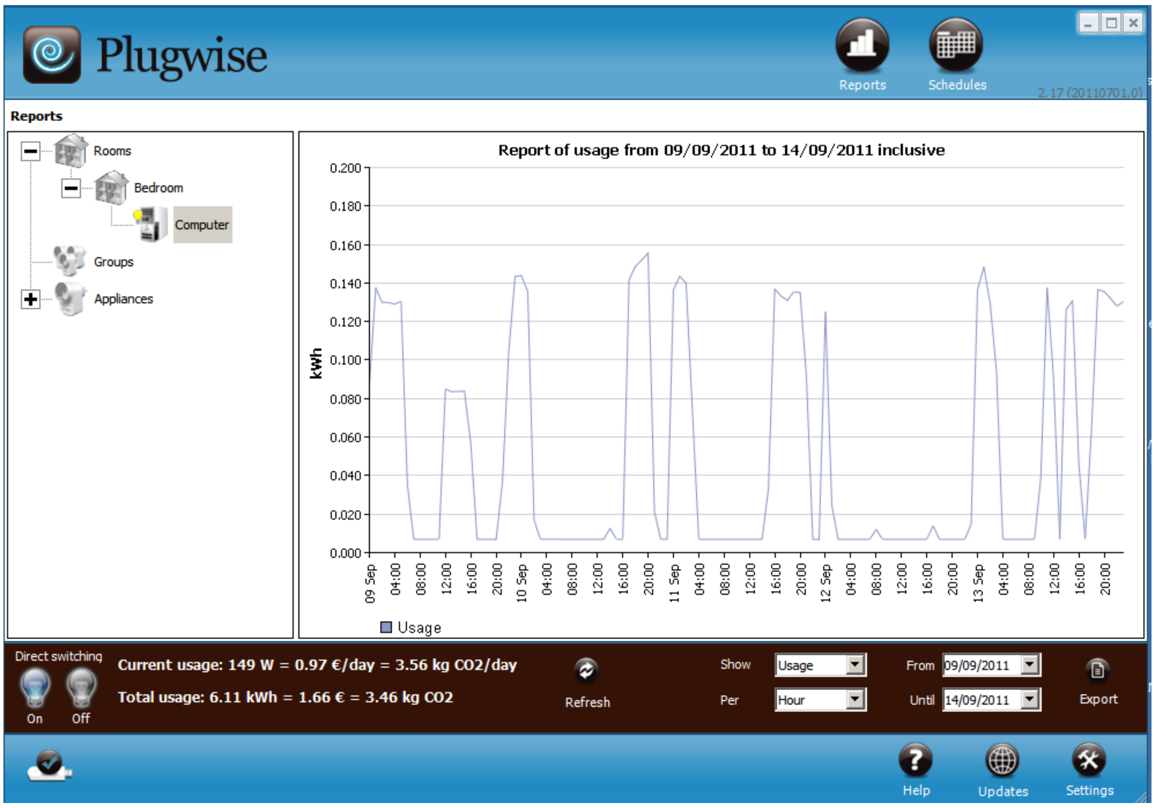
The stick is a USB (A) stick which uses the same wireless standard with Circle. Stick is a link between the Source (Plugwise Software) and the smart plugs (Circle). It is po-

[8] http://www.plugwise.com/idplugtype-f/stealth

wered by the USB host (5V dc) and the nominal power consumption is 0.375Watt. Stick is also has a unique MAC address.

Source

Source is the software used in the Plugwise system. This is software which facilitates the management and aggregation of the energy consumption data. Additionally this software is capable to send On Off commands to the smart plugs and also able to schedule this switching procedure. Each smart plug is initially paired, using its unique MAC address, and afterwards named according to the device attached (e.g. Computer, TV, etc.). Additionally, as shown in Figure 3.8, the devices can be accessed according to their group, appliance, or the room placed.



Report of usage from 09/09/2011 to 14/09/2011 inclusive

Current usage: 149 W = 0.97 €/day = 3.56 kg CO2/day

Total usage: 6.11 kWh = 1.66 € = 3.46 kg CO2

Show: Usage
Per: Hour
From: 09/09/2011
Until: 14/09/2011

Figure 3.8: Source user interface

The user interface indicates the current output power and provides a graphical representation (graph or bar chart) of the energy consumed in KWh. The results can be filtered in certain time periods. By setting the cost for KWh Source can calculate also the overall cost and additionally, according to the KWh consumed, Source can calculate the corresponding CO2 emissions. All these data are stored in a database and can be exported as CSV files for further processing.

PrismaSense

PrismaSense is a developing platform of wireless intelligent sensors delivered by a Greek electronic manufacturer named Prisma Electronics SA.

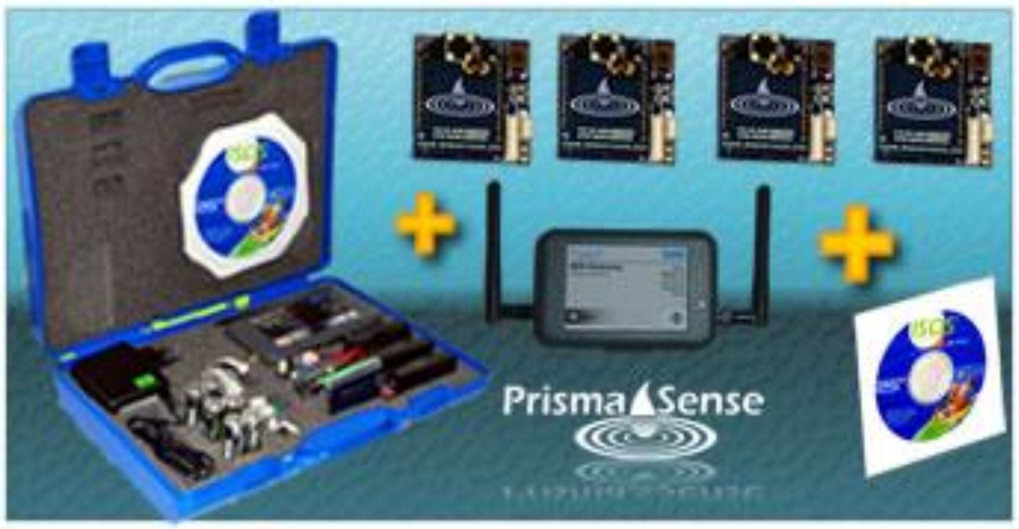


Figure 3.9: PrismaSense components

The main components of PrismaSense are shown in Figure 3.9 provided by Prisma[9]. The hardware devices are the Quaxes and the Gateway. In the PrismaSense platform, a server side software (open API) is included. This software is based on the windows Communication Foundation Services Technology and is easy and flexible for application development.

Quax MS

Quax MS, shown in Figure 3.10 provided by Prisma, is an intelligent multi-sensor device and various sensors (e.g. thermometer, accelerometer etc.) and circuits can be connected to it. Quax is based on a RISC microprocessor, with a very low power consumption, digital inputs/outputs (which can be used for digital sensors), three serial ports, and an Analog to Digital or D/A Converter (which can be used to handle analog signals). This device has a wireless Zigbee transmitter implemented and mesh networks can be created between these modules.

[9] http://www.prismaelectronics.eu

Figure 3.10: Quax MS PRO

The main characteristics of Quax MS PRO are:

- Great power autonomy provided by two AA batteries,
- Up to 10mW Zigbee transmitter,
- PrismaWave (dynamic choice of the communications' channel)
- Power awareness
- Real time clock synchronization
- Adjustment of the transmission power for range greater than 1Km

As an intelligent sensor node Quax MS Pro has an Operating System named ISOS (Intelligent Sensors' Operating System) developed by Prisma Electronics SA. Some features of ISOS are:

- Task Scheduler
- Event Handler
- Frequency Agility
- Packaging Buffer
- Real Time Clock and Synchronization
- Measurement Simulation
- Safe Power Down Modes

The power supply module is software managed and can provide to the user the ability for developing his own power management strategy.

These Quaxes offer a high transmission range, indoor up to 300meters and outdoor up to 1.6Km. With an operating frequency in the 2.4GHz band using 12 Direct Sequence Channels, the interface data rate can reach up to 115Kbps. The receiver sensitivity is -100dBm and a 128bit AES encryption is used to provide secure communication.

Gateway

Prisma SA has developed different types of gateways. The main feature of these gateways is the existence of two connection interfaces. The first is a Zigbee based wireless interface which is used to connect the Gateway with the rest of the Zigbee network created by the Quax nodes. The second can be a WiFi, Ethernet, GPRS, RS232, RS485, or Bluetooth interface. A Zigbee to WiFi gateway is shown in Figure 3.11 (Source: Prisma SA).

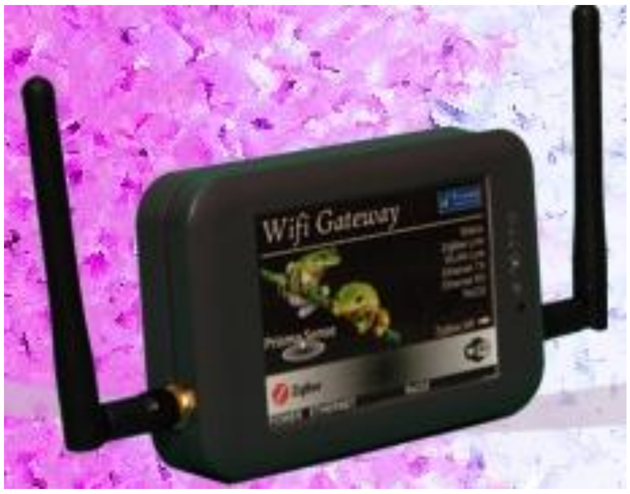


Figure 3.11: Zigbee to WiFi gateway

The gateway is the sink of the WSN and the data is aggregated through the first interface. The second interface facilitates the transport of the aggregated data to the various devices (e.g. PC, PDA etc.) where further processing will be performed.

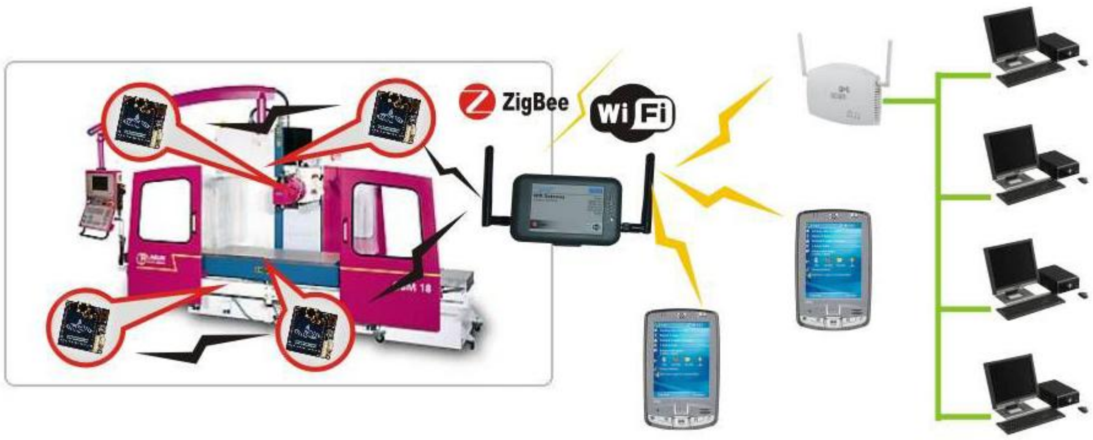


Figure 3.12: PrismaSense System Example

An example of the PrismaSense System is shown in Figure 3.12, where the Quaxes are the sensor nodes deployed in the physical environment creating a Zigbee network. Afterwards the data will be aggregated to the gateway which is connected to various devices (PCs, PDAs) providing this data to different users. Additionally the users can send data to the Quaxes. Since there are digital outputs and Digital to Analog converters in the nodes, analog or digital signals can be generated in order to send data or to control different actuators by using proper electronic circuits.

In a Zigbee to WiFi gateway, 802.11 b/g standards are implemented, in the WiFi interface, providing the range and the data rate that are described in those standards. The Zigbee interface has similar characteristics with the Quaxes, regarding the range, operating frequency, authentication etc. This Gateway has an internal web server and a storage capacity of 1.2 MB. A more detailed description is provided in the manual provided by Prisma Electronics SA.

OWL

OWL is a company which provides power monitoring systems. The Plugwise system can provide data about separate devices in an environment. Additionally there is a maximum power limitation of the Circle device (maximum load 3680 Watts or 16 A). Therefore these devices are not suitable for high load devices like electrical ovens, high power air conditioners (in some cases even low power air conditioners can instantly reach a peak of 16 A or more at the moment they start). The OWL power monitoring platform provides data about the total power consumption of a building facilitating large scale power monitoring.

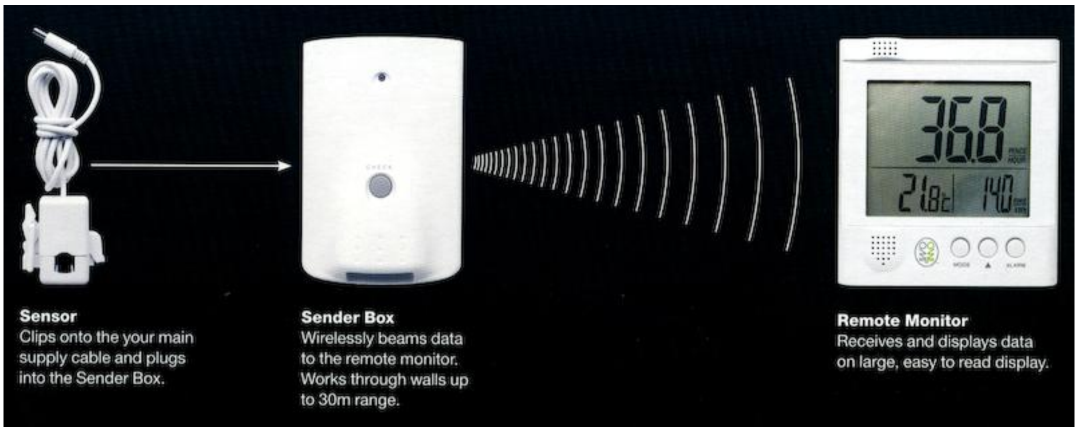

Sensor
Clips onto the your main supply cable and plugs into the Sender Box.
Sender Box
Wirelessly beams data to the remote monitor. Works through walls up to 30m range.
Remote Monitor
Receives and displays data on large, easy to read display.

Figure 3.13: OWL power monitoring system

Figure 3.13, provided by the OWL[10] web site, indicates the basic components of the OWL power monitoring system. The sensor is a current clamp which measures the power. It is connected to the Sender Box which wirelessly, in the 433MHz band, sends the data to the Remote monitor display in a 30 meter range. Additionally the data can be forwarded wirelessly into a computer using a USB device, shown in Figure 3.14 pro-

[10] http://www.theowl.com/

vided by OWL's web site. A software platform is provided in order to manage the power measurement data, energy consumption, cost, estimated CO2 emission, etc.

Figure 3.14: OWL USB Connect

The installation of such a system is simple. A typical installation is shown in Figure 3.15, provided by the Smart IHU team.

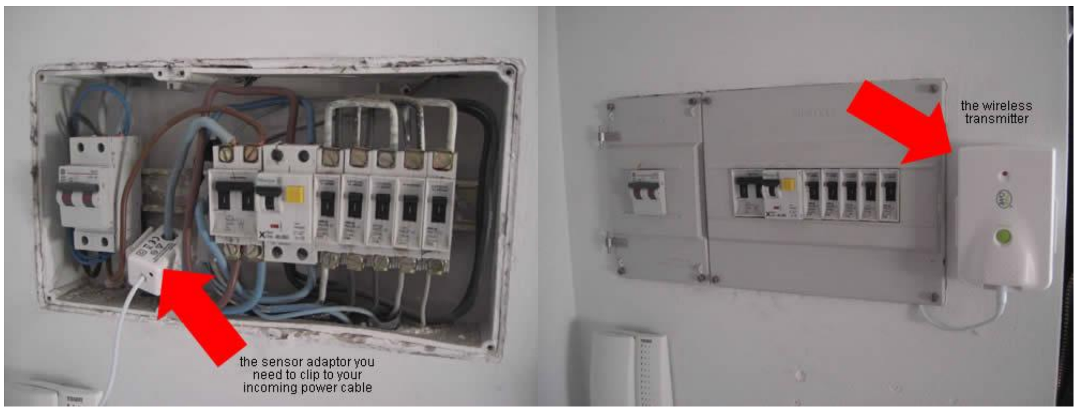

the sensor adaptor you need to clip to your incoming power cable
the wireless transmitter

Figure 3.15: OWL installed

The transmitter and the power monitoring display are battery powered. Both these devices have low energy consumption and the battery lifetime can reach up to 2 years. An important disadvantage, when using the USB receiver, is that the data transmitted by the sender will be lost, at the time that the computer is powered off.

CurrentCost

CurrentCost is another company which provides power monitoring systems similar to the OWL systems. These system is also used for large scale energy monitoring.

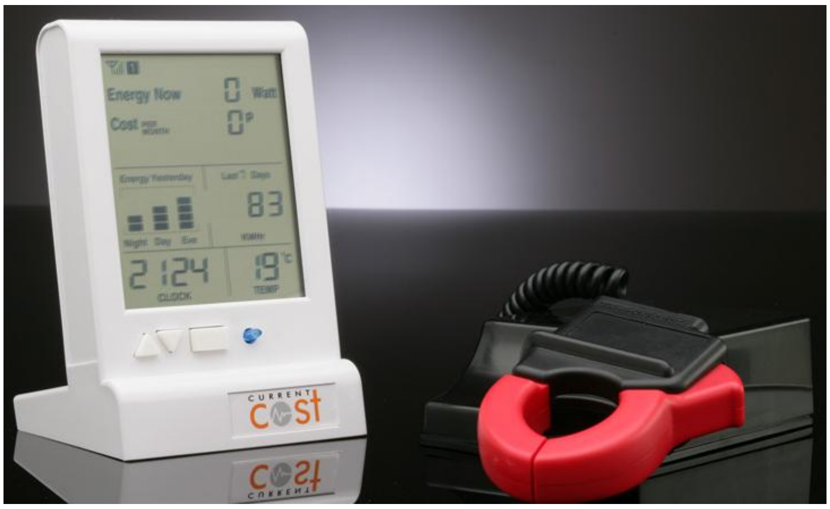


Figure 3.16: CurrentCost System

Figure 3.16 indicates the basic components of the CurrentCost system, on the left side the power monitoring receiver and display, and on the right the sensor (current clamp) and transmitter.

The main difference with the OWL system is that this system is offered at a lower price in comparison to OWL. The way the data is imported to the middleware used in the Smart IHU project, is similar to the OWL system.

3.1.3 aWESoME Middleware

As presented in the previous section the in the Smart IHU project there are different devices which are managed by different application platforms. These distributed systems need to be managed by using the Service Oriented Architecture implementing Web Services to facilitate the AmI implementation. Therefore the existence of a middleware to link these heterogeneous devices, which are mostly distributed, in a single software application and providing operations as web services, is crucial.

In the Smart IHU project the middleware developed in 2010 by the Smart IHU Team, is named aWESoME (a WEb Service MiddlewarE). This middleware was created using the NetBeans IDE 6.9.1, based on the Java EE 5 platform and contain JAX-WS based web services. As shown in Figure 3.17, provided by the Smart IHU team, aWESoME integrates the application platforms of three different systems (Plugwise, PrismaSense and OWL). The aWESoME application is deployed in the Agent which is a web server and web services are provided for each system. Therefore aWESoME provides three different web services: Plugwise Over the Web (POW) which facilitates acquiring data about the power consumption and switching On or Off devices using the Plugwise system; Prisma Web, where the PrismaSense system is provided as a Web Service retrieving data from the Quaxes; and Web OWL which also provides the functions and operations of the OWL system as web services. To consume these web services, JAX-WS web service clients are required. In present, such software applications, regarding the Smart IHU project, are under development. Additionally user friendly GUIs can be used to consume these web services. An example is iDEALISM, a GUI developed by the smart IHU team. In order to develop such client applications the aWESoME application must be deployed in an agent (web server) and afterwards the location (i.e. the path) of the WSDL of each web service must be specified.

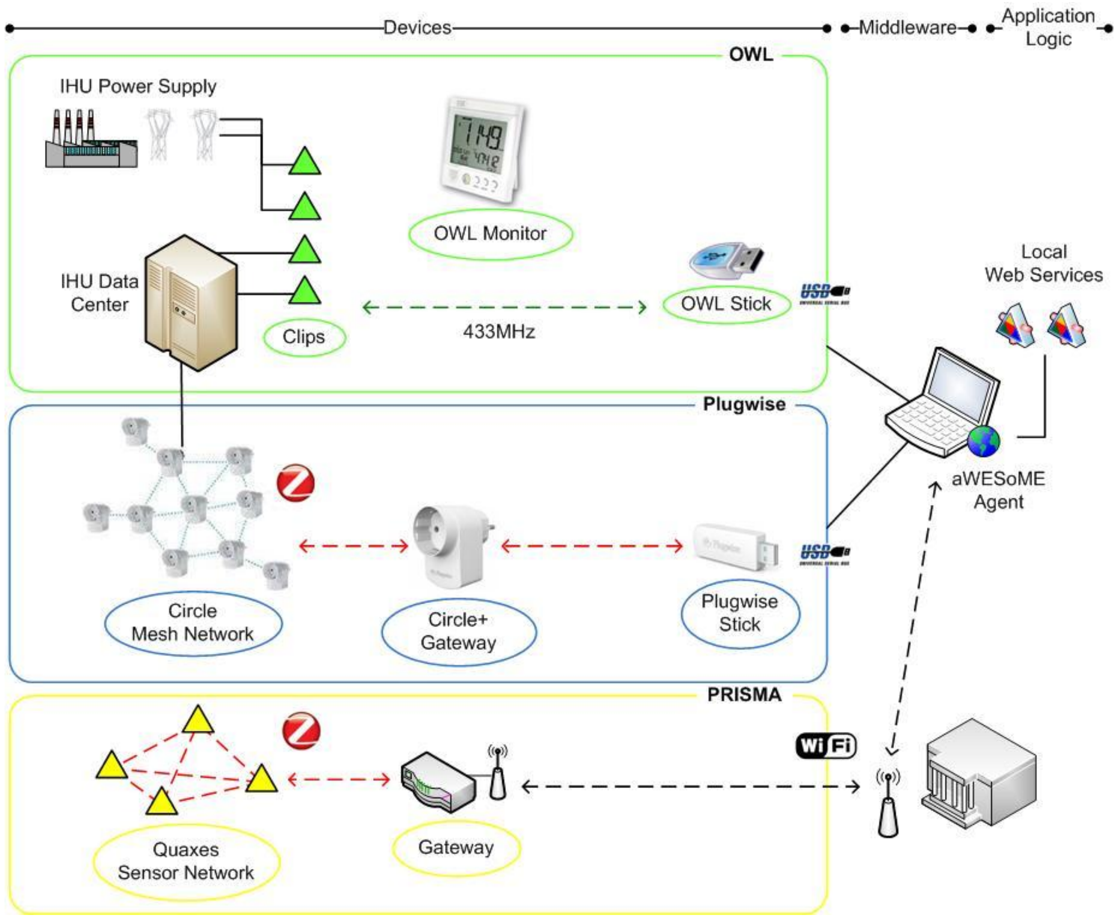


Figure 3.17: The existing topology of aWESoME

Plugwise Over Web

Plugwise Over the Web (POW) is a JAX-WS web service and contains various operations. These operations facilitate functions of a Plugwise system. To create these web service, initially the Smart IHU team investigated the protocol used by Plugwise. The operation of each function (e.g. Switch On/OFF, power measurement etc.) was investigated and java software was developed in order to execute these operations. These scripts were converted into operations of the web service and last packed in the aWESoME project.

POW uses the USB device provided by Plugwise (Stick) and the data are transferred serially using the RxTxSerial library. Therefore the *RxTxSerial.dll* and *RxTxcomm.jar* must be imported in the Agent's (web server's) JRE's path when using a Windows OS. The user must also define the Com port where the Stick is Located (e.g. COM0, COM1 etc.). In case a Linux OS is used the *RxTxSerial.so* and *RxTxcomm.jar* must be transferred in the JRE's location and the port must be defined (e.g. /dev/ttyUSB0)

PrismaWeb

PrismaWeb is web service which includes functions of the Prisma Sense system, providing the ability to access data, initially collected by the Quaxes, from the Gateways via WiFi. Initially the data were exported as Microsoft Excel documents. Afterwards the class ReadXL was developed in order to access the data in this Excel document. The PrismaWeb web service was packed in the aWESoME platform.

WebOWL

The OWL platform is similar to the PrismaWeb platform since the data is accessed by reading and registering data. In this case two open source programs, provided by the OWL API, were used in order to retrieve the data and publish them as web services. The first was OWL Server, which is executed on the computer where the USB receiver is connected and connects the API with the transmitters and the sensors. The second was ElectricOWL used to retrieve the data from OWL server using its URL and presenting those data graphically (e.g. using Charts etc.).

3.2 Power Consumption

The Smart IHU project consists of many devices installed inside the University. Most of these devices are Plugwise Circle and Circle+ which monitor the energy consumption of various devices (e.g. TV, Computers, etc.).

Future steps include the installation of additional sensors (e.g. motion, temperature, humidity etc.). In this way more data will be available making possible to the system to act according to the required conditions.

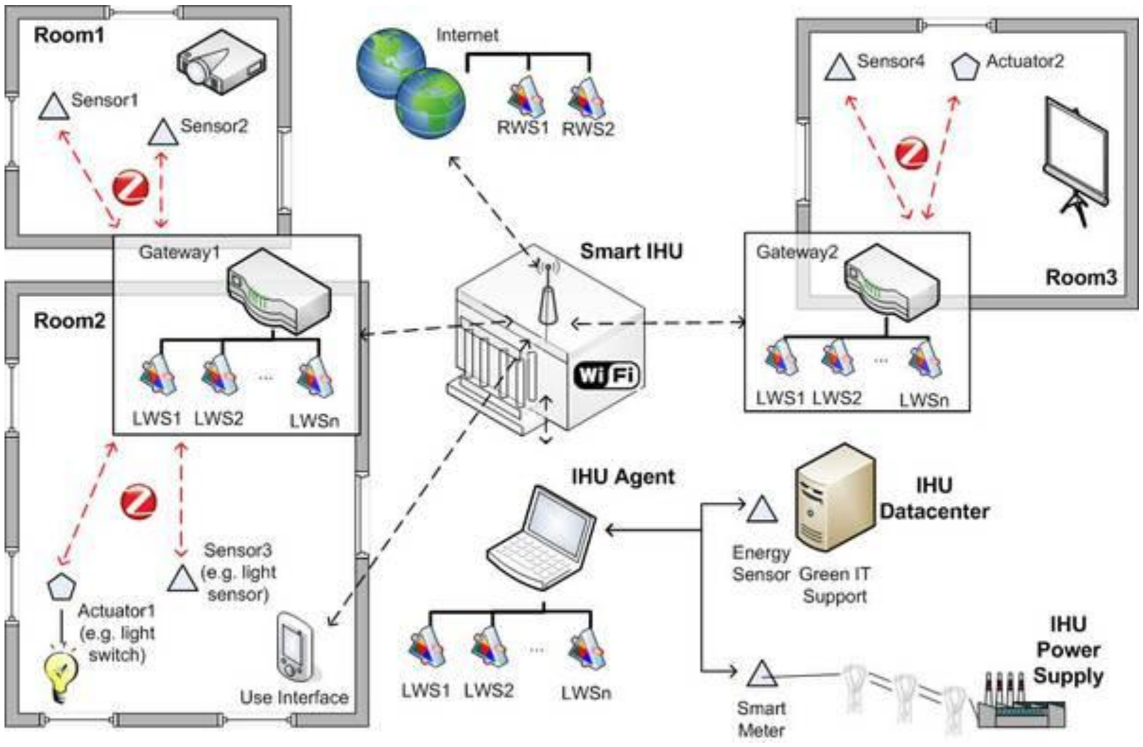


Figure 3.18: Architecture of the Smart IHU System

Figure 3.18, provided by the IHU Labs, indicates the architecture of the Smart IHU system. It is noticed that there are many hardware devices deployed in this environment. In combination these devices have certain energy consumption. The energy is consumed in two sectors, first by the sensors (i.e. the combination of sensors and transmitters) and second by the computers (i.e. gateways) where the data are collected. In the OWL and CurrentCost power monitoring system and also the PrismaSense, most of the devices are battery powered and as given by the manufacturers these devices have a very low energy consumption (e.g. the battery lifetime of the OWL devices may reach approximately 2 years). In the Plugwise system the power needed by the devices could be considered low (e.g. the power dissipation for Circle and Circle+ varies from 0.3 to 1.1 Watts, respectively for the Stick is 0.3 Watt). Even though each device has a low energy consumption in the overall system the total energy consumption of these devices could be considerable. Currently in the International Hellenic University there are research projects which focus into improving the wireless sensor network protocols in order to achieve lower energy consumption, extending the battery lifetime in the battery powered devices and reducing the energy consumed by these sensor nodes.

Additionally energy is consumed by the gateways which collect the data and offer web services. In present, these gateways are normal computers which act as web servers and the aWESoME application is deployed in these devices collecting data from the different sensor systems (e.g. Plugwise, OWL, and PrismaSense).

Table 3.1: Power dissipation in Watts, regarding different types of PCs (excluding displays), in 2007

PC Type	**Maximum**	**Idle**[11]	**Average**
"High End" used for gaming or CAD/CAM	380	320	350
Standard Desktop PC	130	70	100
Energy Efficient Desktop PC	60	40	50
Energy Saving Notebook	40	20	30

The aWESoME application does not require very high computing power and memory resources, therefore an Energy efficient Desktop PC can be used, not necessarily with high processing power. The typical power requirement of and energy efficient desktop PC is approximately 40-50Watts, shown in Table 3.1 provided by Nordin H. (Nordin, 2008, p. 5). These power values are indicative because the power dissipation of a PC is variable and depends on various factors like: the devices connected in it (e.g. an external hard drive will be powered by the PC's power supply unit, the existence of many internal hard drives requires more energy); the number of fans and ambient temperature (e.g. more fans will consume more energy additionally when the temperature is high the speed of the fans will increase, consuming more energy); the percentage of the CPU utilization etc. Since these gateways will operate providing web services, the existence of a displaying monitor will not be required during its operation, but only

[11] Idle defined: the computed is operational but not active

during configuration and maintenance. Therefore the energy consumption of the display will not be taken into consideration. Additionally this device will run in 24/7 base since it is required by some systems (e.g. in contrast to the Plugwise devices which can store data up to 10 days ,depending on the device's usage, the OWL USB receiver will lose data when the computer will be powered off). To conclude, the existence of many gateways will result in a significant amount of energy consumed. Therefore it is needed to adapt devices with lower energy consumption to act as gateways and run the aWESoME application.

3.3 Remote Control and Monitoring

Facilities at a University include the existence of many computers. These desktop PC are often distributed. Managing these devices may result increasing the overall efficiency reducing the energy waste and operational costs. The management of these computers requires three basic steps:

- *Sensing*: Remote monitoring of certain parameters (e.g. CPU utilization, Memory available, network traffic etc.).
- *Reasoning*: Estimate if a specific device is utilized or Idle, based on the previous parameters.
- *Acting*: Remotely power on or off, sleep, hibernate or wake up computers

3.4 Software tools and hardware

Possible solutions to the previously mentioned problems can be given using software tools and hardware devices. These tools will be presented in the following subsections.

3.4.1 Hardware

In order to reduce the power consumption of the gateways deployed in the Smart IHU project, these devices must be replaced with low energy consumption devices. Such devices are single board computers (SBC). There are many available SBC's in the market like: FoxBoard G20 by ACME, TS-7800 by Technologic Systems Inc.[12]. Both these devices are based on the 32-bit Advanced RISC Machine (ARM) microprocessors, specifically ARM9. Additionally these devices are also Linux Embedded and can operate on a Linux OS. There are also many other SBC's with higher Memory and computing power developed by VIA Technologies[13]. Such examples are the ITX series (e.g. mini ITX, nano ITX, pico ITX etc.) and there are a lot of projects presented in the mini-ITX[14] web site.

FoxBoard G20 Specifications

FoxBoard G20, shown in Figure 3.19, is a Linux Embedded single board computer (SBC) developed my ACME Systems in Italy. The architecture of a typical SBC is shown in Figure 3.20.

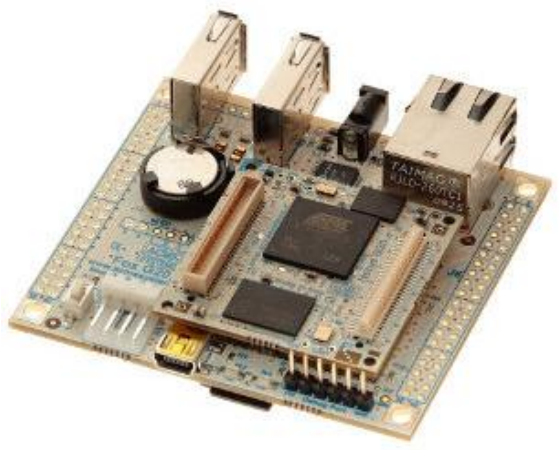


Figure 3.19: FOXBOARD G20

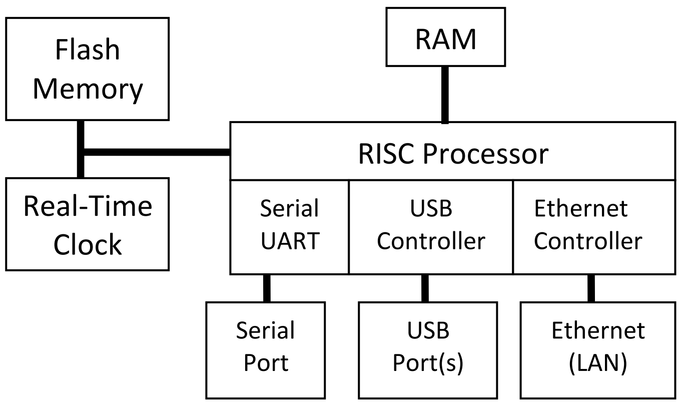


Figure 3.20: Single Board Computer Architecture

[12] http://www.embeddedarm.com
[13] http://www.via.com.tw
[14] http://www.mini-itx.com

Usually SBC are based in Reduced Instruction Set Computing (RISC) with embedded USB-Ethernet controller and Serial Ports. These microprocessors are designed with a simplified instruction set to provide a better performance by executing the instructions faster. Some examples of instruction set architectures that are based on RISC architecture are: ARM, ARC, Atmel AVR, AMD 29k, SPARC etc.

The main features of FoxBoard G20 are shown in Table 3.2, provided by (ACME Systems) and the inputs and outputs of this device are shown in Figure 3.21, provided by ACME.

Table 3.2: FoxBoard G20 technical specifications

Two 40 pin sockets pitch 2.54mm are available to plug the board on specific application carriers or add-on boards. On these pins 3.3 Volt signals are available which can be used to implement RS232/RS485/RS422, I2C, SPI, GPIO, A/D and PWM interfaces.	
✓ Built on the Atmel ARM9 @ 400Mhz CPU module Netus G20-L (included) ✓ 64MB of RAM ✓ 256KB of FLASH memory for the boot loader ✓ Up to **16GB** on bootable microSD (*) ✓ Two USB 2.0 host ports (12 Mbits) ✓ One Ethernet 10/100 port ✓ One USB device port (12 Mbits) ✓ One debug serial port (3.3v) ✓ Two serial ports (3.3v) ✓ One serial port for 4DSystems oLed displays ✓ 5VDC power supply input (compatible with PS5V1A) ✓ Real Time Clock with on-board backup battery (**)	✓ GPIO lines (3.3v) ✓ 4 A/D converter lines ✓ I2C ✓ SPI ✓ Built-in quad power supply Netus PS1 module ✓ Same footprint and pin-out of the old FOX Board LX832 ✓ Fully mechanical compatible with TUXCASE and FOXCASE ✓ Temperature range: -15 to +70 Celsius degree (°C) ✓ Average power consumption: 80 mA @ 5V (0.4 Watt) without microSD, Ethernet link, USB devices or other peripherals.
(*) the microSD card memory is optional (**) the lithium backup battery for RTC is optional	

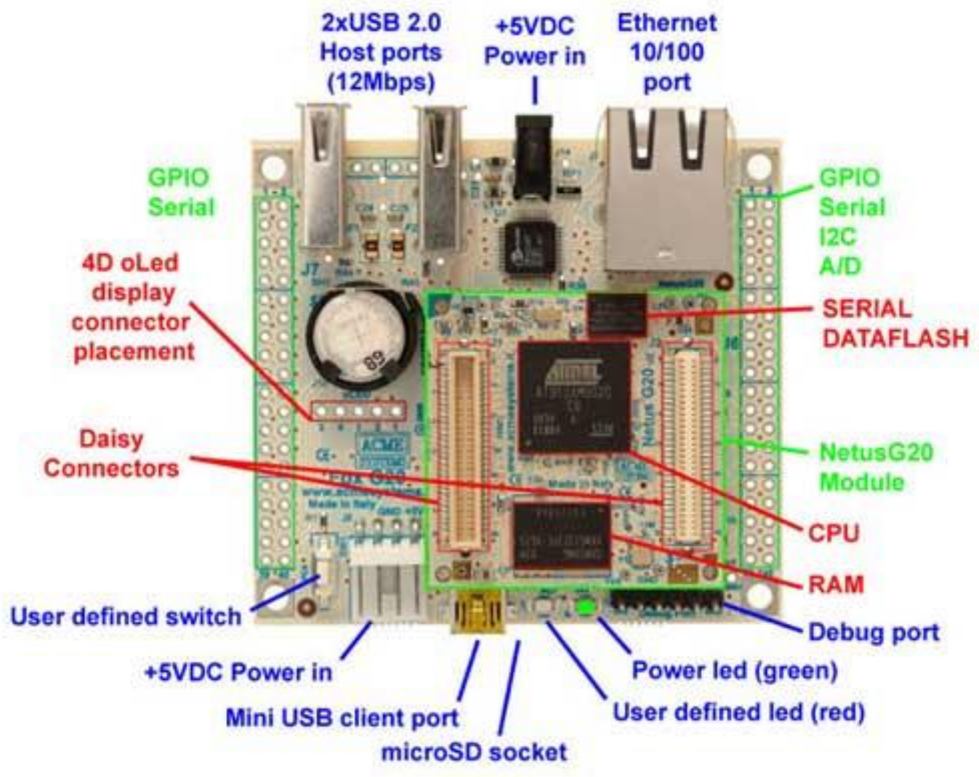


Figure 3.21: FoxBoard G20 -Input/outputs

FoxBoard G20 is a Linux embedded device which can operate on a Debian operating system. Two versions are available Debian Lenny and Debian Squeeze. The OS boots from the microSD drive. Additionally many programming languages like C and Python could be used to develop customized applications. The features that FoxBoard provides can be exploited to implement this device in existing wireless sensor network infrastructure. This device can be utilized as a web server (i.e. a gateway) where the data could be aggregated and also the existence of A/D converters can be used to gather data from analog sources (i.e. sensors etc.). FoxBoard costs approximately €185.

TS-7800

Technologic Systems has developed many SBCs. One of these is TS-7800 shown in Figure 3.22, provided by Technologic Systems. This is also a Linux Embedded device, similar to FoxBoard, based on an ARM9 processor. The technical specifications of this device are shown in Table 3.3.

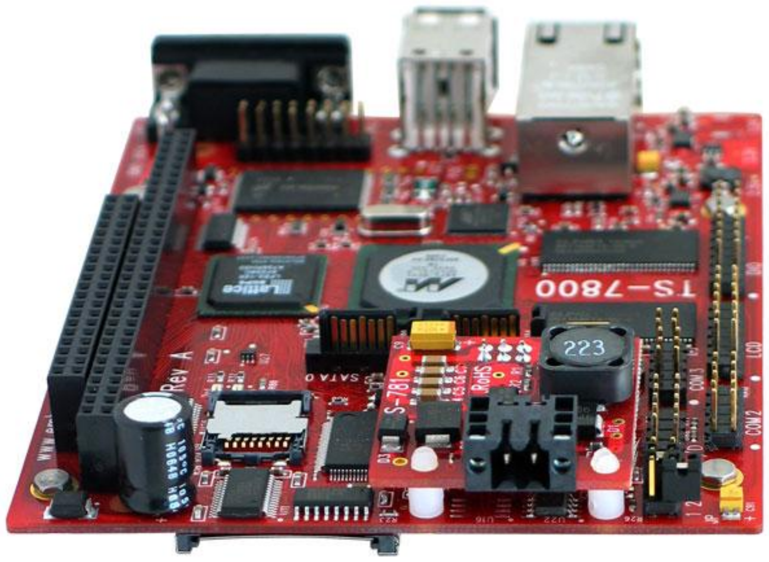


Figure 3.22: TS-7800 a SBC developed by Technologic Systems

Table 3.3: TS-7800 Technical Specifications

✓ 500MHz ARM9 CPU	✓ 110 GPIO (86 as a PC/104 bus)
✓ Internal PCI bus, PC/104 connector	✓ Matrix Keypad and text LCD support
✓ 128MB DDR-RAM	✓ Optional Temp Sensor, RTC, and WiFi
✓ 512MB NAND Flash (17MB/s)	✓ Low-power (4W @ 5V)
✓ 12,000 LUT programmable FPGA	✓ Sleep mode (uses 200 microamps)
✓ 2 SD Card slots (1 micro-SD, 1 full-SD)	✓ Watchdog Timer
✓ 2 SATA ports	✓ Fan less Operation from -20°C to +70°C
✓ 2 USB 2.0 480Mbit/s Host/Device	✓ Boots Linux in 0.69s from Flash
✓ Gigabit Ethernet, 10/100/1000 speeds	✓ Kernel 2.6 and Debian Linux
✓ 5 10-bit ADC channels	✓ Eclipse IDE out-of-the-box
✓ 10 serial ports, 2 optional RS-485	

TS-7800 has a higher processor clock in comparison to FoxBoard and the instructions are executed faster therefore providing a slightly higher processing power. The most noticeable feature is the DDR-RAM which is 128MB, the double size in comparison to FoxBoard. TS-7800 costs approximately €167.

ITX Series

The ITX Series devices are embedded boards developed by VIA Technologies. The size of the ITX series in comparison to other products developed by the same manufacturer (e.g. micro-ATX, Flex-ATX) is shown in Figure 3.23. In comparison to the two previous devices the ITX series provide a higher processing power and higher memory resources. These more sophisticated boards have an implemented graphic card with a VGA output which facilitates the use of display monitors.

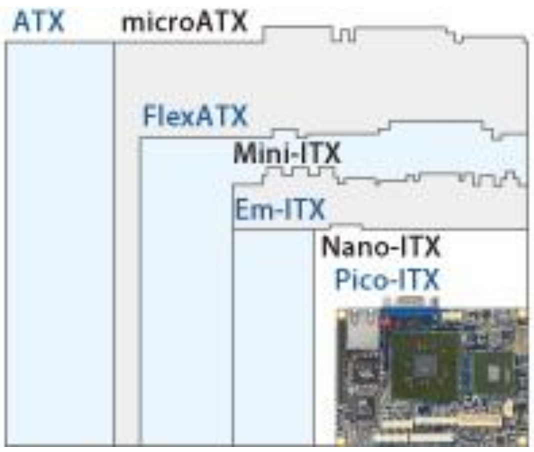


Figure 3.23: ITX Series size

Additionally these devices can operate using an operating system like: Microsoft Windows XP or Vista; or a Linux OS (e.g. Ubuntu, Debian etc.).

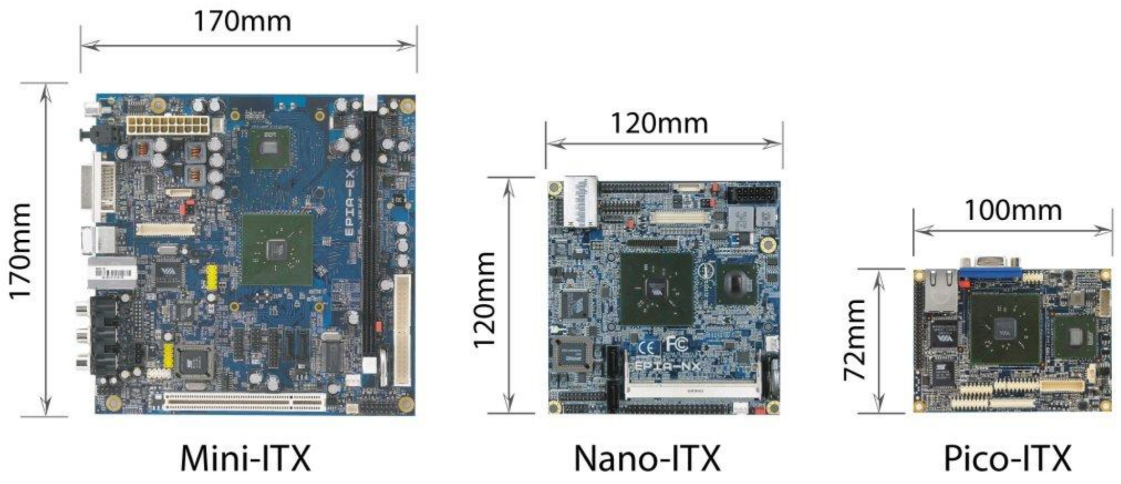


Figure 3.24: Mini, Nano and Pico ITX dimensions

The dimensions of Mini, Nano and Pico ITX are shown in Figure 3.24, provided by windowsfordevices.com. The technical specifications of Pico-ITX, provided by VIA Technologies, are shown in Table 3.4. In regards to the power dissipation of a Pico ITX, the required maximum (peak) power is 25 Watts, as shown by min-itx.com[15], but the system may consume less than 25Watts. The ITX series are significantly more powerful than FoxBoard and TS-7800. The computing power is tripled and the memory increased from 64MB (FoxBoard) and 128MB (TS-7800) up to 2GB. Even though there are so many advantages there is a disadvantage. Pico ITX has higher power dissipation in comparison to FoxBoard and TS-7800, since the power consumption in the later devices is estimated less than 5Watts.

Table 3.4: Pico ITX technical specifications

Model Name	**EPIA-P710-10L**
Processor	1.0 GHz VIA C7®
Chipset	VIA VX800 Unified Digital Media IGP chipset
System Memory	1 x DDR2 533/667 SODIMM socket Up to 2GB memory size
VGA	Integrated VIA Chrome9™ HC3 DX9 3D/2D graphics with MPEG-2 video decoding acceleration
Onboard IDE	1 x UltraDMA 133/100 pin connector with 2.0mm 44-pin
Onboard Serial ATA	1 x SATA connector 1 x SATA power connector (5V)
Onboard LAN	1 x VIA VT6122 Gigabit LAN controller
Onboard Audio	1 x VIA VT1708B High Definition Audio Codec
Onboard I/O Connectors	2 x SUMIT QMS connectors (3 USB, LPC, 2 PCIe x1, PCIe x4, SM Bus and SPI),1 x Giga LAN pin header,1 x Audio pin connector for Line-out, Line-in, Mic-in,1 x Front panel pin header,1 x CRT pin header,1 x Single-channel, LVDS pin connector,(powered with selectable 5V/3V),1 x CPU fan connector, 1 x PS/2 KB/MS pin header, 1 x +12V DC-in 2-pin jack with lock

BIOS	Award BIOS 4/8Mbit flash ROM
System Monitoring & Management	Keyboard Power-on, Timer Power-on System power management, AC power failure recovery Wake-on LAN, Watch Dog Timer
Operating System	Windows XP, Windows Embedded CE, Windows Embedded Standard, Linux
Operating Temperature	0°C ~ 50°C
Operating Humidity	0% ~ 95% (relative humidity; Non-condensing)
Form Factor	Pico-ITXe (12-layer) 10 cm x 7.2 cm

The previous comparisons were made with Pico ITX the smaller device of the ITX series. Nano ITX and Mini ITX offer higher computing power and with a raising power consumption. Most of the ITX series are used in building low power PC (i.e. "green"

[15] http://www.mini-itx.com/86950182

PC's). There are many projects available in the mini-ITX[16] web site. The price of a Pico ITX board is approximately €181 and the case, including the power supply unit (PSU), costs about €105, in total €286.

3.4.2 Software

To solve the existing problems and improve the system, additional software tools are needed: first a web server that could be used to run the aWESoME application in a SBC; and second and integrated developing platform to develop JAX-WS web services and applications.

Apache Tomcat 6.0

Apache Tomcat is an open source Servlet/JSP container. The 6.0 version implements specifications of the Java Community Process like Servlet 2.5 and Java Server Pages 2.1. There are also other features included which make Tomcat 6.0 a useful platform for developing and deploying web applications and web services.

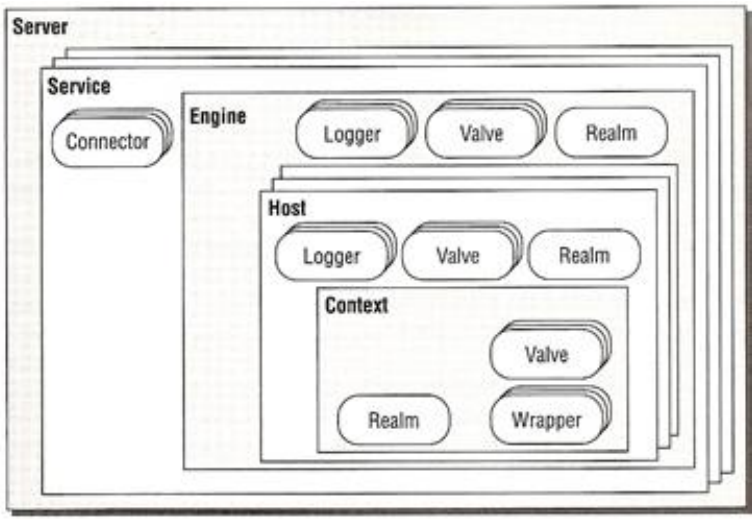


Figure 3.25: Apache Tomcat 6.0 architecture

The architecture and the key elements of Tomcat 6.0 are shown in Figure 3.25, provided by datadisk.co.uk[17].

[16] http://www.mini-itx.com/projects.asp
[17] http://www.datadisk.co.uk/html_docs/java_app/tomcat6/tomcat6_architecture.htm

Table 3.5: Tomcat 6.0 key elements

Server	*The server is Tomcat and represents the whole container*
Service	*A service is an intermediate component which lives inside a Server and ties one or more Connectors to exactly one Engine. The service is responsible for accepting requests, routing them to the specified Web application and specific resources and then returning the result of the processing of the request, they are the middle man between the client's web browser and the container.*
Engine	*The engine represents request processing pipeline for a specific Service. As a Service may have multiple Connectors, the Engine received and processes all requests from these connectors, handing the response back to the appropriate connector for transmission to the client.*
Host	*A Host is an association of a network name, e.g. www.yourcompany.com, to the Tomcat server. An Engine may contain multiple hosts, and the Host element also supports network aliases such as yourcompany.com and abc.yourcompany.com*
Connector	*A Connector handles communications with the client. There are multiple connectors available with Tomcat. These include the HTTP connector which is used for most HTTP traffic, especially when running Tomcat as a standalone server, and the AJP connector which implements the AJP protocol used when connecting Tomcat to a web server such as Apache HTTPD server*
Context	*A Context represents a web application. A Host may contain multiple contexts, each with a unique path*

Table 3.5 describes the key elements of Tomcat 6.0, the data are provided by the Apache Tomcat Web Site[18] and datadisk.co.uk[17] , in the later link there is also a detailed description about the other elements (e.g. Logger, Valve, Realm etc.).

Tomcat 6.0 can be used to deploy the aWESoME application in a web server. There are also other servers that can be used for the same purpose, one of them is an open source application server named Glassfish. A comparison between the two servers is shown in ninthavenue.com.au[19]. The general scope is to deploy the aWESoME application in a low power device like FoxBoard or TS-7800. In this case Glassfish, which may offer some functionalities that Tomcat does not, cannot be used since according to ORACLE's glassfish server system requirements, the minimum RAM available must be 100MB but FoxBoard has only 64MB of RAM available.

NetBeans IDE 7.0

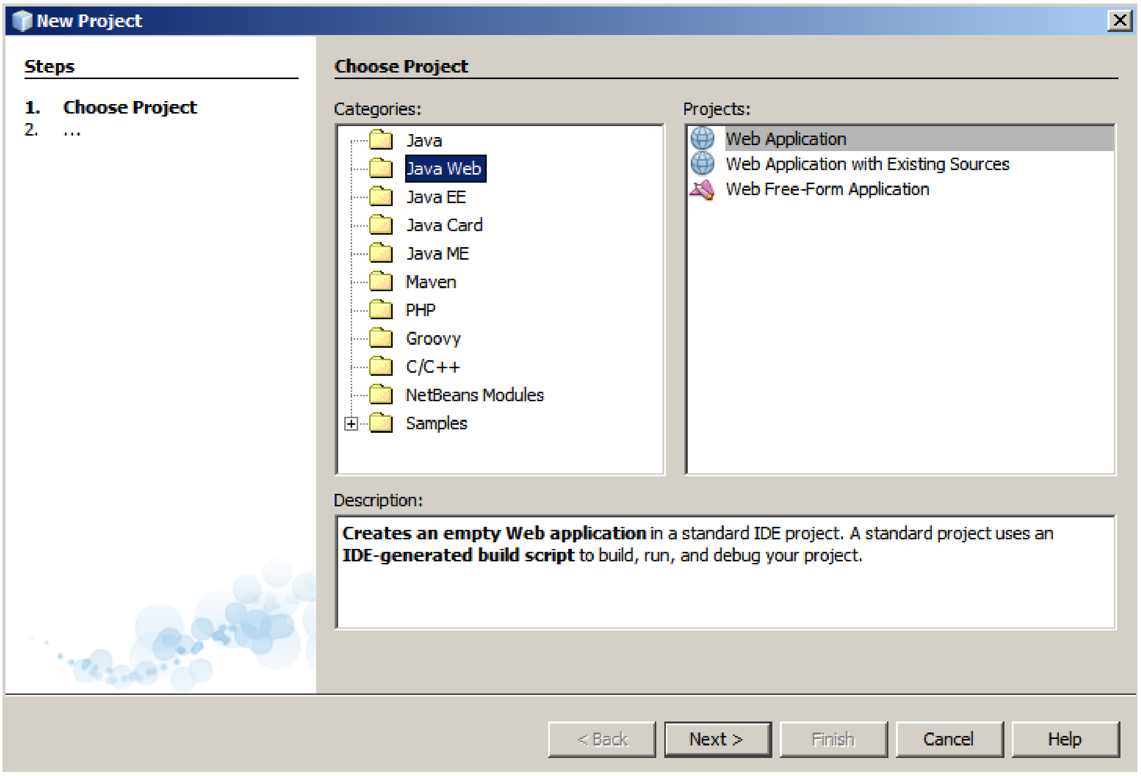
There are different ways to develop web applications and web services. One of them is using an integrated development environment (IDE) like NetBeans IDE. NetBeans is an open source Java IDE, initially developed by Sun Microsystems. It supports several programming languages (e.g. PHP, JavaFX, C/C++, JavaScript, etc.) and frame-

[18] http://tomcat.apache.org/tomcat-6.0-doc/architecture/overview.html
[19] http://www.ninthavenue.com.au/blog/glassfish-vs-tomcat

works. This IDE provides the tools to create professional desktop, enterprise, web and mobile applications with the Java platform, C/C++, PHP, JavaScript and Groovy. The Highlights of NetBeans are shown in Table 3.6, provided by the NetBeans community (NetBeans Community, 2011).

Table 3.6: NetBeans Highlights

Java Desktop Applications *Create professional standards-based user interface with the NetBeans Swing GUI Builder. Save years of work by building Java Swing desktop applications with the NetBeans Platform framework.*	Java Enterprise and Web Applications *Build web applications using CSS, JavaScript, and JSP: Support for frameworks includes JSF (Facelets), Struts, Spring, Hibernate, and a full set of tools for Java EE 6, CDI, GlassFish 3, EJB, and web services development.*
Dynamic Languages *Benefit from one combined tool that supports PHP (Zend and Symfony), CSS, Groovy and Grails, and JavaScript.*	C and C++ Development *Edit, profile, and debug C/C++ applications, and make the most of multiple project configurations, remote development, and packaging.*
Visual Mobile Development *Create, test and debug applications that run on mobile phones, set-top boxes, and PDAs using JavaFX Mobile and the Java ME SDK 3.0 Platform.*	



Creates an empty Web application in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.

Figure 3.26: NetBeans main categories

Figure 3.26 indicates the main categories of the applications that can be developed using NetBeans IDE 7.0. The categories of interest in this work will be: the Java, for developing simple java applications; the Java Web and Java EE, used to develop standards-based web and enterprise applications which can be accessed from a wide range of clients such as web browsers, mobile devices and more. Additionally create Java EE 6 applications with support for all relevant Java EE 6 technologies.

As shown in Table 3.6, to test and deploy Java EE and Java Web applications, NetBeans 7.0 by default uses the GlassFish Server 3.1. It is possible to add also other servers installed like Tomcat. NetBeans may run/stop these servers and even deploy applications or web services in them, while testing/running/deploying web applications.

4 Contribution

After investigating the theoretical concepts, infrastructure and the main problem possible improvements were defined. The content of this chapter regards the implementation of the suggested solutions and improvements.

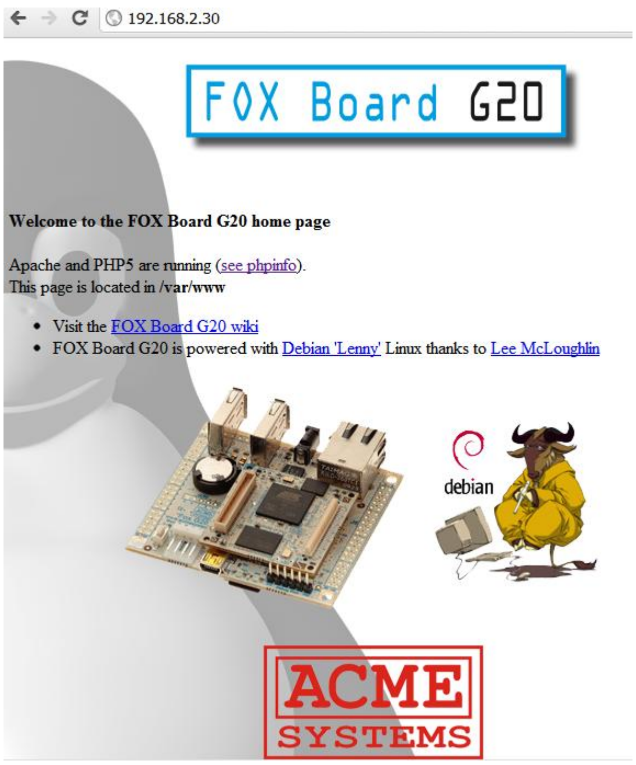
4.1 FoxBoard Gateway

FoxBoard was the device available to implement and deploy the aWESoME web application. The OS used in FoxBoard was Debian Lenny. This OS is booted via MicroSD card the detailed procedure to create such a card is described in Appendix B. After inserting the bootable MicroSD card and powering on the device, the OS will load in a short and after approximately 30 seconds and FoxBoard will be reachable via LAN (i.e. the default web server will be available and the user can access FoxBoard via a browser). The successful boot of the OS is indicated by the blinking LED (red) (e.g. in a successful OS load the led will blink SHORT ON - SHORT ON – PAUSE). By default the OS was configured to receive automatically the IP address from a DHCP server available in the network. The IP address that FoxBoard retrieved could be discovered by searching the DHCP attributes or by using software which scan all the IP addresses of the subnet and shows their computer names. Such software is Angry IP[20] scanner. After accessing the device the user can configure a static IP address to this device, the detailed procedure is also shown in Appendix B.

In the provided Linux version, by ACME Systems, Lighttpd[21] is the default web server. This server runs in port 80 and by accessing FoxBoard via a browser using its IP address the user can have access in the default web page shown in Figure 4.1. This page provides links to the FoxBoard G20 wiki, to the web page of Debian Lenny and a link

[20] http://www.angryip.org/w/Download
[21] http://www.lighttpd.net/

named “see phpinfo” which provide information about the system (e.g. the OS version, build date, etc.) and documentation about the PHP configuration.



Welcome to the FOX Board G20 home page

Apache and PHP5 are running (see phpinfo).
This page is located in /var/www

- Visit the FOX Board G20 wiki
- FOX Board G20 is powered with Debian 'Lenny' Linux thanks to Lee McLoughlin

Figure 4.1: FoxBoard’s default web page

4.1.1 Installation and Configuration

FoxBoard does not include a display and there are two ways to access it. First using the Debug Port Interface (DPI) and second via the LAN, using a Secure Shell (SSH) connection. The DPI can also be used to install ad hoc applications developed in various programming languages (e.g. python, C etc.). In both cases the manufacturer company suggests the use of the putty.exe[22] which is a freeware utility that runs on windows and can emulate a serial terminal or manage an SSH session. To transfer files the manufacturer suggests WinSCP[23] which is free Windows SCP client. SCP is the protocol used to browse remotely the file system on a Linux system and transfer files over the network. Alternatively, a computer which runs on a Linux version can be used to facilitate both SSH and SCP via the terminal. This was the preferred method selected in this

[22] http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
[23] http://www.winscp.net

work. Therefore a Linux Ubuntu 10.10 was used installed in a USB flash drive (i.e. similar to a Live CD) to be able to run it in different computers.

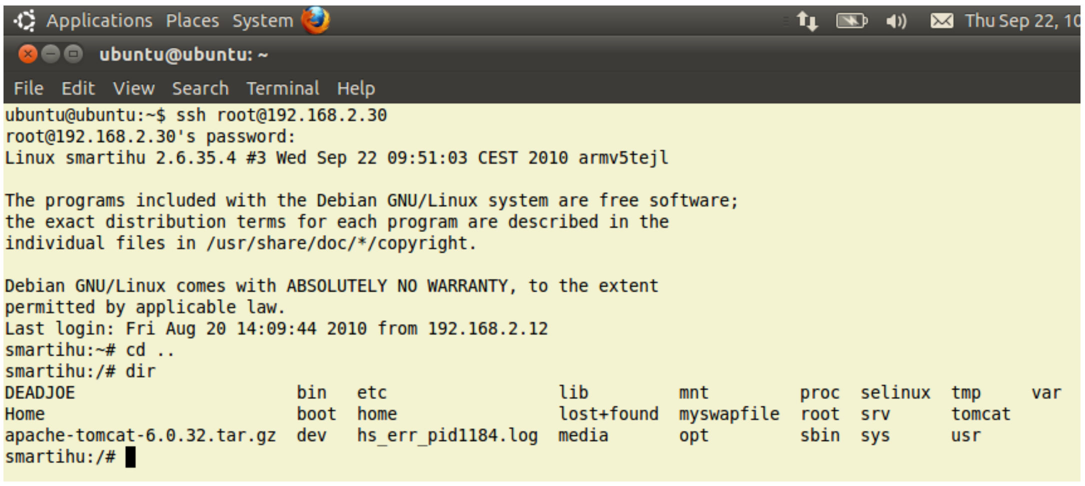
After discovering the IP address of FoxBoard, the user can access it typing:

```
hostname:/ssh root@FOX_IP_Address
```

An Example:

```
smartihu:/ssh root@192.168.2.30
```

Afterwards the root password will be required, which by default is *netusg20*. Once the correct password is entered the user has access to the terminal of FoxBoard's Debian Lenny OS. An example of a SSH connection is shown in Figure 4.2, where the user has accessed to the contents of the root folder. It can be noticed that the hostname of Ubuntu is *ubuntu@ubuntu* and the hostname of FoxBoard is *smartihu*.


Applications Places System
Thu Sep 22, 10
ubuntu@ubuntu: ~
File Edit View Search Terminal Help

```
ubuntu@ubuntu:~$ ssh root@192.168.2.30
root@192.168.2.30's password:
Linux smartihu 2.6.35.4 #3 Wed Sep 22 09:51:03 CEST 2010 armv5tejl

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Aug 20 14:09:44 2010 from 192.168.2.12
smartihu:~# cd ..
smartihu:/# dir
DEADJOE                     bin   etc                 lib         mnt         proc  selinux  tmp     var
Home                        boot  home                lost+found  myswapfile  root  srv      tomcat
apache-tomcat-6.0.32.tar.gz dev   hs_err_pid1184.log  media       opt         sbin  sys      usr
smartihu:/# █
```

Figure 4.2: Example of a SSH connection

Basic setting and configurations of FoxBoard (e.g. setup date and time, change hostname etc.) are shown in Appendix B. Additionally there are also many examples provided by the manufacturer's web site (ACME Systems).

To transfer files remotely via the network, the SCP protocol was used. Examples are provided bellow for different scenarios where the remote host's IP address is 192.168.2.30.

To transfer a file named file.txt from a remote machine to my machine:

```
scp root@192.168.2.30:/home/smartihu/Desktop/file.txt home/me/Desktop/file.txt
```

To transfer a file, named sample.war, from my machine to a remote machine:

```
scp sample.war root@192.168.2.30:/usr/local/tomcat/webapps/
```

In addition the remove file (rm) and remove directory (rmdir) commands were be used to remove files and directories.

As presented in previous chapters the middleware used I the Smart IHU project is aWESoME which is a web application based Java EE 5 platform and provides JAX-WS web services. In order to deploy such an application, a web server which implements the Java Servlet and provides Java HTTP web server environment for java code to run, is needed. Lighttpd, the default web server, does not provide these features, in contrast with Apache Tomcat. Both these servers are frequently used together running in different ports and often linked between them (e.g Lighttpd proxy to Tomcat).

Apache Tomcat 6.0

Before installing Apache Tomcat, it is required to install the default jdk or jre. Once FoxBoard has access on the internet, to perform this installation the following command must be executed:

```
apt-get install openjdk-6-jdk
```

After the successful installation of the JDK, the next step is to install Apache Tomcat. The selected version was 6.0.32, this version is available in many locations. Two of them are suggested bellow

```
wget http://apache.hoxt.com/tomcat/tomcat-6/v6.0.32/bin/apache-tomcat-
6.0.32.tar.gz. tar xvzf apache-tomcat-6.0.32.tar.gz
```

or

```
wget http://www.ecoficial.com/apachemirror/tomcat/tomcat-6/v6.0.32/bin/apache-
tomcat-6.0.32.tar.gz
```

The next step is to extract the tape archive (tar) using the following command.

```
tar -xzvf apache-tomcat-6.0.32.tar.gz
```

Afterwards move the extracted folder in a different location /usr/local/tomcat typing:

```
mv apache-tomcat-6.0.32 /usr/local/tomcat
```

By executing the files *startup.sh* and *shutdown.sh*, Tomcat can start and stop respectively. These files are found in the location: /usr/local/tomcat/bin. An example, which starts and stops Tomcat, is shown below:

```
sh /usr/local/tomcat/bin/startup.sh

sh /usr/local/tomcat/bin/shutdown.sh
```

The previous manual procedure can be done automatically, creating an executable script and placing it the /etc/init.d directory, which contains scripts that start/stop various applications of the system. There are many websites which provide such examples one of them is howtogeek.com[24]. The steps are shown below.

First the creation of a file named tomcat in the /etc/init.d/ location. The content of the file is:

```
# description: Auto-starts tomcat
# processname: tomcat
# pidfile: /var/run/tomcat.pid

export JAVA_HOME=/usr/lib/jvm/java-6-openjdk


case $1 in
start)
        sh /usr/local/tomcat/bin/startup.sh
        ;;
stop)
        sh /usr/local/tomcat/bin/shutdown.sh
        ;;
restart)
        sh /usr/local/tomcat/bin/shutdown.sh
        sh /usr/local/tomcat/bin/startup.sh
        ;;
esac
exit 0
```

Initially this script will define the JAVA_HOME location.

The next step is to make this file executable by running:

```
chmod +x /etc/init.d/tomcat
```

Finally, symbolic links must be created with the startup folders. The following commands were used:

```
ln -s /etc/init.d/tomcat /etc/rc1.d/K99tomcat

ln -s /etc/init.d/tomcat /etc/rc2.d/S99tomcat
```

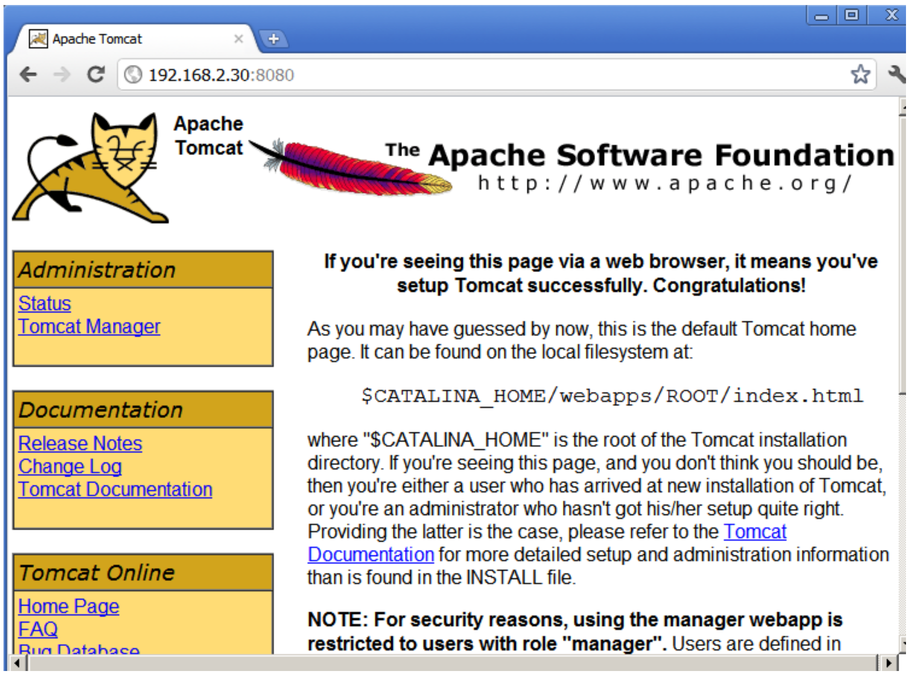
Now Tomcat should be able to start automatically on system startup. In case it is needed to start/stop/restart tomcat the following commands can be used:

```
/etc/init.d/tomcat start
/etc/init.d/tomcat stop
```

[24] http://www.howtogeek.com/howto/linux/installing-tomcat-6-on-ubuntu/

```
/etc/init.d/tomcat restart
```

After the previous steps, it is possible to test if tomcat is running using a browser. By default tomcat runs in port 8080, the port number can be changed by the user. An example of accessing tomcat via a browser is shown in Figure 4.3.


Apache Tomcat

192.168.2.30:8080

Apache Tomcat

The Apache Software Foundation
http://www.apache.org/

Administration
Status
Tomcat Manager

Documentation
Release Notes
Change Log
Tomcat Documentation

Tomcat Online
Home Page
FAQ
Bug Database

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.html
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the manager webapp is restricted to users with role "manager". Users are defined in

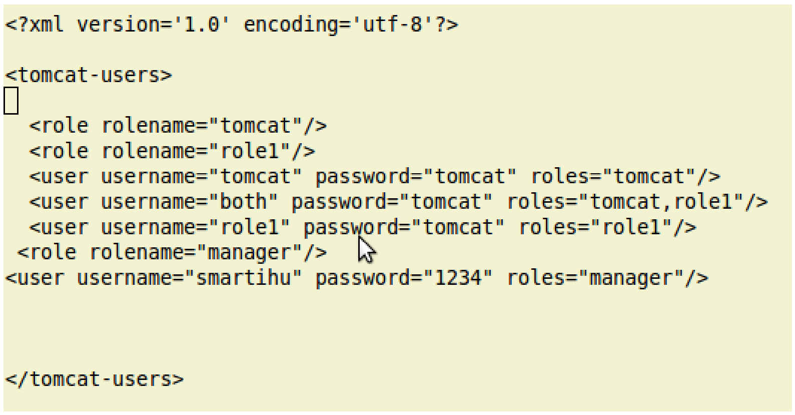
Figure 4.3: Testing Tomcat 6.0.32

Apache Tomcat Management

Tomcat can be managed using the Tomcat Manager which is a user friendly graphic interface. To gain access to this interface it is required to import a user account. Most of the setting of tomcat can be made by modifying XML files. In the previous case the file that must be modified is found in the usr/local/tomcat/conf# location. The modification of this file can be made using nano and running the following command:

```
smartihu:/usr/local/tomcat/conf# nano tomcat-users.xml
```

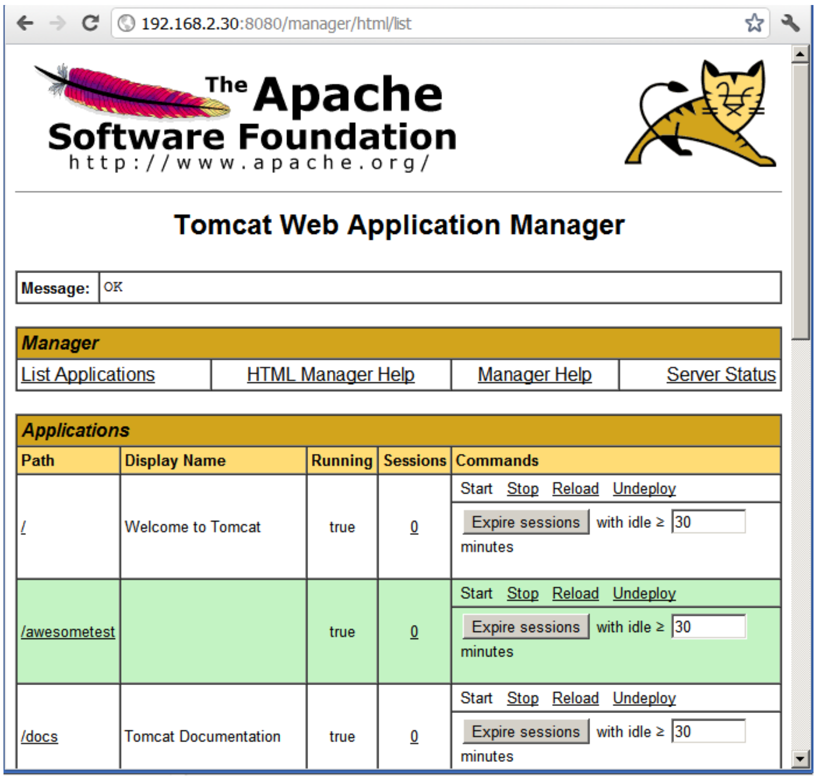
An example of this xml file is shown in Figure 4.4. This file provides the ability to select different roles, which provide different type of access to each user. In the example below a user named *smartihu* was entered with a *manger* role and *1234* as password. Once this file is saved in the same location, the user can log in the tomcat manager found in the initial page shown in Figure 4.3.



```
<?xml version='1.0' encoding='utf-8'?>

<tomcat-users>

  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
 <role rolename="manager"/>
<user username="smartihu" password="1234" roles="manager"/>



</tomcat-users>
```

Figure 4.4: Tomcat-users xml file

Once the user has logged in successfully he has access to the manager's interface, shown in Figure 4.5. In this interface a list of the Applications is shown to the manager where it is possible to control these applications (e.g. Start, Stop, Reload, Undeploy, Deploy), documentation regarding to the manager (e.g. Manager Help), and the Server Status which provides information about the Server's complete status (e.g. the OS version, the processor's architecture and additional parameters like the free memory, bytes sent/received, Max processing time, errors etc.).



192.168.2.30:8080/manager/html/list

The Apache Software Foundation
http://www.apache.org/

Tomcat Web Application Manager

Message:	OK

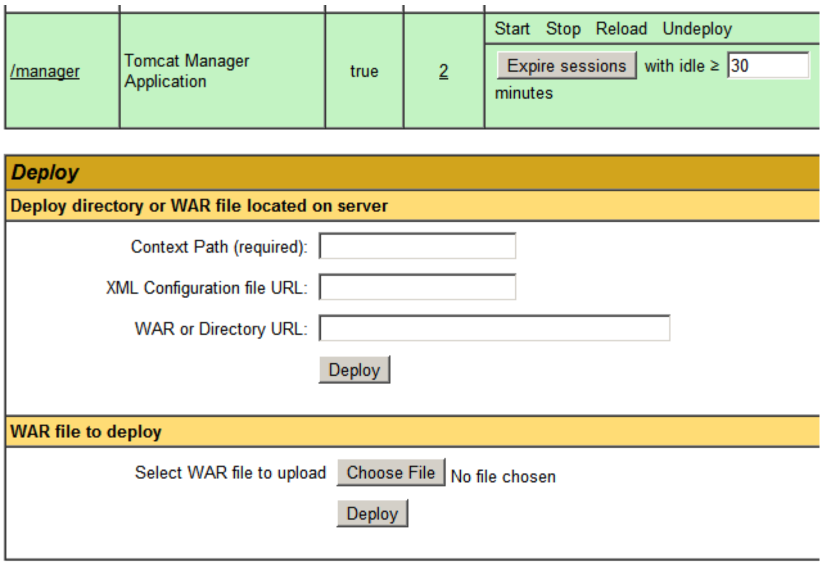
Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

Path	Display Name	Running	Sessions	Commands
/	Welcome to Tomcat	true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes
/awesometest		true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes
/docs	Tomcat Documentation	true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes

Figure 4.5: Tomcat Manager's interface

Deploying the web application

Once the application was developed, NetBeans was configured to create a .war package when building the application. This is actually a Web Application ARchive (WAR), a format developed by Sun Microsystems, which may contain the components of a web application or web service. These components are: Java Server Pages (JSP), Java Servlets, Java Classes, XML files, tag libraries and static HML pages. There are two ways to deploy WAR files in Tomcat. The first is simply to copy the war file in the *$CATALINA_BASE/webapps*[25] , under the condition that the attribute *liveDeploy* (or *autoDeploy* in other versions) is activated, the application will be deployed automatically. In case *liveDeploy* is deactivated, after copying the WAR file in the *webapps*, it is required to restart Tomcat. The attribute *liveDeploy* (or *autoDeploy*) is found in the location *$CATALINA_BASE/conf/server.xml*. In the same xml file it is possible to setup the port where the server runs.



/manager	Tomcat Manager Application	true	2	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Deploy

Deploy directory or WAR file located on server

Context Path (required):

XML Configuration file URL:

WAR or Directory URL:

Deploy

WAR file to deploy

Select WAR file to upload Choose File No file chosen

Deploy

Figure 4.6: Deploying of WAR archives using Tomcat Manager

Another way to deploy a WAR archive is by using Tomcat Manager. Once the manager has logged in, on the List Applications, by scrolling down, two ways are provided to deploy a WAR file, shown in Figure 4.6. One way the WAR file can be deployed choosing a file from the computer where Manager is accessed and pushing Deploy.

[25] The Catalina base is the main location of Tomcat in the FoxBoard's example is /etc/local/tomcat

The other way a directory or war file located on a server can be deployed defining the path or the URL.

Troubleshooting

The main significant problems encountered when deploying applications was not being able to access the WSDL files after a successful deployment and the hardware failure of FoxBoard when deploying aWESoME.

Initially to test Tomcat 6.0.32, WAR files developed by various systems with different parameters were deployed. Even though, these files were deployed successfully and the home JSP page was accessed, it was not possible to open the WSDL files of each web service. This meant that only the JSP page was running and the web services were unavailable. To address this problem a simple web application, named CalculatorWS, was developed under the directions of a tutorial from netbeans.org[26] . The web application (i.e. web service) was created based on the Glassfish server, including a client application. Afterwards was deployed locally using NetBeans Glassfish server, following the instructions given in the tutorial, it was observed that it worked perfectly. Once the WAR file was deployed on Tomcat, the application did not run. The problem was that the web application was developed based on the Glassfish Server. When the web application was developed defining Tomcat 6.0 as a Server it was noticed that additional libraries were added by net beans (e.g. Metro 2.0, Tomcat 6.0 including many jar files etc.). Afterwards the web application was tested remotely (i.e. the application was deployed in a PC and the client in a different PC) operating as desired . As a result of adding these libraries the size of the file increased significantly, approximately about 6MB more in comparison to the .war based on Glassfish. In case of the aWESoME web application, the size of the WAR file was approximately 10MB when developed using Glassfish and when developed using Tomcat 6.0 the size of the WAR file rose up to 16MB.

FoxBoard is a device with insufficient resources (i.e. low memory resources and an ARM9 processor with a 400MHz clock) therefore a second problem occurred. This problem was faced when deploying the 16 MB aWESoME web application. It was con-

[26] http://netbeans.org/kb/docs/websvc/jax-ws.html

sidered more serious since the FoxBoard project was in jeopardy. During this deployment, after approximately two to three minutes, FoxBoard stopped responding on the SSH connection and the operation led (i.e. red led which indicate if the OS has booted and is running propriety) was blinking in an unusual manner. To address this problem the *top* command was used. This command is used in Linux to display the top CPU processes that are running. During the monitoring of the processes it was noticed that when the deployment began, in Tomcat, the Java Process was using the 90 to 99.3 % of the CPU, as time passed the memory percentage allocated by Java rose gradually and after approximately 2-3 minutes the memory percentage reached 76% the system was starting to stop responding. These observations provided the conclusion that after a certain time FoxBoard suffered of memory starvation. The RAM memory is 64MB and there is no extension slot available. A solution, provided by FoxBoard's community[27], came on using a Swap file. The objective was to create a 100MB file and use swap on to activate it.

Initially create the file:

```
hostname:/dd if=/dev/zero of=/myswapfile bs=1024 count=102400
```

After this, it is required to setup the swap space:

```
hostname:/mkswap /myswapfile
```

Immediately turn it on:

```
hostname:/swapon /myswapfile
```

Afterwards the swap file is on and can be used as RAM. If the system is rebooted this memory cannot be used. To make this memory available after a system reboot, the following command must be executed:

```
hostname:/ nano /etc/fstab
```

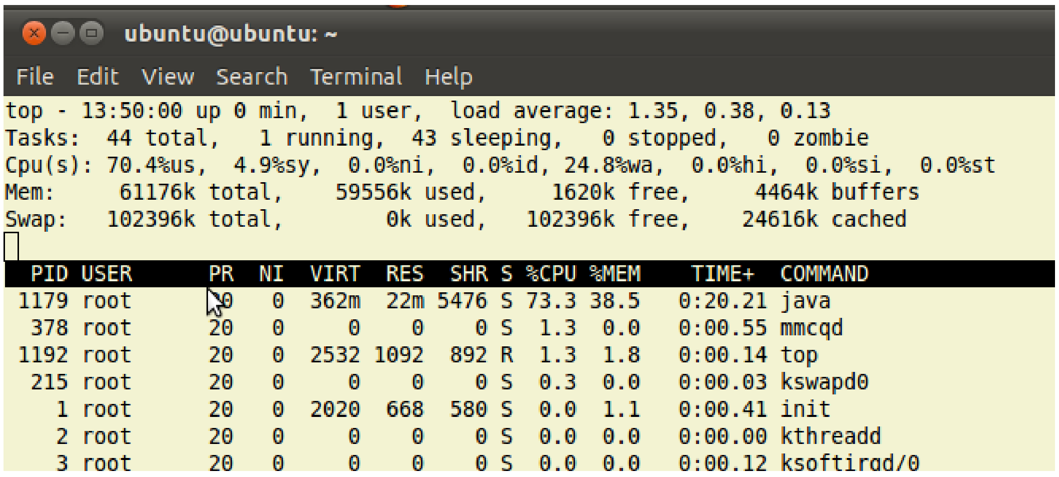
and inside the fstab file insert the following line:

```
/myswapfile swap swap defaults 0 0
```

Even though the memory rose from 64MB to 164MB there is a very important drawback. The Computer writes and reads very often in the RAM. RAM memory has a

[27] http://www.asksander.com/?p=126

lower access time and is faster in comparison to Flash memory. If a swap file is created in the microSD card, which is a flash memory, it means that this solution could reduce the operation speed and is not suitable for time sensitive applications.



```
top - 13:50:00 up 0 min,  1 user,  load average: 1.35, 0.38, 0.13
Tasks:  44 total,   1 running,  43 sleeping,   0 stopped,   0 zombie
Cpu(s): 70.4%us,  4.9%sy,  0.0%ni,  0.0%id, 24.8%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:     61176k total,    59556k used,     1620k free,     4464k buffers
Swap:   102396k total,        0k used,   102396k free,    24616k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1179 root      20   0  362m  22m 5476 S 73.3 38.5   0:20.21 java
  378 root      20   0     0    0    0 S  1.3  0.0   0:00.55 mmcqd
 1192 root      20   0  2532 1092  892 R  1.3  1.8   0:00.14 top
  215 root      20   0     0    0    0 S  0.3  0.0   0:00.03 kswapd0
    1 root      20   0  2020  668  580 S  0.0  1.1   0:00.41 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:00.12 ksoftirqd/0
```

Figure 4.7: Sample while Monitoring running processes

Once the memory was increased there was another attempt to deploy the 16MB aWESoME application. Again, while deploying the application on Tomcat, the top command was executed in order to monitor the running processes. In this procedure twenty samples were taken. The first sample is shown in Figure 4.7, and shows the CPU and Memory allocation by the *java* process after twenty seconds. In addition other parameters were available (e.g. Memory and Swap total size, used and free).

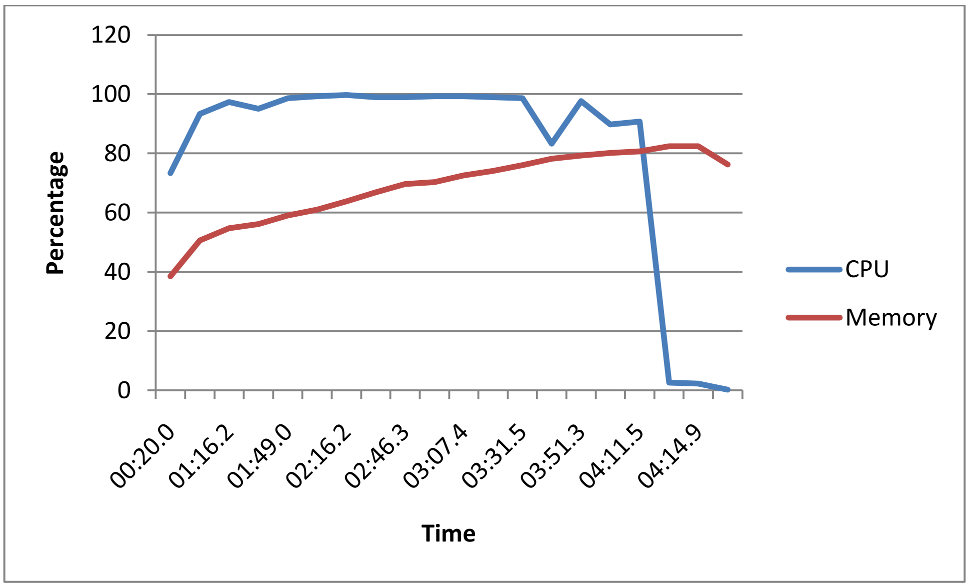
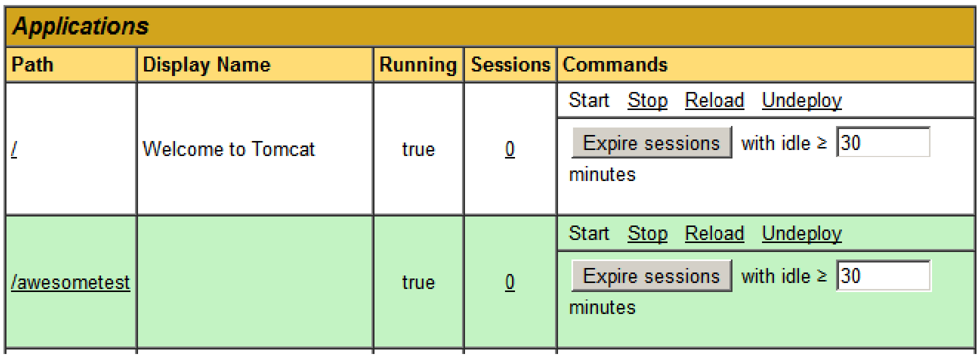


Chart 4.1: CPU and Memory percentage used

The percentage of CPU and Memory used by the *java* process, during the deployment of aWESoME, is shown in Chart 4.1. After approximately four minutes the web application was deployed successfully and was running on Tomcat. This can be confirmed in Tomcat Manager, shown in Figure 4.8.



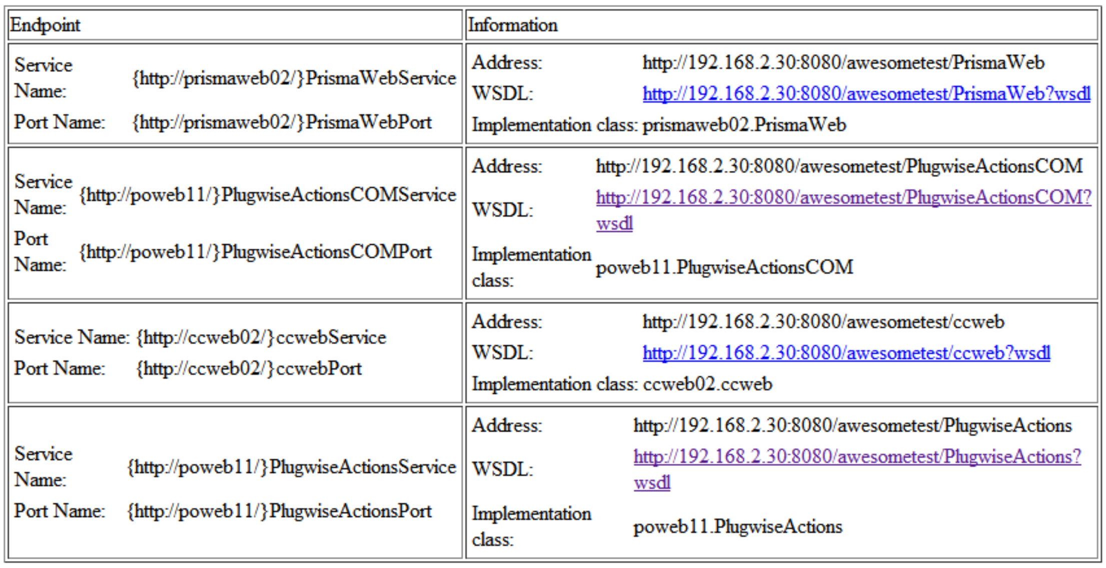
Applications

Path	Display Name	Running	Sessions	Commands
/	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/awesometest		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Figure 4.8: aWESoME deployed

Consuming Web Services

The aWESoME web application provides mainly three web services: PrismaWeb, Plugwise Over the Web and PrismaWeb. Once the web application is deployed the web services are available. To consume the web services a client application must be developed using the WSDL files which describe each web service. The available web services and the WSDL URL's are shown in Figure 4.9.



Endpoint	Information
Service Name: {http://prismaweb02/}PrismaWebService Port Name: {http://prismaweb02/}PrismaWebPort	Address: http://192.168.2.30:8080/awesometest/PrismaWeb WSDL: http://192.168.2.30:8080/awesometest/PrismaWeb?wsdl Implementation class: prismaweb02.PrismaWeb
Service Name: {http://poweb11/}PlugwiseActionsCOMService Port Name: {http://poweb11/}PlugwiseActionsCOMPort	Address: http://192.168.2.30:8080/awesometest/PlugwiseActionsCOM WSDL: http://192.168.2.30:8080/awesometest/PlugwiseActionsCOM?wsdl Implementation class: poweb11.PlugwiseActionsCOM
Service Name: {http://ccweb02/}ccwebService Port Name: {http://ccweb02/}ccwebPort	Address: http://192.168.2.30:8080/awesometest/ccweb WSDL: http://192.168.2.30:8080/awesometest/ccweb?wsdl Implementation class: ccweb02.ccweb
Service Name: {http://poweb11/}PlugwiseActionsService Port Name: {http://poweb11/}PlugwiseActionsPort	Address: http://192.168.2.30:8080/awesometest/PlugwiseActions WSDL: http://192.168.2.30:8080/awesometest/PlugwiseActions?wsdl Implementation class: poweb11.PlugwiseActions

Figure 4.9: Available Web Services and WSDL URL's

In this work, a client was created to consume the Plugwise Over the Web service. This web service is also shown in Figure 4.9, and is named *PlugwiseActionsService*. After defining the WSDL of this service (e.g. http://192.168.2.30:8080/awesometest/PlugwiseActions?wsdl) in the client project, web service references were available. In the *PlugwiseActionsService* eight functions were defined by the developers of aWE-SoME. These functions are shown in Figure 4.10. Each function represents an action and requires one or more arguments. At least one argument, the MAC address of the smart plug, can be used to define a specific plug.

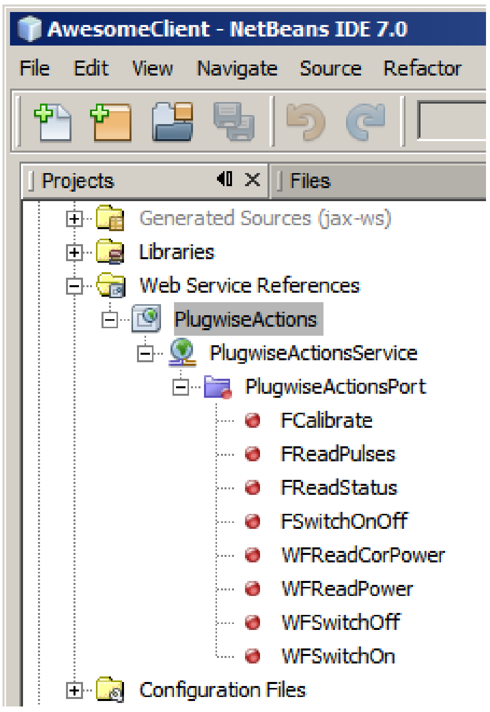


Figure 4.10: Plugwise Available actions

For example the function WFSwitchOn takes two arguments, the MAC address of a plug and another which is the port where Stick is located. In this WS this argument can be blank because the port was previously defined. This function switches on that specific plug (e.g. the command syntax is: wfSwitchOff("000D6F000076D557", "");) and does not return anything. Another example is the function WFReadrPower, which read the pulses of a specific plug and calculates the power measured. In order to use these functions, the developer must insert (e.g. a function can be inserted by dragging and dropping it) the needed function in the main project.

The code of the first client, named Client1, developed is shown below:

```
import java.util.logging.Level;
import java.util.logging.Logger;
import poweb11.IOException_Exception;
import poweb11.InterruptedException_Exception;
import poweb11.UnsupportedCommOperationException_Exception;
import java.text.DateFormat;
import java.util.Calendar;
public class Main {
  public static void main(String[] args) throws InterruptedException_Exception{
    try {
          // Retreive the current Date/time and print it
           Calendar cal = Calendar.getInstance();
           DateFormat df = DateFormat.getDateTimeInstance(DateFormat.FULL,
           DateFormat.MEDIUM);
         // Define a table and and the integer used in the while loop
         long[ ][ ] resptimes= new long[6][21];
         int i;
         i=1;
    //Loop
         while(i<=20) {
        // Define the starting point
        long start = System.currentTimeMillis();
   // Execute 1st command and switch off the 76D557 plug
   wfSwitchOff("000D6F000076D557", "");
        // Calculate the time difference in seconds
        resptimes[0][i]=(System.currentTimeMillis()- start)/1000;
        System.out.println(df.format(cal.getTime())+"1st command: "+resptimes[0][i]+" sec , round: "+i);
    // Execute 2nd command and switch off the 76CDF4 plug
    wfSwitchOff("000D6F000076CDF4", "");
        // Calculate the time difference in seconds
        resptimes[1][i]=(System.currentTimeMillis()- start)/1000;
        System.out.println(df.format(cal.getTime())+"2nd command: "+resptimes[1][i]+" sec , round: "+i);
   // Execute 3rd command and switch off the 76CDF4 plug
    wfSwitchOff("000D6F000043B4CA", "");
        // Calculate the time difference in seconds
        resptimes[2][i]=(System.currentTimeMillis()- start)/1000;
        System.out.println(df.format(cal.getTime())+"3rd command: "+resptimes[2][i]+" sec , round: "+i);
    // Execute 4th command and switch off the 76D557 plug
    wfSwitchOn("000D6F000076D557", "");
        // Calculate the time difference in seconds
        resptimes[3][i]=(System.currentTimeMillis()- start)/1000;
        System.out.println(df.format(cal.getTime())+"4th command: "+resptimes[3][i]+" sec , round: "+i);
   // Execute 5th command and switch off the 76CDF4 plug
    wfSwitchOn("000D6F000076CDF4", "");
        // Calculate the time difference in seconds
        resptimes[4][i]=(System.currentTimeMillis()- start)/1000;
        System.out.println(df.format(cal.getTime())+"5th command: "+resptimes[4][i]+" sec , round: "+i);
   // Execute 6th command and switch off the 76CDF4 plug
    wfSwitchOn("000D6F000043B4CA", "");
        // Calculate the time difference in seconds
        resptimes[5][i]=(System.currentTimeMillis()- start)/1000;
        System.out.println(df.format(cal.getTime())+"6th command: "+resptimes[5][i]+" sec , round: "+i);
i++;
        }
//Exceptions in case something goes wrong
} catch (IOException_Exception ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    } catch (UnsupportedCommOperationException_Exception ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
  }
// The Functions Imported
private static void wfSwitchOff(java.lang.String arg0, java.lang.String arg1) throws IOException_Exception,
UnsupportedCommOperationException_Exception {
      poweb11.PlugwiseActionsService service = new poweb11.PlugwiseActionsService();
      poweb11.PlugwiseActions port = service.getPlugwiseActionsPort();
      port.wfSwitchOff(arg0, arg1);
   }
private static void wfSwitchOn(java.lang.String arg0, java.lang.String arg1) throws UnsupportedCommOpe-
rationException_Exception, IOException_Exception {
      poweb11.PlugwiseActionsService service = new poweb11.PlugwiseActionsService();
      poweb11.PlugwiseActions port = service.getPlugwiseActionsPort();
      port.wfSwitchOn(arg0, arg1);
   }
}
```

The function of the previous code is to execute sequentially six commands. The first three commands switches off three smart plugs and the next three switches on the same plugs. Initially the current time is calculated and after each command the time elapsed is calculated and printed. In order to receive more samples the whole previous procedure is repeated twenty times. In one command a function takes place where the user enters an argument, the MAC address to specify the smart plug. This function is retrieved from the server where the web application and the web service were previously deployed. Detailed descriptions, for each part of the code, are given using comments in grey.

A second client, named Client2, was developed. This client is similar to the previous with a difference in the order the commands are executed. In this client the devices were switched on and off, one by one. Again the response time was calculated.

4.1.2 Experiments Results and Comparisons

The main purpose of this section is to run experiments regarding power dissipation, CPU utilization, memory usage and the response time of each command. The implementation includes the deployment of the aWESoME middleware, which contains the web services, in different servers. Afterwards to consume a web service using a client while monitoring and taking samples regarding the response time and the power dissipation of the server.

The tools used in these experiments are hardware and software. Initially the software used are shown and described in Table 4.1. The hardware used is shown and described in Table 4.2.

Table 4.1: Description of Software used in the experiments

Software	**Info**
Apache Tomcat 6.0	Used as a server for the deployment of web applications.
NetBeans 7.0	Used to develop and run the Client applications.
PwScript	A Java application developed by the Smart IHU team to measure power using a Plugwise smart plug and a Stick.
Client1 and Client2	JAX-WS web service clients, developed in this work assisted by the Smart IHU team, used to consume a web service
aWESoME	A middleware containing JAX-WS developed by the Smart IHU team.

Table 4.2: Description of hardware used in the experiments

Device	**QTY**	**Specification**	**Usage**
Stick	2	USB device used in the Plugwise system.	Collect data from the sensor nodes in a PC.
Smart Plug	4	Plugwise's Smart Plugs.	Measure power and switch on/off devices.
PC1	1	*Processor:* AMD Phenom™ 9600 Quad Core 2.31GHz, *RAM*: 3GB, 2X500hdd, 1X160hdd,*Motherboard*: ASUS M3A32-MVP Deluxe, Nvidia GeForce 8600 GT, *OS:* Windows 7. PSU: 600W max	Measure Power and take samples using a Stick, a Smart Plug and PwScript.
PC2	1	*Model*: Dell Optiplex GX260, *Processor:* Intel P4 1.8Ghz, *RAM:* 512MB, 40GB hdd. *OS:* Linux Ubuntu 10.10 *PSU:* 80W without the PSU losses.	Used as a server while running Apache Tomcat 6.0. or to run client applications
Mini	1	*Model:* Toshiba NB250, *Processor:* Intel Atom ™ N455 2X1.66GHz, RAM: 1GB DDR3, Intel Graphics Media Accelerator 3150, *Display:* 10.1" TFT Toshiba TruBrite, *OS:* Linux Ubuntu 10.10, PSU: 30W without loses.	Used as a server while running Apache Tomcat 6.0. or to run client applications
FoxBoard	1	*Model:* FoxBoard G20, *Processor:* Atmel ARM9 400MHz, RAM: 64MB, *OS:* Linux Debian Lenny, *PSU:* 5W max without losses of the PSU.	Used as a server running Apache Tomcat 6.0.
Lamps	3	IKEA: 230V AC, 20Watts Lamps.	Plugged as loads in the Smart Plugs

Experiment 1

The scope of this experiment was to measure the power dissipation of different machines, which will be used as servers, under different scenarios and mainly while being idle. Three different machines were used, FoxBoard, Mini Note Book and PC2. Fifteen to twenty samples were taken for each scenario and the average Power dissipation for each device in each scenario is shown in Table 4.3.

The scenarios are:

- *Scenario1:* Tomcat was running, the aWESoME (i.e. Web Application, WA) was deployed, Stick was plugged in but the server was Idle (i.e. there are no clients consuming web services).
- *Scenario2:* Same as Scenario 1 but the Stick was Plugged Off.
- *Scenario3:* Stick was plugged off and Tomcat was stopped.
- *Scenario4:* Same as Scenario 3 but the Ethernet cable was disconnected.

Table 4.3: Scenarios and Power Dissipation averages

Scenario	FoxBoard	Mini Note Book	PC2
1	2.1324 W	13.5703 W	48.3361 W
2	2.1325 W	13.3764 W	48.4783 W
3	2.1325 W	13.3764 W	48.3361 W
4	2.1325 W	13.3764 W	47.9096 W

FoxBoard was not affected in these scenarios since the power dissipation did not change. The Mini Note Book again was not affected in these scenarios but the viewing the dataset in each scenario the measured values were fluctuating in an almost periodic manner taking two values, 12.79Watts and 14.92Watts. In regards to PC2 again this computer was fluctuating taking two values, 46.91Watts and 49.04Watts. In all the devices while changing scenarios there was no difference in power dissipation. In addition there were no patterns noticed.

Conclusions

There were two conclusion retrieved in this experiment:

- Some devices (e.g. Stick, Ethernet) and applications (e.g. WA, Tomcat) does not affect and are not correlated to the power dissipation when the server is idle
- Second that FoxBoard needs 15.9% of the power that the Mini Note Book needs; 4.4% of the power that PC2 uses; and the Mini needs 27.7% of the power that PC2 needs. In addition the Mini has a 10.1” display and more computing power and memory and still is more power efficient in comparison to an old Pentium 4 model.

Experiment 2

This experiment was performed to measure the power dissipation and the response time to execute a command in the client software. In general in this application all the devices shown in Table 4.2 and all the software applications were used, except of Client2. There were three setups where in each case the server was running Apache Tomcat 6.0 and aWESoME. In each setup a different device was acting as a server. PC1 in all setups was used to measure and take samples regarding the power dissipation of the server. In addition, in each setup, there were two different wireless sensor and actuator networks: WSAN1 using a Stick plugged in PC1 connected to a smart plug where the server was powered, measuring the power dissipation of the server used in each case; and WSAN2 where a Stick plugged in the Server, the stick was connected to three smart plugs powering three lamps. The server - client connection is granted by a local area network. This could also be a Wide Area Network (WAN). A conceptual connection, of the one used in Experiment 2 is shown in Figure 4.11.

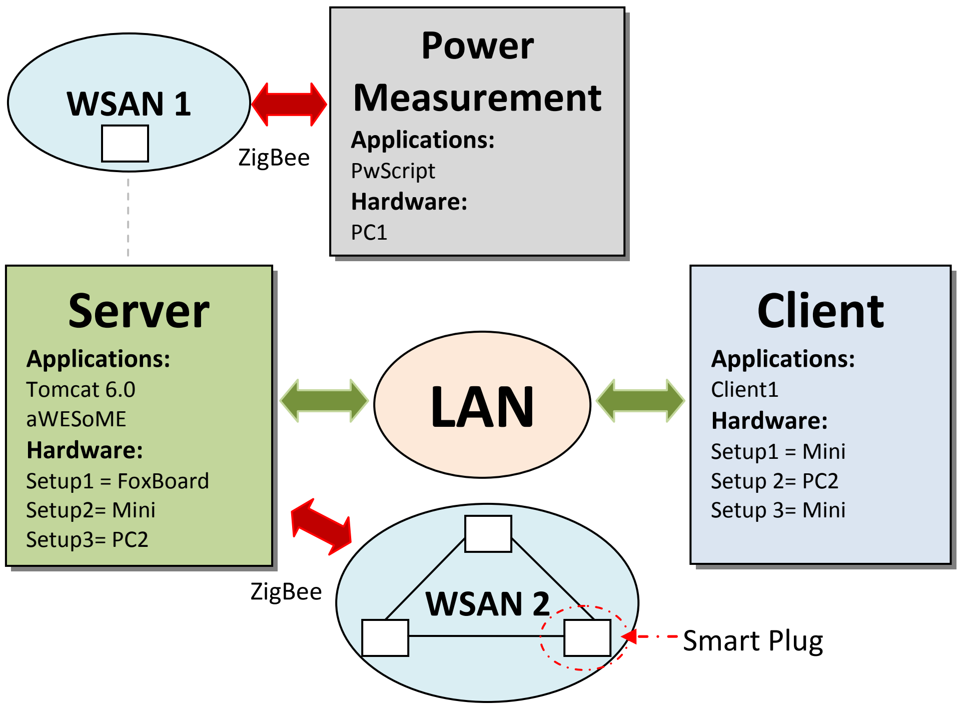


Figure 4.11: Experiment 1: Conceptual connection, devices and applications

In general the Server runs a Web application containing web services and the client, via a LAN or a WAN, is connected to the server. By running a web service client application, the client may consume the available web services. Since the web services are related with WSAN 2, the client has remotely access to WSAN 2 via the Server (e.g.

when the server is connected to the Plugwise System, the client can receive power measurements from each of the three smart plugs or switch on/off each plug, remotely).

In this Experiment the servers were running a Linux Operating System (e.g. Foxboard running Debian Lenny and the other two Linux Ubuntu 10.10). Therefore the middleware (i.e. aWESoME) was customized to be able to run in Linux Environments. After monitoring the servers, it was noticed that the port where Stick was found was */dev/ttyUSB0.* In addition some security features in aWESoME (e.g. registering the hostname and IP address of the client who requested the operation) were disabled due to incompatibility with the Linux OS.

Setup1: FoxBoard as a Server

In this setup the FoxBoard runs aWESoME using Tomcat 6.0 as a server. The client application, named *Client1* ran on the Mini Note Book. The *Client1* application uses the *wfSwithcOff* function, and initially switches off the three devices connected in the smart plugs (e.g. three 20W laps), one by one, and the *wfSwitchOn* function switches on them. The procedure was repeated twenty times. Meanwhile the power dissipation and the response time of the server were monitored. In addition some samples of the CPU utilization and memory allocation were taken. The samples of the power dissipation were taken before the client started consuming the WS and during the WS consumption. The CPU utilization and memory samples were taken during the WS consumption and also the response time after each command (e.g. switch on/off).

One repetition executed six commands. The loop ran 20 times and there were 120 commands executed collecting 120 samples of the response time. The average total time to execute the client application was *5 minutes and 58 seconds (358s)*. The client ran two times. The samples are show in Table 4.4.

Table 4.4: Response time samples of FoxBoard

Value (Seconds)	**1st Execution**		**2nd Execution**		**Total**	
	# Samples	**%**	**# Samples**	**%**	**# Samples**	**%**
2	4	3.333	9	7.5	13	5.417
3	110	91.667	108	90	218	90.833
4	6	5	2	1.667	8	3.333
5	0	0	1	0.833	1	0.417
Total	120	100	120	100	240	100

The distribution of the response time is graphically presented in Chart 4.2, where also as shown in Table 4.4 in the 90.8% of the cases the response time is three seconds, respectively in 5.4% of the cases two second, in the 3.3% is four and in 0.4% is five. Te values were not integer multiples of a second, and were divided in bins to be easily represented.

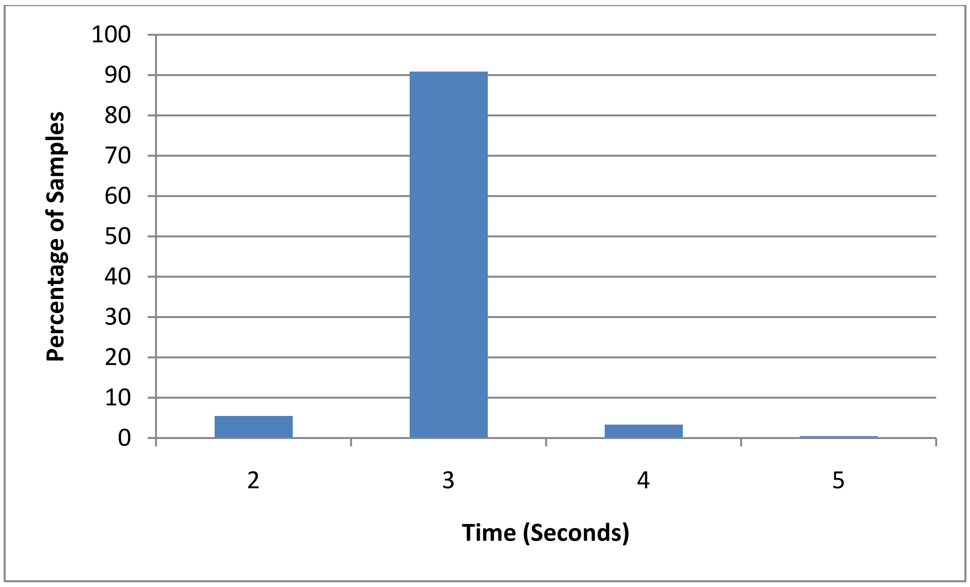


Chart 4.2: Response time Distribution of FoxBoard

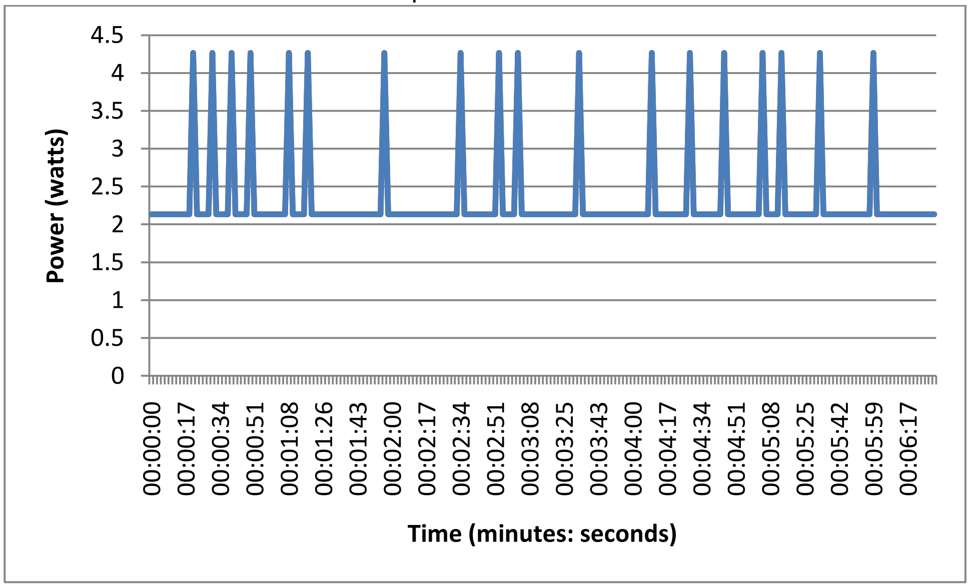


Chart 4.3: Power dissipation of FoxBoard while active

The power dissipation of the server during the time period while the PlugwiseActions web service was consumed is shown in Chart 4.3.

In addition few samples were taken, in the same time period, regarding the CPU utilization and memory usage by the Java process. The results are illustrated in Chart 4.4.

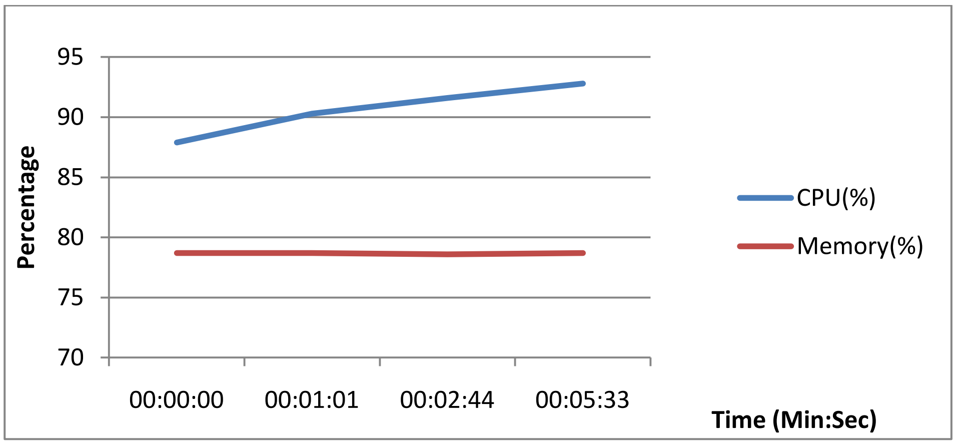


Chart 4.4: Memory Usage (%) and CPU(%) utilization in FoxBoard

Setup 2: Mini Note Book as a Server

The differences of this setup with the previous are that the fact that the Mini Note Book is the server and PC2 is the client. After running the Client1 application, results were retrieved regarding the power dissipation, response time, CPU utilization and memory usage of the Mini Note Book. Table 4.5 indicates the distribution of the response time samples. The total time to execute 120 commands was *2 minutes and 22 seconds (142 seconds)*. A graphic presentation of the distribution is shown in Chart 4.5. In regards to the power dissipation the results are shown in Chart 4.6.

Table 4.5: Response Time Samples of the Mini Note Book

Value (Seconds)	**1st Execution**		**2nd Execution**		**Total**	
	# Samples	**%**	**# Samples**	**%**	**# Samples**	**%**
1	99	82.5	98	81.667	197	82.1
2	21	17.5	22	18.333	43	17.9
Total	120	100	120	100	240	100

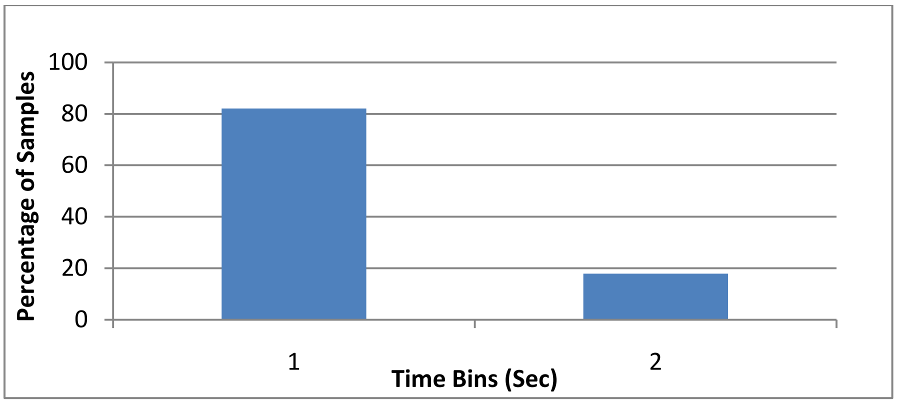


Chart 4.5: Response time Distribution of the Mini Note Book

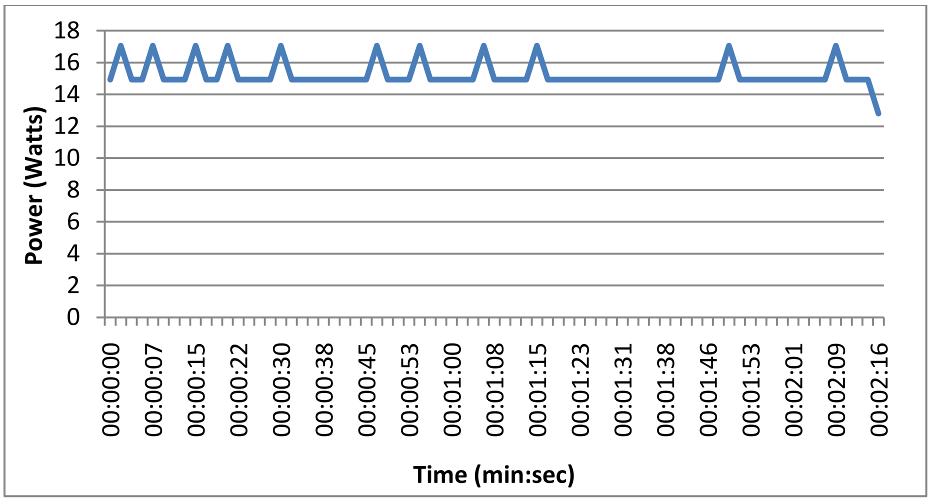


Chart 4.6: Power dissipation of Mini while active (W)

Last the memory usage and CPU utilization by the java process, during this time period is shown in Chart 4.7. There was a value that exceeds 100% and instantly reaches *115%,* shown in the top processes of Ubuntu. This happens because the processor is a dual core and in that time instance it was utilizing both processor's cores.

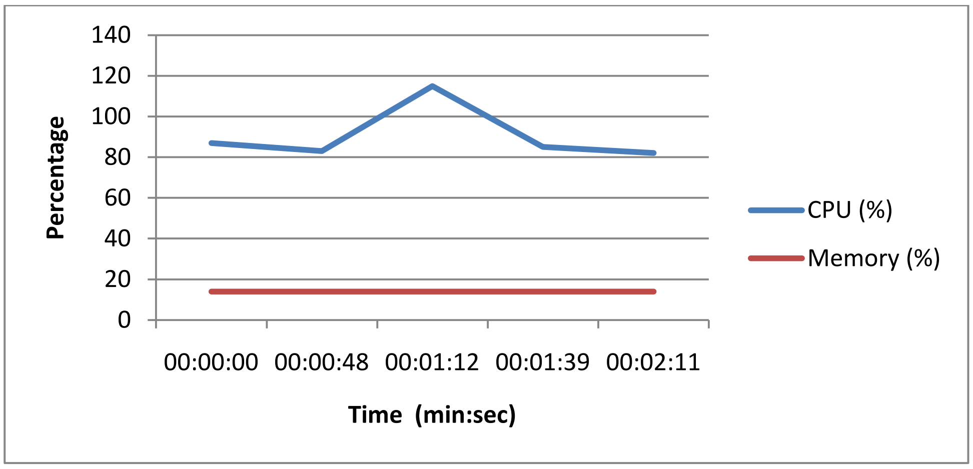


Chart 4.7: Memory Usage (%) and CPU(%) utilization in Mini

Setup3: Mini Note Book as a Server, display excluded

In this setup the PC2 was used as a Server and the Mini Note Book was the client the rest of the equipment remained as was. Again the same parameters were monitored. The distribution results of the samples regarding the response time of the server are shown in Table 4.6

Table 4.6 Response Time Samples of the Mini Note Book

Value (Seconds)	**1st Execution**		**2nd Execution**		**Total**	
	# Samples	**%**	**# Samples**	**%**	**# Samples**	**%**
1	100	83.333	98	81.667	198	82.5
2	20	16.667	22	18.333	42	17.5
Total	120	100	120	100	240	100

A graphic representation is provided in Chart 4.8. The total time to execute 120 commands was approximately *2 minutes and 22 Seconds* (average *142 seconds*). Next the power dissipation is shown in Chart 4.9. It was notices that the power exceeds the maximum output power given by the manufacturer (i.e. 80W). The values above 80W are the losses of the PSU. The maximum values reach almost 10Watts (i.e. 12.5%).

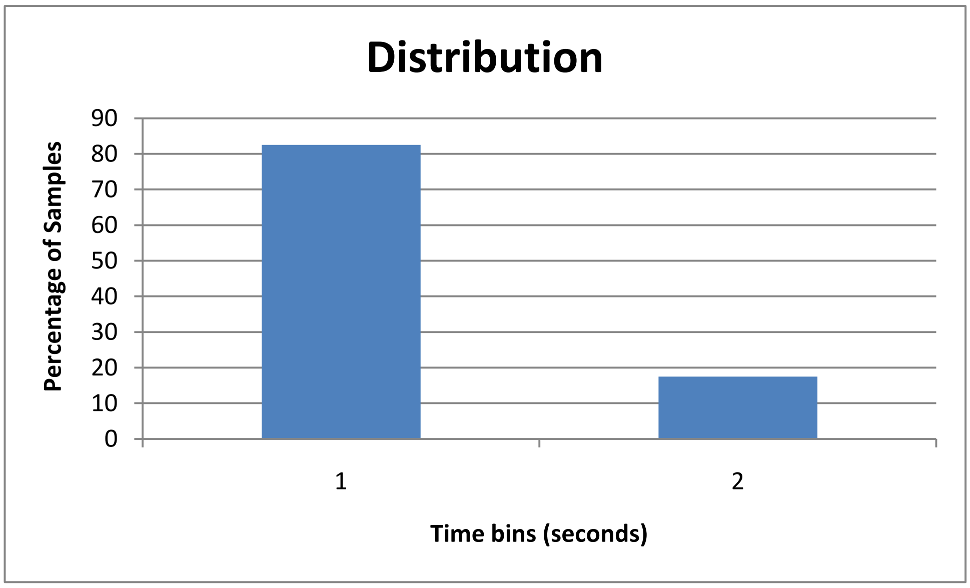


Chart 4.8: Response time Distribution of the PC2

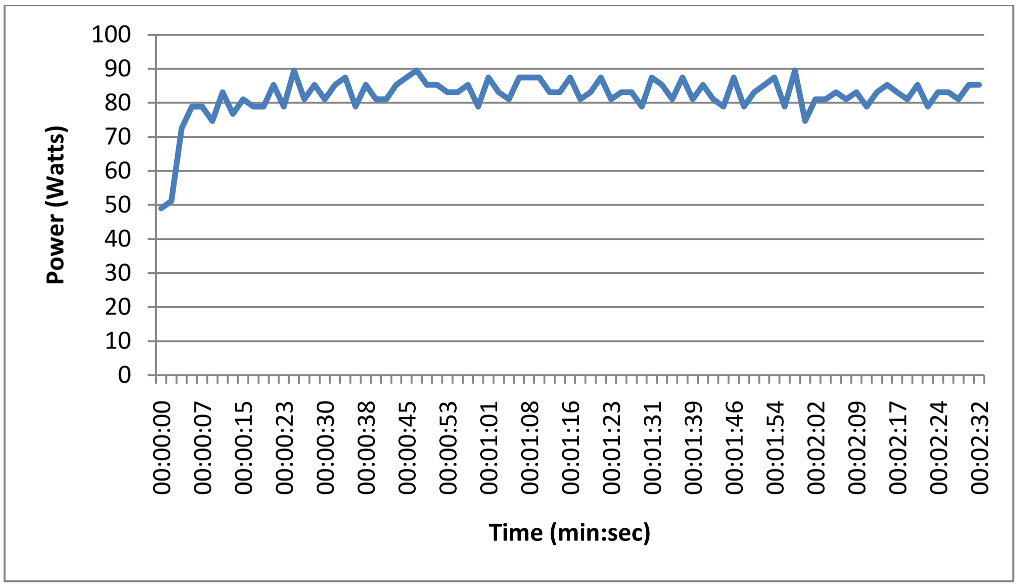


Chart 4.9: Power dissipation of Mini while active (W)

Finally, the CPU utilization and memory usage are shown in Chart 4.10.

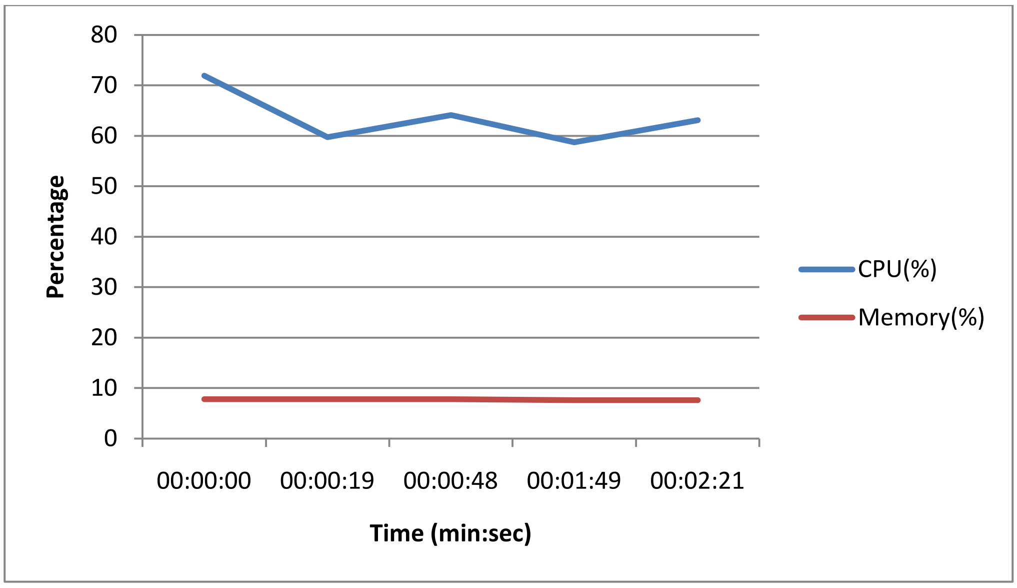


Chart 4.10: Memory Usage (%) and CPU (%) utilization in PC2

Comparisons

The possible comparisons involve the response time and the power dissipation. First the power dissipation in multiple samples in each setup was compared graphically in Chart 4.11.

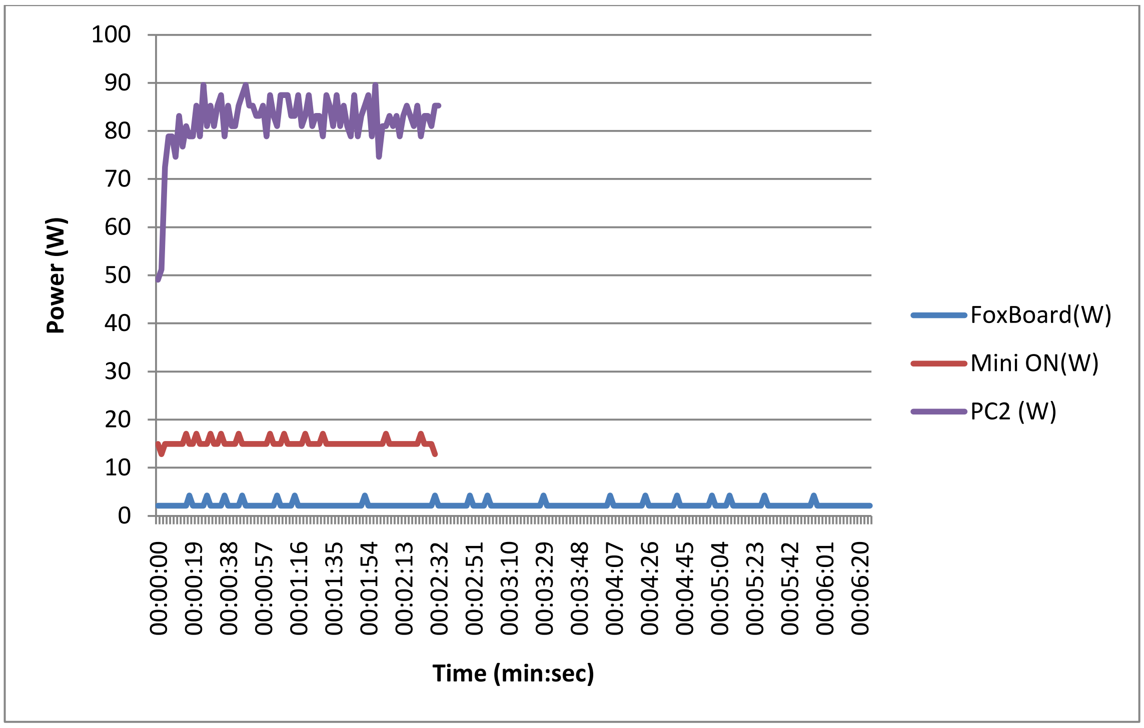


Chart 4.11: Server power dissipation comparison

Another comparison is shown in Table 4.7, related to the average power dissipation when active and idle.

Table 4.7: Average power dissipation when idle and active

FoxBoard			Mini Note Book			PC2		
AVG Power Idle (W)	**AVG Power Active (W)**	**Difference (W)**	**AVG Power Idle (W) ON**	**AVG Power Active (W) ON**	**Difference (W)**	**AVG Power Idle (W)**	**AVG Power Active (W)**	**Difference (W)**
2.133	2.337	0.204	13.376	15.220	1.844	48.265	81.998	33.742

In regards to the response time the total percentage are shown in Table 4.8 an illustrated graphically in Chart 4.12.

Table 4.8: Response time overall distribution

Value (Seconds)	**FoxBoard (Percentage)**	**Mini (%)**	**PC2 (%)**
1		82.1	82.5
2	5.417	17.9	17.5
3	90.833		
4	3.333		
5	0.417		
Total	100	100	100

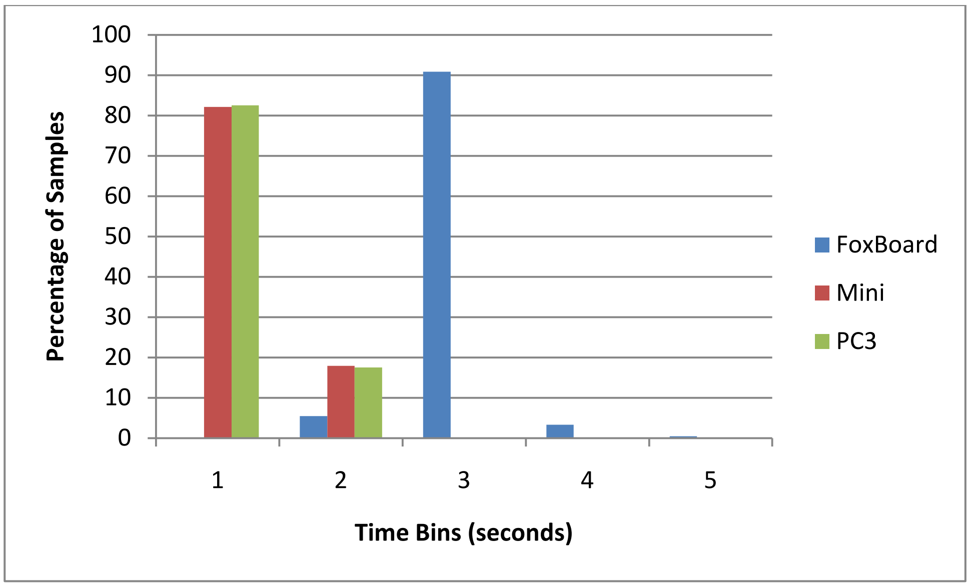


Chart 4.12: Overall Distribution.

The average time periods needed to execute the 120 commands is shown in Table 4.9.

Table 4.9: Average time elapsed to execute Client1

FoxBoard (Sec)	Mini (Sec)	PC2 (Sec)
358	142	142

Conclusions

As shown in Chart 4.11, there is a significant difference between the three devices. Since the three devices are running exactly the same application, PC2 is the most power inefficient. In addition, in PC2, the power dissipation rose by almost 70% from idle to active, in contrast to FoxBoard and Mini Note Book where the increase was 9.59% and 13.79% respectively. Comparisons could be made, between the two devices remaining (FoxBoard and Mini) regarding the resources they provide.

In all three settings a pattern was noticed and is shown in Chart 4.3, Chart 4.6 and Chart 4.9. It is noticed that when the server is active there is an increase in power dissipation. There is the need to investigate the factors responsible for this increase.

The distribution of the response time of Mini and PC3 are similar, reverse exponential and in FoxBoard's is a normal distribution. As shown in Table 4.9 the average time to execute Client1 is the same in Mini and PC2 in contrast to FoxBoard where it is 2.5 times more. Both Mini and PC2 are faster and the probabilities for the response time are closely the same.

PC3 can be excluded, because is the most power inefficient. In a system where there are many gateways operating 24/7 the energy consumption will reach significant levels. The selection must be done between FoxBoard and Mini. A quantitative comparison is provided in Table 4.10.

Table 4.10: Comparison between Foxboard and Mini Notebook

Parameter	**Mini Notebook**	**FoxBoard**	**Difference**
Processor's Clock	2X1.6GHz	400MHz	8 times more
RAM	1GB	64MB	16 times more
AVG Power Dissipation when active	15.220	2.337	6 times higher
Total Response Time	142	358	2.5 times faster

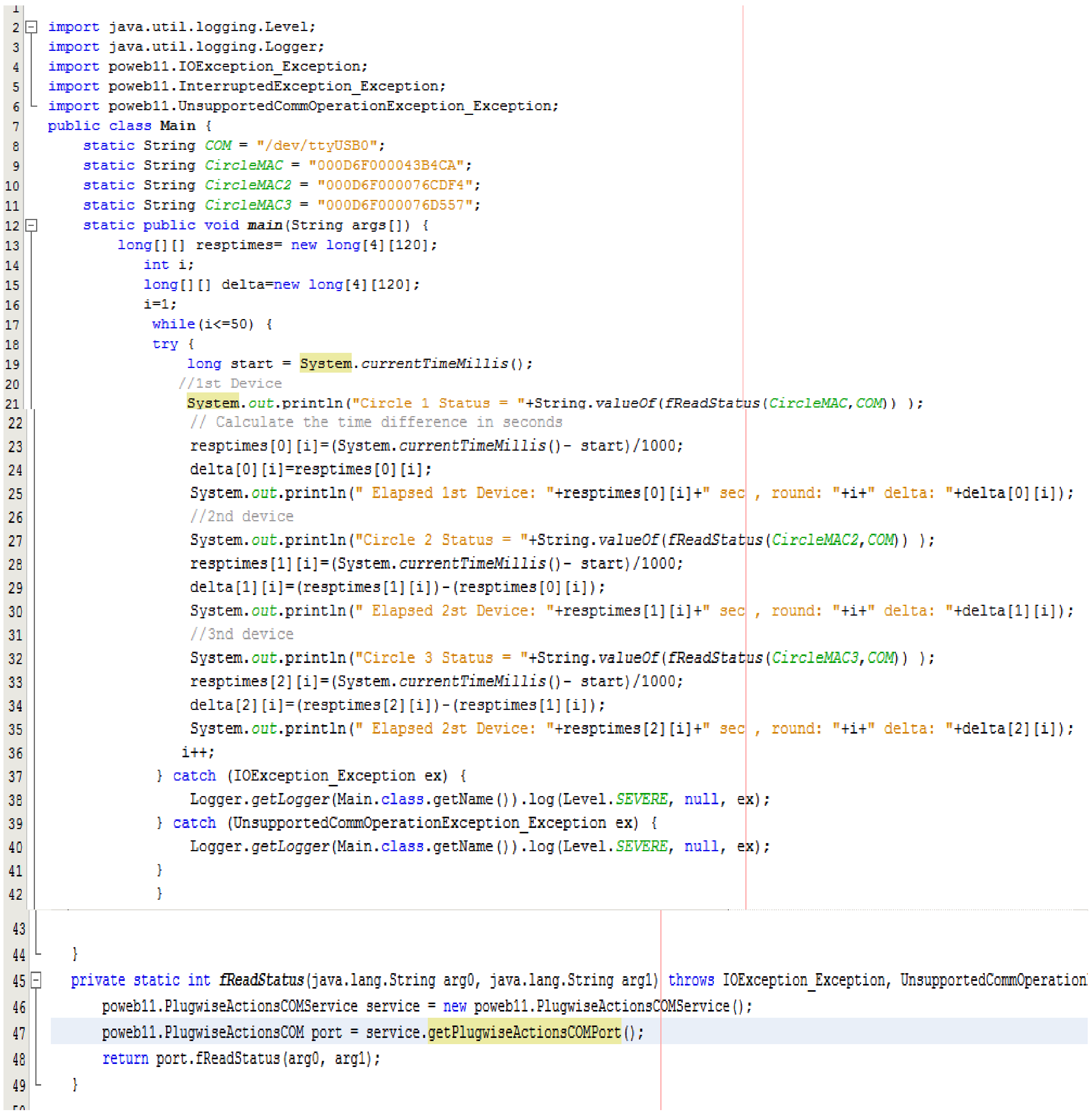
To conclude FoxBoard is really energy efficient but it has reached its limits (e.g. the ram was not enough and was extended using a swap file) and the hardware is not flexible (e.g. the ram is difficult to be extended). Though, its CPU Clock and RAM are 8 and 16 times less than Mini, the average response time while executing a command in 90% percent of the cases is 3 seconds. FoxBoard could be suggested in applications with a low number of users and les time sensitive applications. A possible intermediate system between FoxBoard and Mini would be one of the Pico-ITX devices.

Experiment 3

In the previous experiment it was noticed that the power dissipation is correlated with the activity of the server. The response time was measured in regards of two operations, *wfSwitchOff* and *wfSwitchOn*. In this experiment the response time of the other operations was investigated and measured while FoxBoard and Toshiba NB250 were acting as servers in each case. The main remaining operations are: *fReadStatus* which reads the status of a specific Circle and *wfReadPower* reading the power of a Circle.

fReadStatus

To utilize the *fReadStatus* operation a client was created based on the PlugwiseActionsCOM?wsdl (e.g. http://192.168.2.30:8080/awesometest/PlugwiseActionsCOM?wsdl). The client was named ClientReadStatus and the source code is shown in Figure 4.12. This client reads the status of three Circle Smart Plugs sequentially. The MAC addresses of these plugs are: 43B4CA; 76CDF4; and 76D557. The returned value of *fReadStatus* can be: 0 when the device is OFF; 1 when the device is ON; and -1 when an error occurs. Again the version of aWESoME used was a customized version named *awesometest*, disabling some security features and defining the appropriate port for Linux (e.g. /dev/ttyUSB0), once the server was running on a Linux based OS.



```java

import java.util.logging.Level;
import java.util.logging.Logger;
import poweb11.IOException_Exception;
import poweb11.InterruptedException_Exception;
import poweb11.UnsupportedCommOperationException_Exception;
public class Main {
    static String COM = "/dev/ttyUSB0";
    static String CircleMAC = "000D6F000043B4CA";
    static String CircleMAC2 = "000D6F000076CDF4";
    static String CircleMAC3 = "000D6F000076D557";
    static public void main(String args[]) {
        long[][] resptimes= new long[4][120];
           int i;
           long[][] delta=new long[4][120];
           i=1;
            while(i<=50) {
            try {
                long start = System.currentTimeMillis();
               //1st Device
               System.out.println("Circle 1 Status = "+String.valueOf(fReadStatus(CircleMAC,COM)) );
                // Calculate the time difference in seconds
                resptimes[0][i]=(System.currentTimeMillis()- start)/1000;
                delta[0][i]=resptimes[0][i];
                System.out.println(" Elapsed 1st Device: "+resptimes[0][i]+" sec , round: "+i+" delta: "+delta[0][i]);
                //2nd device
                System.out.println("Circle 2 Status = "+String.valueOf(fReadStatus(CircleMAC2,COM)) );
                resptimes[1][i]=(System.currentTimeMillis()- start)/1000;
                delta[1][i]=(resptimes[1][i])-(resptimes[0][i]);
                System.out.println(" Elapsed 2st Device: "+resptimes[1][i]+" sec , round: "+i+" delta: "+delta[1][i]);
                //3nd device
                System.out.println("Circle 3 Status = "+String.valueOf(fReadStatus(CircleMAC3,COM)) );
                resptimes[2][i]=(System.currentTimeMillis()- start)/1000;
                delta[2][i]=(resptimes[2][i])-(resptimes[1][i]);
                System.out.println(" Elapsed 2st Device: "+resptimes[2][i]+" sec , round: "+i+" delta: "+delta[2][i]);
               i++;
            } catch (IOException_Exception ex) {
                Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
            } catch (UnsupportedCommOperationException_Exception ex) {
                Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
            }
            }


    }
    private static int fReadStatus(java.lang.String arg0, java.lang.String arg1) throws IOException_Exception, UnsupportedCommOperation
        poweb11.PlugwiseActionsCOMService service = new poweb11.PlugwiseActionsCOMService();
        poweb11.PlugwiseActionsCOM port = service.getPlugwiseActionsCOMPort();
        return port.fReadStatus(arg0, arg1);
    }
```

Figure 4.12: ClientReadStatus client Software

The resulted response time distribution, in seconds, for the two devices, in 150 samples, is illustrated in Chart 4.13. The total time to complete for FoxBoard was *8 minutes and 46 seconds* and for Toshiba NB250 was *one minute and 44 seconds*. The actual values of the response time were divided in seven bins (e.g. 0.6, 1, 2...7 seconds).

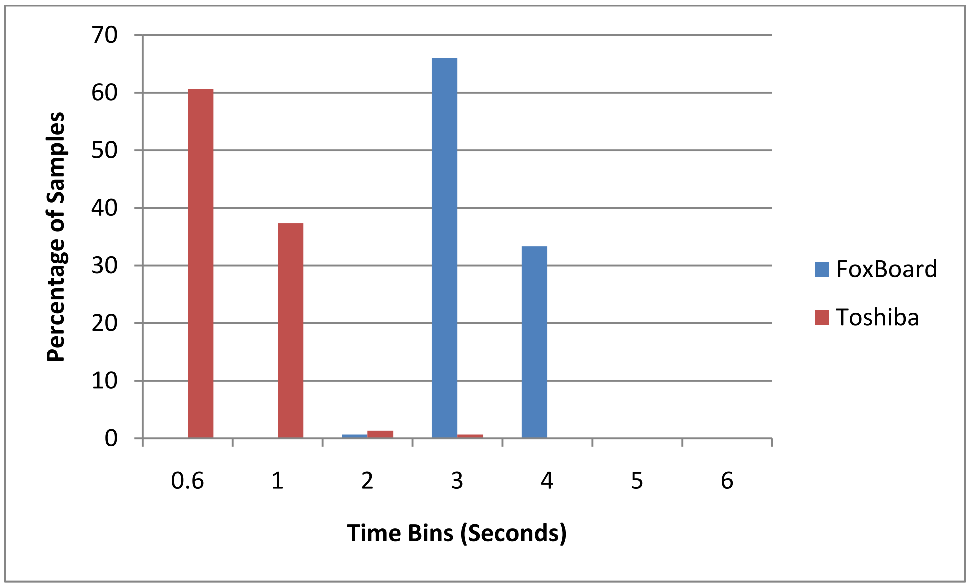
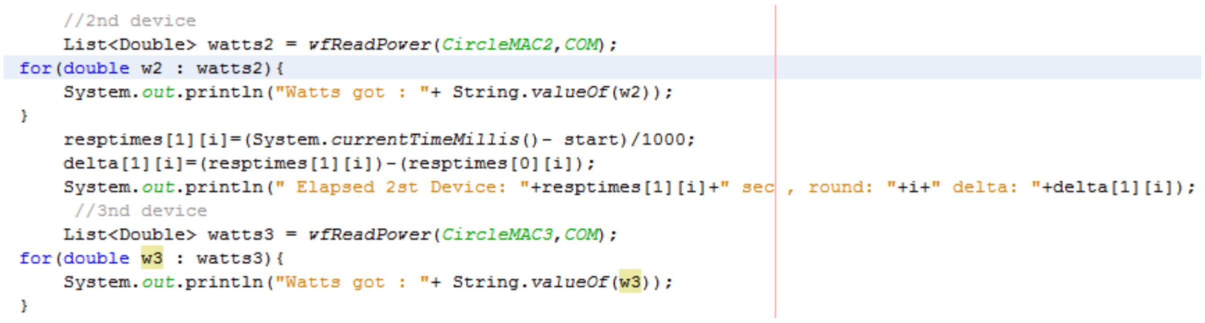


Chart 4.13: ReadStatus Response Time Distributions in seconds

wfReadPower

wfReadPower was tested creating a client, named ClientReadPower, similar to the ClientReadStatus client. The main difference is in Main, where for each device the following code were added replacing the fReadStatus operation.



```
    //2nd device
    List<Double> watts2 = wfReadPower(CircleMAC2,COM);
for(double w2 : watts2){
    System.out.println("Watts got : "+ String.valueOf(w2));
}
    resptimes[1][i]=(System.currentTimeMillis()- start)/1000;
    delta[1][i]=(resptimes[1][i])-(resptimes[0][i]);
    System.out.println(" Elapsed 2st Device: "+resptimes[1][i]+" sec , round: "+i+" delta: "+delta[1][i]);
     //3nd device
    List<Double> watts3 = wfReadPower(CircleMAC3,COM);
for(double w3 : watts3){
    System.out.println("Watts got : "+ String.valueOf(w3));
}
```

Additionally the appropriate operation was added.

```
private static java.util.List<java.lang.Double> wfReadPower(java.lang.String arg0, java.lang.String arg1) throws IOE
    poweb11.PlugwiseActionsCOMService service = new poweb11.PlugwiseActionsCOMService();
    poweb11.PlugwiseActionsCOM port = service.getPlugwiseActionsCOMPort();
    return port.wfReadPower(arg0, arg1);
}
```

The results in both cases are shown in Chart 4.14. Again there were three Circle smart plugs and the power was retrieved 50 times from each, there were 150 samples available.

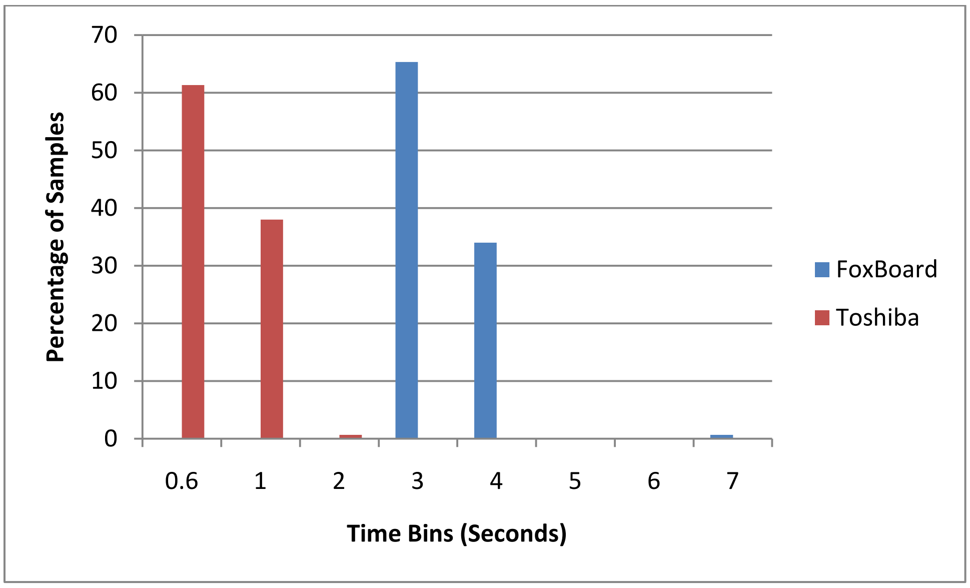


Chart 4.14: ReadPower Response time distributions in seconds

The total time for FoxBoard to serve these requests was *8 minutes and 53 seconds* and for Toshiba NB 250 was *one minute and 40 seconds*.

Conclusions

Both in ReadPower and ReadStatus, it was noticed that FoxBoard’s response time is significantly higher which varies from three to four seconds in contrast to 0.6 and one second in Toshiba NB250. In regards to the total time in each of the previous clients, it was noticed that FoxBoard was approximately five times slower than the Toshiba NB250.

4.2 Web Services

As mentioned in the beginning of this chapter the need of remote control and monitoring system, in environments with distributed computers, is obvious. I this section the following techniques will be investigated: Wake On LAN (WOL), LAN Shutdown and system parameter monitoring. Afterwards java applications were developed and each technique was deployed as a function in a web service in a web application. Last the entire web application will be tested using NetBeans but without creating a client application.

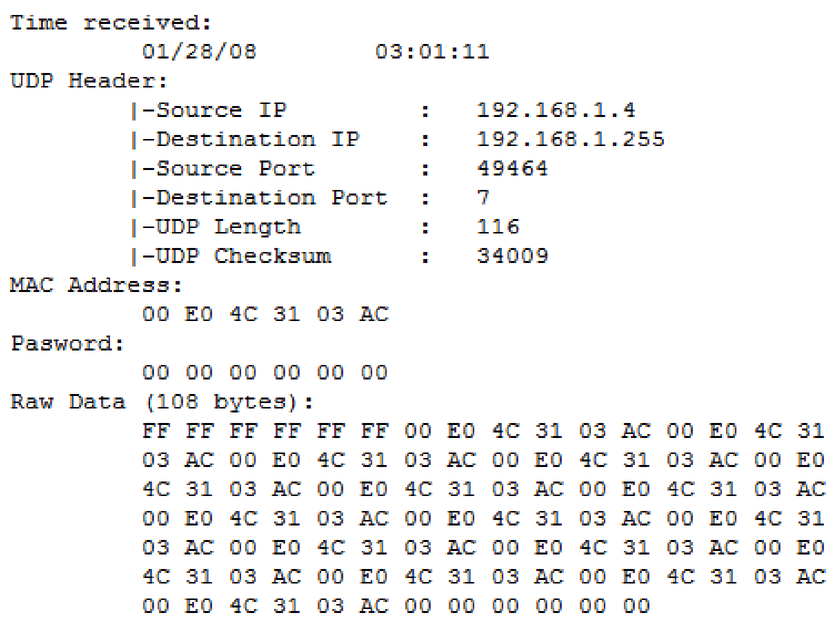
4.2.1 Wake On LAN (WOL)

As a standard supported by many OS and network interface manufacturers, the wake on LAN technology is generally used in order to remotely "wake up" computers, while being in sleep/hibernate/shutdown. This can be easily implemented inside a LAN, by creating an application which generates magic packets with the parameters of the computer needed to wake up. WOL packet cannot be routed but WOL can also be implemented on a WAN (Wake ON WAN or WOW), by the use of Subnet Directed Broadcasts (SDB).

The magic packet is a UDP datagram which is sent on the link layer of the OSI model and it is broadcasted to all the interfaces of the network using the broadcast address, containing the MAC address of the destination computer. Once WOL is enabled in the listing computer, it waits for a magic packet containing its address and wakes the system, when receives it.

A Magic Packet is a UDP broadcast message containing the MAC address of the destination computer. This datagram contains a six bytes of synchronization stream of FFs (i.e. Hex: FF FF FF FF FF FF) and 16 times the repetition of the MAC address. An actual example is shown in Figure 4.13 provided by profshutdown.com[28]

[28] http://www.profshutdown.com/wakeonlan_troubleshoot.aspx



```
Time received:
         01/28/08        03:01:11
UDP Header:
        |-Source IP          :   192.168.1.4
        |-Destination IP     :   192.168.1.255
        |-Source Port        :   49464
        |-Destination Port   :   7
        |-UDP Length         :   116
        |-UDP Checksum       :   34009
MAC Address:
         00 E0 4C 31 03 AC
Pasword:
         00 00 00 00 00 00
Raw Data (108 bytes):
         FF FF FF FF FF FF 00 E0 4C 31 03 AC 00 E0 4C 31
         03 AC 00 E0 4C 31 03 AC 00 E0 4C 31 03 AC 00 E0
         4C 31 03 AC 00 E0 4C 31 03 AC 00 E0 4C 31 03 AC
         00 E0 4C 31 03 AC 00 E0 4C 31 03 AC 00 E0 4C 31
         03 AC 00 E0 4C 31 03 AC 00 E0 4C 31 03 AC 00 E0
         4C 31 03 AC 00 E0 4C 31 03 AC 00 E0 4C 31 03 AC
         00 E0 4C 31 03 AC 00 00 00 00 00 00
```

Figure 4.13: Magic Packet Example

When WOL is enabled in a NIC it requires some parts of the hardware to stay this results into increasing slightly the energy consumption when is sleep/hibernate/power off[29] . Therefore disabling WOL may conclude in reducing, slightly, the total energy consumption. Additionally WOL also depend on hardware, since must be supported by the NIC's hardware. Since the magic packet uses UDP, there is a probability that the sent can be lost. Therefore supplement operations (e.g. the use of *ping*) can be utilized to check if the destination PC "woke up".

Implementation

To implement Wake On LAN in a computer using the magic packet, initially the user must enable WOL in the NIC's settings (Windows: *LocalAreaConnection>Properties>Configure> Advanced > Property: Wake-up Capabilities Value: Magic Packet)* in this case the computer will wake up only when the system is in sleep or hibernate mode. In case it is needed to wake up the pc when in Shutdown mode then the Remote wake up setting in the BIOS must be enabled.

A java application was created, in cooperation with the Smart IHU team, by using parts of code, of the open source GUI Java WOL Project[30] available in sourceforge.net.

[29] In shutdown mode it is supposed that the plug remains pugged-in and the PSU is powered.
[30] http://guijavawol.sourceforge.net/

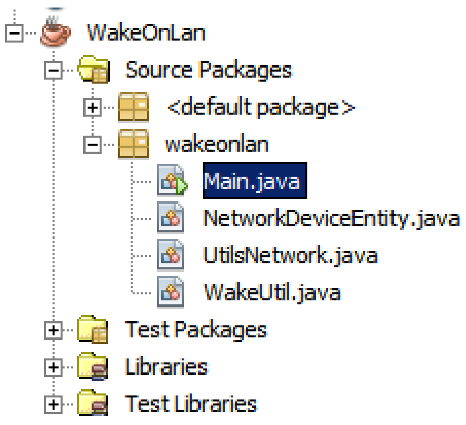
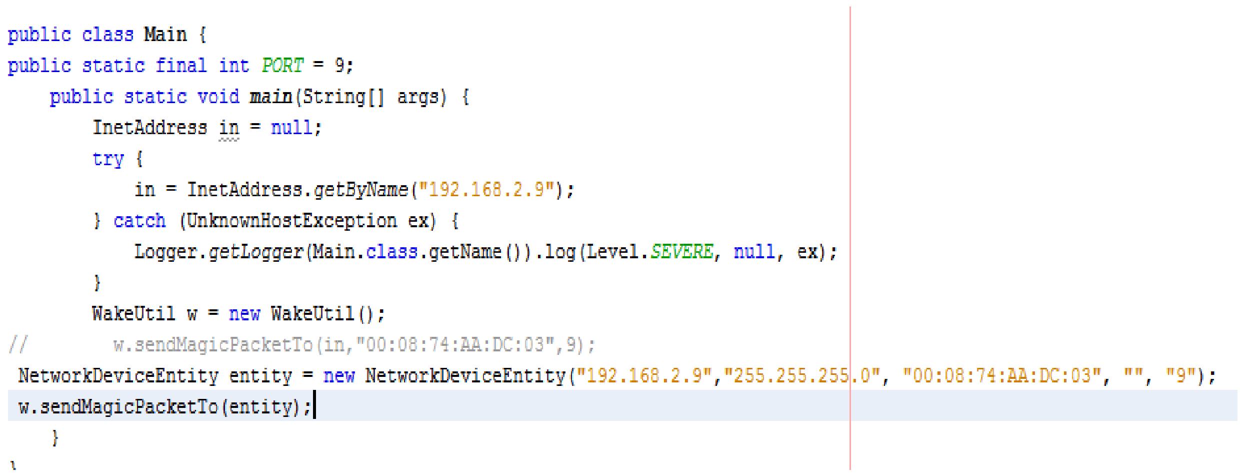


Figure 4.14: Wake On LAN java application software.

As shown in Figure 4.14 the java files *NetworkDeviceEntity.java*, *UtilisNetwork.java* and *WakeUtil.java* were taken by the GUI Java WOL Project. Afterwards the Main.java code was developed by the Smart IHU team. The source code of *Main.java* is shown in Figure 4.15.



```
public class Main {
public static final int PORT = 9;
    public static void main(String[] args) {
        InetAddress in = null;
        try {
            in = InetAddress.getByName("192.168.2.9");
        } catch (UnknownHostException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
        WakeUtil w = new WakeUtil();
//          w.sendMagicPacketTo(in,"00:08:74:AA:DC:03",9);
 NetworkDeviceEntity entity = new NetworkDeviceEntity("192.168.2.9","255.255.255.0", "00:08:74:AA:DC:03", "", "9");
 w.sendMagicPacketTo(entity);
    }
}
```

Figure 4.15: Wake On LAN java application Main.java

In this package the main classes imported by other packages are: java.net.InetAddress, java.net.UnknownHostException, java.util.logging.Level and java.util.logging.Logger. This code actually sends a Magic packet in port 9 to the computer with the following parameters; NIC's MAC address: 00:08:74:AA:DC:03; IP address: 192.168.2.9; and Network Mask: 255.255.255.0.

Monitoring

The following experiment was conducted using: the applications, wakeonlan java project, NetBeans and Wireshark; and hardware PC1 and PC2, shown in Table 4.2, interconnected via a LAN. The settings and parameters of both computers are shown in Table 4.11.

Table 4.11: Parameters and Settings while using wake on LAN

Hostname	**IP Address**	**NIC's MAC address**	**Network Address**	**Role**
PC1= Andrew-PC	192.168.2.13	00:1E:8C:02:BB:E3	255.255.255.0	Source
PC2= DellComp_aa	192.168.2.9	00:08:74:AA:DC:03	255.255.255.0	Destination

The NIC of PC1 was monitored using Wireshark and afterwards the wake on LAN java application was executed in PC1, with the parameters and settings of PC2. The magic packet captured by wire shark is shown in blue in Figure 4.16. It was noticed that the source IP was 192.168.2.13 (i.e. the IP address of PC1) and the destination IP address was 192.168.2.255 (i.e. the broadcast address of the subnet).

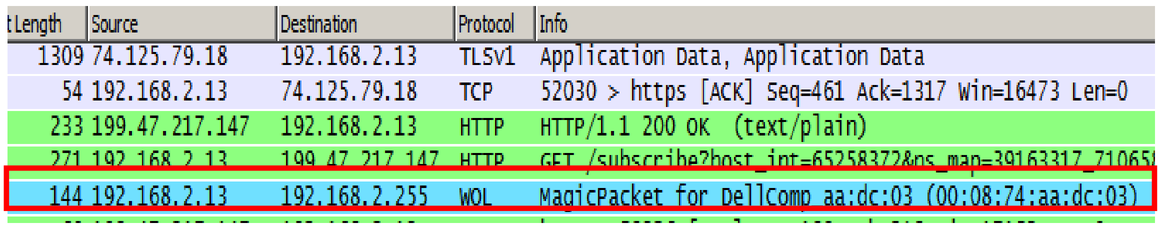

```
tLength Source          Destination     Protocol Info
   1309 74.125.79.18    192.168.2.13    TLSv1    Application Data, Application Data
     54 192.168.2.13    74.125.79.18    TCP      52030 > https [ACK] Seq=461 Ack=1317 Win=16473 Len=0
    233 199.47.217.147  192.168.2.13    HTTP     HTTP/1.1 200 OK  (text/plain)
    271 192.168.2.13    199.47.217.147  HTTP     GET /subscribe?host_int=65258372&ns_map=39163317_71065
    144 192.168.2.13    192.168.2.255   WOL      MagicPacket for DellComp_aa:dc:03 (00:08:74:aa:dc:03)
```

Figure 4.16: Magic Packet Captured by Wireshark

The exact bytes contained in this magic packet is shown in Figure 4.17, and some details are given in Figure 4.18

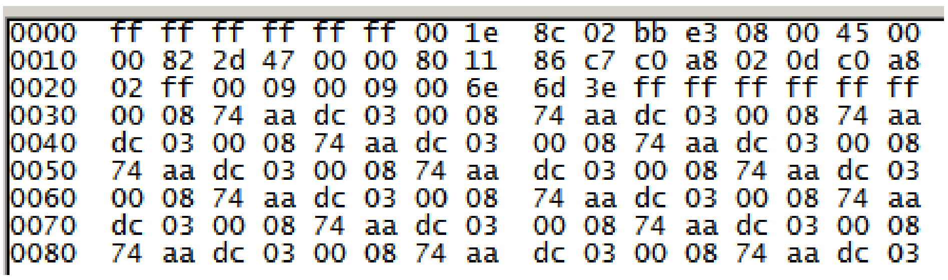
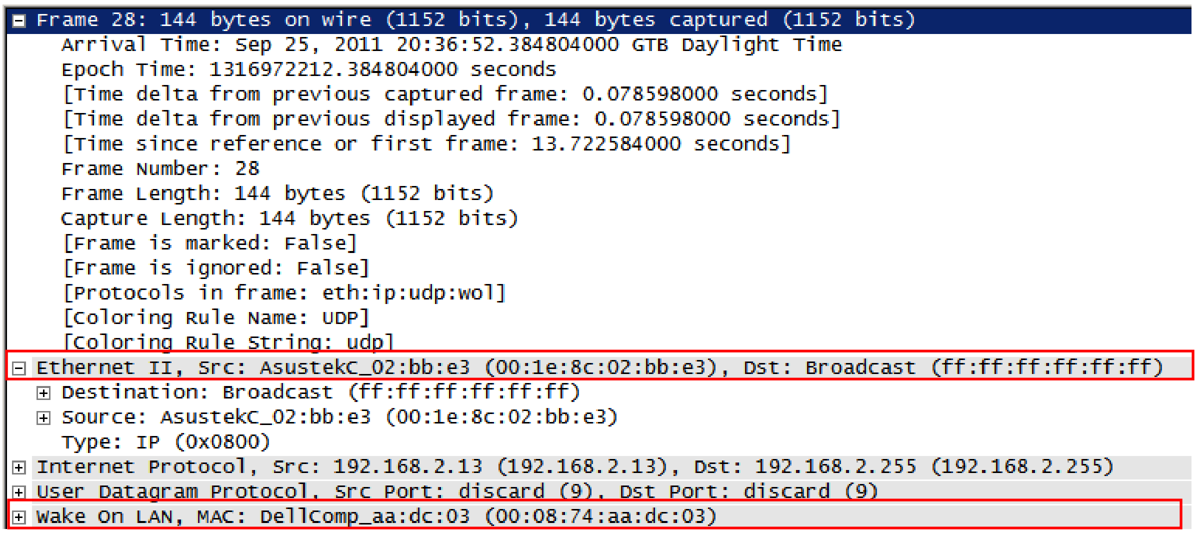

```
0000  ff ff ff ff ff ff 00 1e  8c 02 bb e3 08 00 45 00
0010  00 82 2d 47 00 00 80 11  86 c7 c0 a8 02 0d c0 a8
0020  02 ff 00 09 00 09 00 6e  6d 3e ff ff ff ff ff ff
0030  00 08 74 aa dc 03 00 08  74 aa dc 03 00 08 74 aa
0040  dc 03 00 08 74 aa dc 03  00 08 74 aa dc 03 00 08
0050  74 aa dc 03 00 08 74 aa  dc 03 00 08 74 aa dc 03
0060  00 08 74 aa dc 03 00 08  74 aa dc 03 00 08 74 aa
0070  dc 03 00 08 74 aa dc 03  00 08 74 aa dc 03 00 08
0080  74 aa dc 03 00 08 74 aa  dc 03 00 08 74 aa dc 03
```

Figure 4.17: Bytes found in Magic Packet

Again in Figure 4.18 it can be noticed the destination broadcast MAC and the WOL MAC. After performing several tests of this application it was noticed experimentally that, if WOL was enabled only in the NIC's parameters in the OS, the listening PC woke up only when it was in sleep/hibernate mode and not when it was in shutdown mode. When WOL was enabled in the NIC's BIOS settings the listening PC woke even from Shutdown Mode.



```
⊟ Frame 28: 144 bytes on wire (1152 bits), 144 bytes captured (1152 bits)
    Arrival Time: Sep 25, 2011 20:36:52.384804000 GTB Daylight Time
    Epoch Time: 1316972212.384804000 seconds
    [Time delta from previous captured frame: 0.078598000 seconds]
    [Time delta from previous displayed frame: 0.078598000 seconds]
    [Time since reference or first frame: 13.722584000 seconds]
    Frame Number: 28
    Frame Length: 144 bytes (1152 bits)
    Capture Length: 144 bytes (1152 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ip:udp:wol]
    [Coloring Rule Name: UDP]
    [Coloring Rule String: udp]
⊟ Ethernet II, Src: AsustekC_02:bb:e3 (00:1e:8c:02:bb:e3), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ⊞ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ⊞ Source: AsustekC_02:bb:e3 (00:1e:8c:02:bb:e3)
    Type: IP (0x0800)
⊞ Internet Protocol, Src: 192.168.2.13 (192.168.2.13), Dst: 192.168.2.255 (192.168.2.255)
⊞ User Datagram Protocol, Src Port: discard (9), Dst Port: discard (9)
⊞ Wake On LAN, MAC: DellComp_aa:dc:03 (00:08:74:aa:dc:03)
```

Figure 4.18: Details of the Magic packet

4.2.2 LAN Shutdown

In this subsection the available methods to shutdown or remotely shutdown a PC will be described. Afterwards the method were implemented in a Java application and tested. This application was created in order to operate between Microsoft Windows XP or latest OSs (i.e. both the source and destination PCs the OS must be Windows XP or latest).

Method Description

The method used to remotely shutdown a pc, using Windows's Command Prompt. There is an available method provided by Microsoft using the *shutdown* command. The use of this command is very simple and the specification[31] of the *shutdown* command is provided by Microsoft. The syntax of this command is:

```
shutdown [{-l|-s|-r|-a}] [-f] [-m [\\ComputerName]] [-t xx] [-c "message"] [-d[u][p]:xx:yy]
```

Table 4.12 indicates the arguments that follow after this command and the description of each argument provided by Microsoft.

[31] http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/shutdown.mspx?mfr=true

Table 4.12: The arguments of the shutdown command in Windows

Argument	Description
-l	*Logs off the current user, this is also the defualt. -m ComputerName takes precedence.*
-s	*Shuts down the local computer.*
-r	*Reboots after shutdown.*
-a	*Aborts shutdown. Ignores other parameters, except -**l** and ComputerName. You can only use **-a** during the time-out period.*
-f	*Forces running applications to close.*
-m [\\*ComputerName or IP*]	*Specifies the computer that you want to shut down.*
-t xx	*Sets the timer for system shutdown in xx seconds. The default is 20 seconds.*
-c	*Specifies a message to be displayed in the Message area of the System Shutdown window. You can use a maximum of 127 characters. You must enclose the message in quotation marks.*
-d [u][p]:xx:yy	*Lists the reason code for the shutdown. The following table lists the different values.*
-i	*Opens the GUI.*

An example which reboots \\MyServer after 60 seconds, forces all applications to close, indicates that the shutdown is planned, logs major reason code 125, and logs minor reason code 1, was provided by Microsoft[32]. The syntax is shown below:

```
shutdown -r -f -m \\MyServer -t 60 -d up:125:1
```

In regards to Linux based OSs, there are also similar commands (e.g. halt, reboot and power off) which can be used to sleep, reboot and shutdown a pc. Again this could be used locally or remotely between two or more Linux Based PCs. A challenging project would be to develop an application which could perform shutdown/hibernate/sleep/reboot between a Windows based and a Linux based PC. There are available solutions[32] that can implement such techniques (e.g. a Linux PC can be shutdown using a Windows PC via a SSH connection. The challenge is to investigate those methods and develop an application which implements those techniques.

Implementation

Before implementing this method as an application, the method was tested. Example: In case it is needed to remotely shut down a PC with the following IP address: 192.168.1.2 the command that must be executed will be:

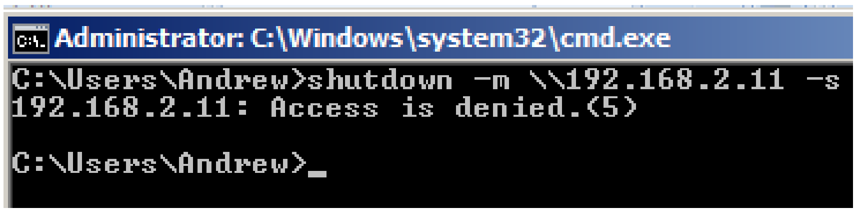
[32] http://lifehacker.com/5275652/shut-down-your-windows-pc-remotely-from-linux and http://www.voipphreak.ca/2007/10/22/shutdown-linux-from-windows-remotely-using-ssh-host-keys/

```
shutdown –m \\192.168.1.2 –s
```

To restart a PC the command has a parameter "-r" instead of an "-s":

```
shutdown –m \\192.168.1.2 –r
```

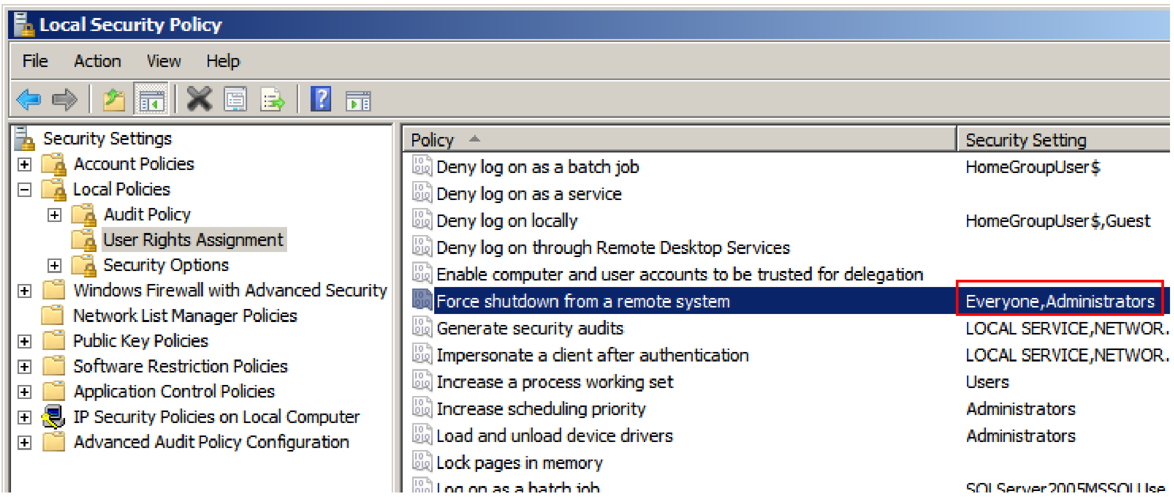
The results of the tests were negative since the access was denied in the remote PC. In most cases the default settings of the local security policy in the remote pc will allow only specific users to remotely shut down a pc. The remote system will reply with the following message "Access is Denied <5>" as shown is the example in Figure 4.19.



```
Administrator: C:\Windows\system32\cmd.exe
C:\Users\Andrew>shutdown -m \\192.168.2.11 -s
192.168.2.11: Access is denied.(5)

C:\Users\Andrew>_
```

Figure 4.19: Access denied in a remote shutdown example

This could be allowed to every user by changing the local security policy to the remote PC. To change this setting in the attribute "*Force shutdown from a remote system*" found in the following location "*Control Panel\System and Security\Administrative Tools\Local Security Policy\Local Policies\ User Rights Assignments*" the user "EVERYONE" must be added as shown in Figure 4.20.



- Security Settings
  - Account Policies
  - Local Policies
    - Audit Policy
    - User Rights Assignment
    - Security Options
  - Windows Firewall with Advanced Security
  - Network List Manager Policies
  - Public Key Policies
  - Software Restriction Policies
  - Application Control Policies
  - IP Security Policies on Local Computer
  - Advanced Audit Policy Configuration

Policy	Security Setting
Deny log on as a batch job	HomeGroupUser$
Deny log on as a service	
Deny log on locally	HomeGroupUser$,Guest
Deny log on through Remote Desktop Services	
Enable computer and user accounts to be trusted for delegation	
Force shutdown from a remote system	Everyone,Administrators
Generate security audits	LOCAL SERVICE,NETWOR.
Impersonate a client after authentication	LOCAL SERVICE,NETWOR.
Increase a process working set	Users
Increase scheduling priority	Administrators
Load and unload device drivers	Administrators
Lock pages in memory	
Log on as a batch job	SQLServer2005MSSQLUse

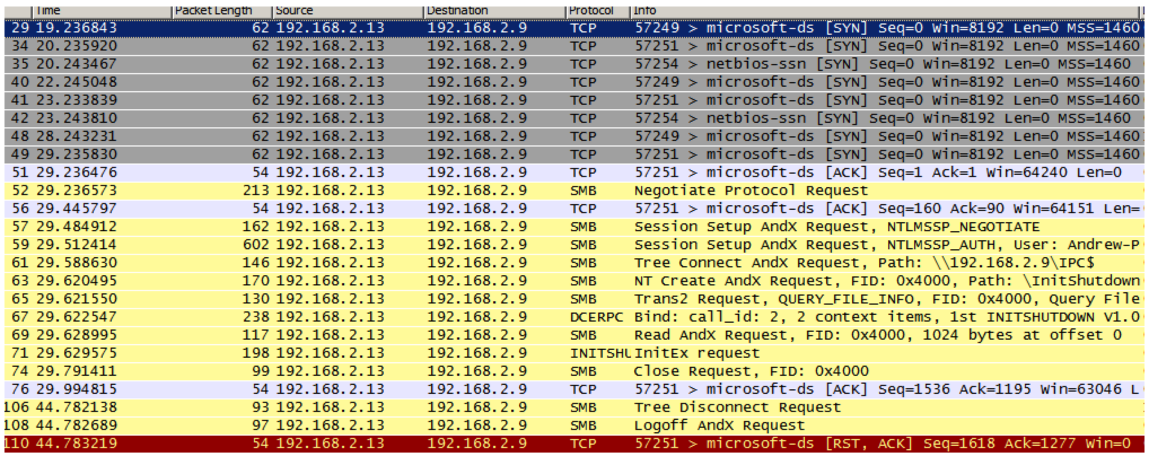
Figure 4.20: Setting up the local security policy

Afterwards a Java application was developed to exploit the *shutdown* command, using Java's *runetime.exec* command. This application was developed by the help of available java applications[33]. A simple script that executes the previously mentioned for a local PC (IP address 192.168.2.9) is shown in Figure 4.21.

```java
package lanshutdownw;
import java.io.*;
public class LanShutDownW {
  public static void main(String arg[]) throws IOException{
     String  ipaddr = "192.168.2.9";
     String operation = "-s";
     Runtime runtime = Runtime.getRuntime();
     Process proc = runtime.exec("shutdown -m \\\\"+ipaddr+" "+operation);
 System.exit(0);
}
}
```

Figure 4.21: Java code that remotely shuts down a PC

The previous example uses two arguments, *ipaddr* and *operation*, which represent the IP address and the operation respectively. This application was tested and it worked once the local policies of the remote pc were previously changed. By monitoring the NIC while executing the application it was noticed that this method uses the TCP protocol. Some of the actual packets are shown in Figure 4.22, where the source is 192.168.2.13 initiating the TCP connection and the remote was 192.168.2.9. The entire communication is shown in Figure 4.22.



```
    Time          Packet Length Source         Destination  Protocol Info
 29 19.236843           62 192.168.2.13   192.168.2.9   TCP    57249 > microsoft-ds [SYN] Seq=0 Win=8192 Len=0 MSS=1460
 34 20.235920           62 192.168.2.13   192.168.2.9   TCP    57251 > microsoft-ds [SYN] Seq=0 Win=8192 Len=0 MSS=1460
 35 20.243467           62 192.168.2.13   192.168.2.9   TCP    57254 > netbios-ssn [SYN] Seq=0 Win=8192 Len=0 MSS=1460
 40 22.245048           62 192.168.2.13   192.168.2.9   TCP    57249 > microsoft-ds [SYN] Seq=0 Win=8192 Len=0 MSS=1460
 41 23.233839           62 192.168.2.13   192.168.2.9   TCP    57251 > microsoft-ds [SYN] Seq=0 Win=8192 Len=0 MSS=1460
 42 23.243810           62 192.168.2.13   192.168.2.9   TCP    57254 > netbios-ssn [SYN] Seq=0 Win=8192 Len=0 MSS=1460
 48 28.243231           62 192.168.2.13   192.168.2.9   TCP    57249 > microsoft-ds [SYN] Seq=0 Win=8192 Len=0 MSS=1460
 49 29.235830           62 192.168.2.13   192.168.2.9   TCP    57251 > microsoft-ds [SYN] Seq=0 Win=8192 Len=0 MSS=1460
 51 29.236476           54 192.168.2.13   192.168.2.9   TCP    57251 > microsoft-ds [ACK] Seq=1 Ack=1 Win=64240 Len=0
 52 29.236573          213 192.168.2.13   192.168.2.9   SMB    Negotiate Protocol Request
 56 29.445797           54 192.168.2.13   192.168.2.9   TCP    57251 > microsoft-ds [ACK] Seq=160 Ack=90 Win=64151 Len=
 57 29.484912          162 192.168.2.13   192.168.2.9   SMB    Session Setup AndX Request, NTLMSSP_NEGOTIATE
 59 29.512414          602 192.168.2.13   192.168.2.9   SMB    Session Setup AndX Request, NTLMSSP_AUTH, User: Andrew-P
 61 29.588630          146 192.168.2.13   192.168.2.9   SMB    Tree Connect AndX Request, Path: \\192.168.2.9\IPC$
 63 29.620495          170 192.168.2.13   192.168.2.9   SMB    NT Create AndX Request, FID: 0x4000, Path: \InitShutdown
 65 29.621550          130 192.168.2.13   192.168.2.9   SMB    Trans2 Request, QUERY_FILE_INFO, FID: 0x4000, Query File
 67 29.622547          238 192.168.2.13   192.168.2.9   DCERPC Bind: call_id: 2, 2 context items, 1st INITSHUTDOWN V1.0
 69 29.628995          117 192.168.2.13   192.168.2.9   SMB    Read AndX Request, FID: 0x4000, 1024 bytes at offset 0
 71 29.629575          198 192.168.2.13   192.168.2.9   INITSHUInitEx request
 74 29.791411           99 192.168.2.13   192.168.2.9   SMB    Close Request, FID: 0x4000
 76 29.994815           54 192.168.2.13   192.168.2.9   TCP    57251 > microsoft-ds [ACK] Seq=1536 Ack=1195 Win=63046 L
106 44.782138           93 192.168.2.13   192.168.2.9   SMB    Tree Disconnect Request
108 44.782689           97 192.168.2.13   192.168.2.9   SMB    Logoff AndX Request
110 44.783219           54 192.168.2.13   192.168.2.9   TCP    57251 > microsoft-ds [RST, ACK] Seq=1618 Ack=1277 Win=0
```

Figure 4.22: Shutdown command monitored communication

[33] http://stackoverflow.com/questions/25637/shutting-down-a-computer-using-java and http://www.velocityreviews.com/forums/t514659-remote-shutdown-using-java.html

4.2.3 CPU utilization and other Information

It is very important to be able to monitor system's parameters and be able to implement those methods in applications. In this sub section such methods will be described and implemented in Java applications. These methods include exporting data regarding system's parameters like, CPU utilization, network traffic, memory, etc.

Method Description

The method to retrieve system's data in a Microsoft Windows OS is using the *typeperf* command. This method is specified by Microsoft[34] and operates in Windows XP or newest Windows editions. The syntax of this command is shown bellow.

```
typeperf [Path [path ...]] [-cf FileName] [-f {csv|tsv|bin}] [-si interval] [-o FileName] [-q [object]] [-qx [object]] [-sc samples] [-config FileName] [-s computer_name]
```

Table 4.13: The parameters of typeperf

Parameter	Description
-c { *Path* [*path ...*] | -cf *FileName* }	*Specifies the performance counter path to log. To list multiple counter paths, separate each command path by a space.*
-cf *FileName*	*Specifies the file name of the file that contains the counter paths that you want to monitor, one per line.*
-f { csv | tsv | bin }	*Specifies the output file format. File formats are* csv *(comma-delimited),* tsv *(tab-delimited), and* bin *(binary). Default format is* csv*.*
-si *interval* [*mm:*] *ss*	*Specifies the time between samples, in the [*mm:*]* ss *format. Default is one second.*
-o *FileName*	*Specifies the pathname of the output file. Defaults to* stdout*.*
-q [*object*]	*Displays and queries available counters without instances. To display counters for one object, include the object name.*
-qx [*object*]	*Displays and queries all available counters with instances. To display counters for one object, include the object name.*
-sc *samples*	*Specifies the number of samples to collect. Default is to sample until you press CTRL+C*
-config *FileName*	*Specifies the pathname of the settings file that contains command line parameters.*
-s *computer_name*	*Specifies the system to monitor if no server is specified in the counter path.*
/?	*Displays help at the command prompt.*

The parameters and their descriptions of this command are described in Table 4.13 and the general format for counter paths is:

[**\\***Computer*]\*object*[*parent***/***instance#index*]**\***counter*]

[34] http://technet.microsoft.com/en-us/library/bb490960.aspx

Some examples given by Microsoft[34] are:

To display processor's counters:

```
typeperf "\processor(_total)\% processor time"
```

To display memory's counters

```
typeperf "\Memory\Available bytes"
```

Implementation

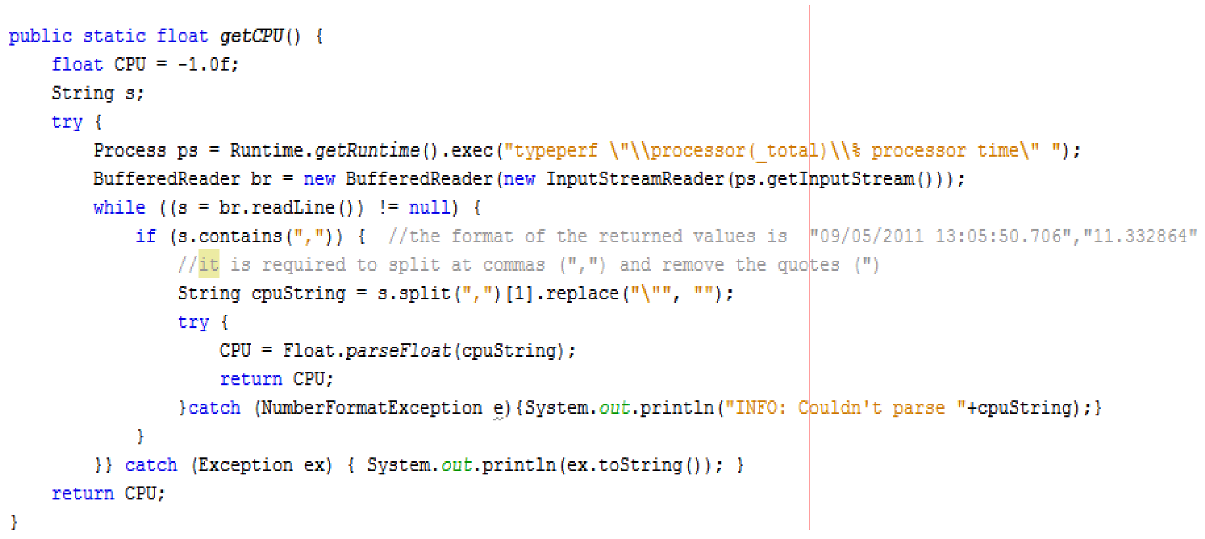
Before implementing this method in a Java application, it was tested in the windows command prompt. An example used to monitor the CPU utilization and export the data in a CSV file is shown below:

```
typeperf -cf cpumeasurement  -f csv "\processor(_total)\% processor time"
```

This example can be implemented in a .bat file and will run by simply double clicking this file. The command inside the .bat file must be:

```
typeperf -cf cpumeasurement  -f csv "\processor(_total)\%% processor time"
```

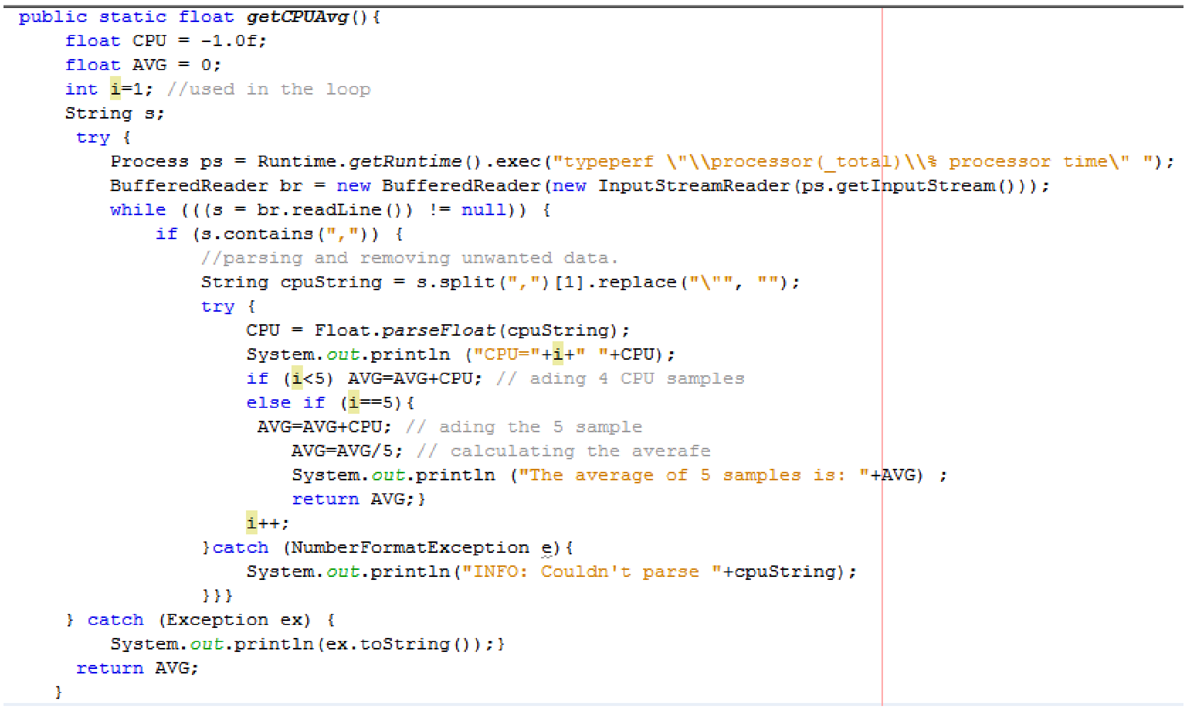
Afterwards the method was implemented in a Java Application. In general this application consists of two functions the *getCPU()* which collects one sample and parses the data retrieved to separate only the CPU (%) value; and the *getCPUAvg()* which collects five samples, separates only the CPU (%) value and calculate the average of these samples. These functions were developed with the help of the Smart IHU team. The getCpu() function is shown in Figure 4.23.



```java
public static float getCPU() {
    float CPU = -1.0f;
    String s;
    try {
        Process ps = Runtime.getRuntime().exec("typeperf \"\\processor(_total)\\% processor time\" ");
        BufferedReader br = new BufferedReader(new InputStreamReader(ps.getInputStream()));
        while ((s = br.readLine()) != null) {
            if (s.contains(",")) {  //the format of the returned values is  "09/05/2011 13:05:50.706","11.332864"
                //it is required to split at commas (",") and remove the quotes (")
                String cpuString = s.split(",")[1].replace("\"", "");
                try {
                    CPU = Float.parseFloat(cpuString);
                    return CPU;
                }catch (NumberFormatException e){System.out.println("INFO: Couldn't parse "+cpuString);}
            }
        }} catch (Exception ex) { System.out.println(ex.toString()); }
    return CPU;
}
```

Figure 4.23: The getCPU() java code

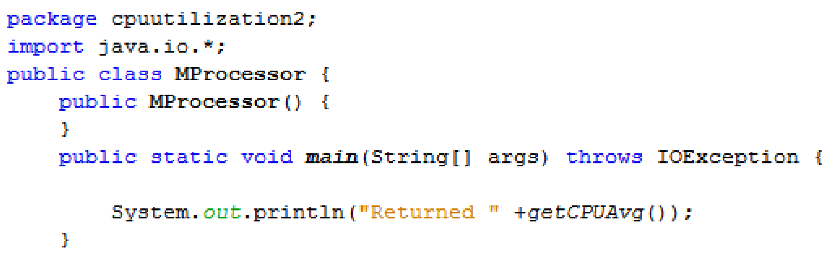
The source code of the getCPUAvg is shown in Figure 4.24 respectively.



```java
public static float getCPUAvg(){
    float CPU = -1.0f;
    float AVG = 0;
    int i=1; //used in the loop
    String s;
     try {
        Process ps = Runtime.getRuntime().exec("typeperf \"\\processor(_total)\\% processor time\" ");
        BufferedReader br = new BufferedReader(new InputStreamReader(ps.getInputStream()));
        while (((s = br.readLine()) != null)) {
            if (s.contains(",")) {
                //parsing and removing unwanted data.
                String cpuString = s.split(",")[1].replace("\"", "");
                try {
                    CPU = Float.parseFloat(cpuString);
                    System.out.println ("CPU="+i+" "+CPU);
                    if (i<5) AVG=AVG+CPU; // ading 4 CPU samples
                    else if (i==5){
                     AVG=AVG+CPU; // ading the 5 sample
                        AVG=AVG/5; // calculating the averafe
                        System.out.println ("The average of 5 samples is: "+AVG) ;
                        return AVG;}
                    i++;
                }catch (NumberFormatException e){
                    System.out.println("INFO: Couldn't parse "+cpuString);
                }}}
    } catch (Exception ex) {
        System.out.println(ex.toString());}
     return AVG;
    }
```

Figure 4.24: The getCPUAvg() java code

In both functions, comments were added in grey to explain the specific part of the code. Afterwards each function can be called inside the main. An example using *getCPUAvg()* and printing the result is shown in Figure 4.25.



```java
package cpuutilization2;
import java.io.*;
public class MProcessor {
    public MProcessor() {
    }
    public static void main(String[] args) throws IOException {

        System.out.println("Returned " +getCPUAvg());
    }
```

Figure 4.25: Example of Calling the getCPUAvg() function

4.2.4 Web Application

Development

After developing the three previously mentioned Java applications the next objective was to implement these applications in a single web application where each java application could be provided as an operation of a web service. The scope is to deploy this web application in a server and to be able to wake, shutdown PC’s in the LAN and additionally to be able to receive data about the CPU utilization of the server, both instant samples and average values.

The application developed was named ManagementWS and contains a JAX-WS web service based on the Java EE 6 platform.

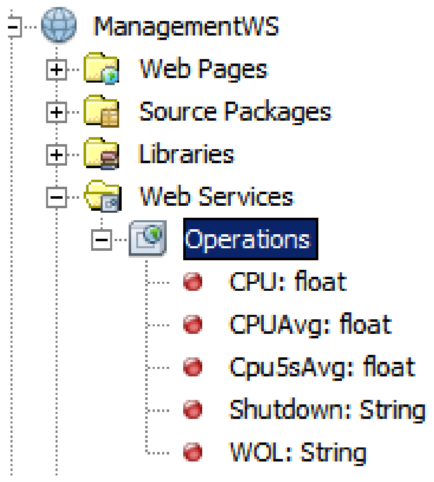

ManagementWS
Web Pages
Source Packages
Libraries
Web Services
Operations
CPU: float
CPUAvg: float
Cpu5sAvg: float
Shutdown: String
WOL: String

Figure 4.26: The structure of ManagementWS

As shown in Figure 4.26, ManagmentWS, consists of one web service, named Operations which is formed of five basic operations, CPU, CPUAvg, Cpu5sAvg, Shutdown and WOL. The operations are described in Table 4.14.

Table 4.14: Description of each operation of the WS

Operation	**Input type**	**Input parameter**	**Output type**	**Description**
CPU	-	-	float	Takes an instant sample of CPU util. (%) and returns the float value
CPUAvg	int	samples	float	Receives the number of samples, calculate the average for that number of samples. Last returns the average as a float.
Cpu5sAvg	-	-	float	Takes 5 samples, calculates the average of 5 samples and returns the average as a float
Shutdown	String, String	IP, Command	String	Receives two strings the Ip and the command (e.g. –s, -r etc.) and execute the command to remotely shutdown the required IP. Last in the end returns the message “Command executed” as a string.
WOL	String, String	IP, MAC	String	Receives two strings the IP and MAC and sends a magic packet to that destination. Last it returns the message “Magic Packet Sent” as a string.

These operations were inserted initially by adding operations to the web service as shown in Figure 4.27

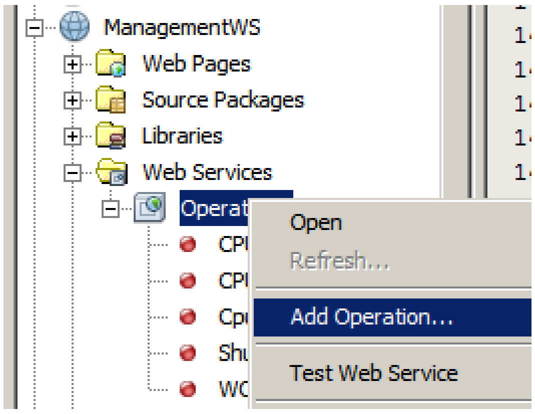
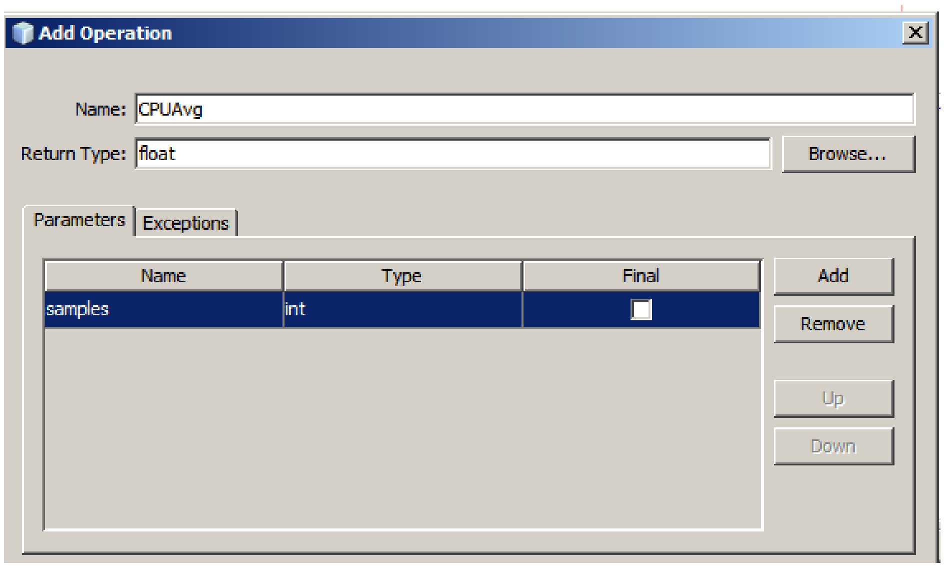


Figure 4.27: Adding an operation

Afterwards it is required to define the Return type and the input parameter name and type. Figure 4.28 indicates the example while adding the operation CPUAvg. In the same way the other four operations were added with the parameters shown in Table 4.14.



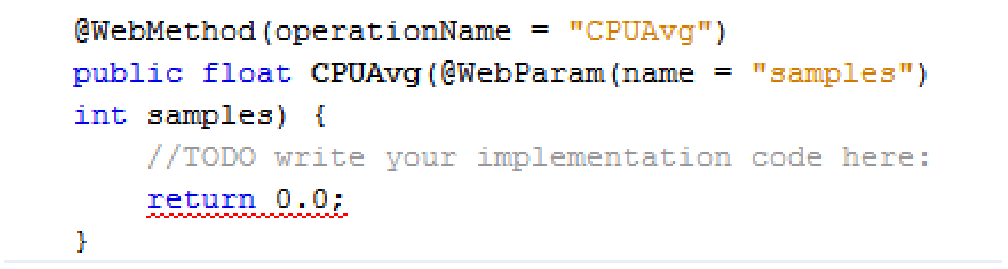
Name: CPUAvg

Return Type: float

Name	Type	Final
samples	int	☐

Figure 4.28: Example of adding the CPUAvg operation

After inserting the operation in the operations.java file the code shown in Figure 4.29 was generated automatically.



```
@WebMethod(operationName = "CPUAvg")
public float CPUAvg(@WebParam(name = "samples")
int samples) {
    //TODO write your implementation code here:
    return 0.0;
}
```

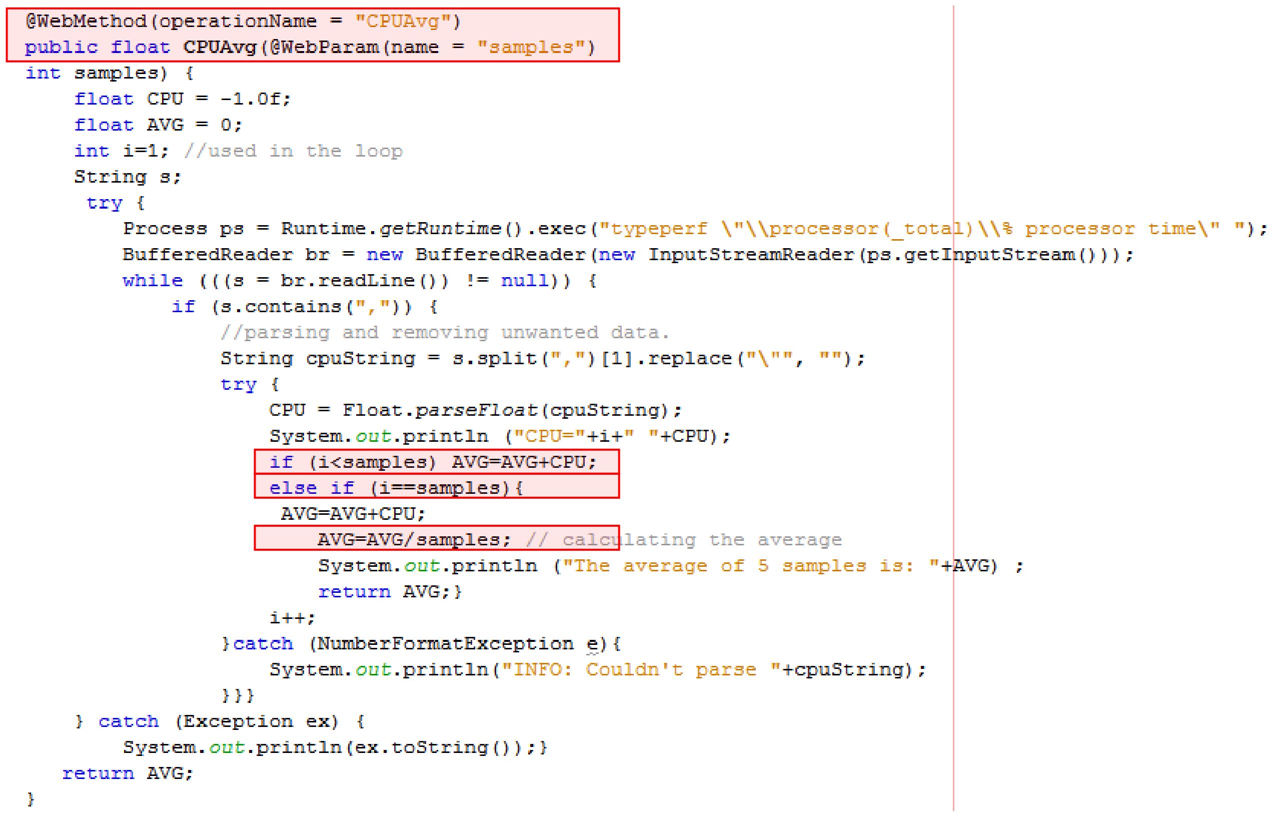
Figure 4.29: Operation’s initially generated code

The next step was to insert the code of the application developed in sub section 4.2.3 and adapting the inputs in that code. The final code is shown in Figure 4.30 and the differences in comparison to the Java Application developed previously are marked in red.

CPUAvg Operation is similar to the *getCpuAvg()* function. The differences are that the CPUAvg Operation is defined using @WebMethod(operationName= “CPUAvg) and inside the public float the *sample* parameter is defined using @WebParam(name=”samples”) int samples. The other differences are that instead of using a static number of samples as previously (i.e. 5 samples) the client who will consume this service can define the number of samples used. Therefore instead of five, inside the “if” statement, the parameter “samples”, which is an integer, was placed.

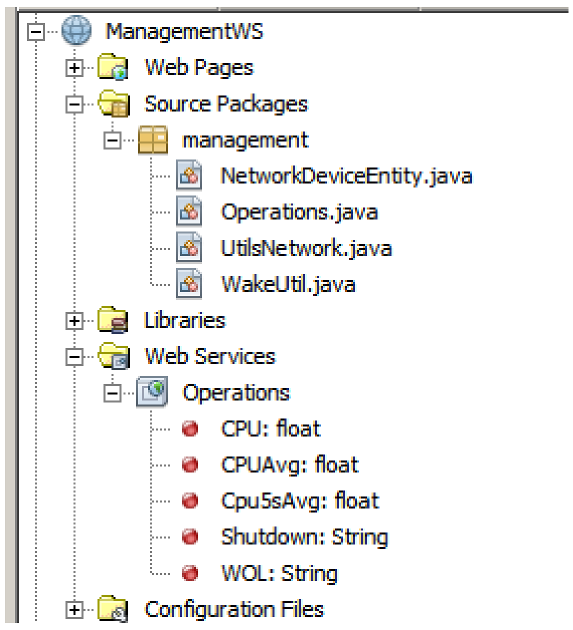
The procedure of the previous paragraph was repeated for the remaining four operations using the Java code developed in sub sections 4.2.1 and 4.2.2. And as a result the Operations.java contained all five operations.

In the case of the WOL operation it was required to insert the classes NetworkDeviceEntity.java, UtilisNetwork.java and WakeUtil.java. The final structure of the Management web application is shown in Figure 4.31.



```
@WebMethod(operationName = "CPUAvg")
public float CPUAvg(@WebParam(name = "samples")
int samples) {
    float CPU = -1.0f;
    float AVG = 0;
    int i=1; //used in the loop
    String s;
     try {
        Process ps = Runtime.getRuntime().exec("typeperf \"\\processor(_total)\\% processor time\" ");
        BufferedReader br = new BufferedReader(new InputStreamReader(ps.getInputStream()));
        while (((s = br.readLine()) != null)) {
            if (s.contains(",")) {
                //parsing and removing unwanted data.
                String cpuString = s.split(",")[1].replace("\"", "");
                try {
                    CPU = Float.parseFloat(cpuString);
                    System.out.println ("CPU="+i+" "+CPU);
                    if (i<samples) AVG=AVG+CPU;
                    else if (i==samples){
                     AVG=AVG+CPU;
                        AVG=AVG/samples; // calculating the average
                        System.out.println ("The average of 5 samples is: "+AVG) ;
                        return AVG;}
                    i++;
                }catch (NumberFormatException e){
                    System.out.println("INFO: Couldn't parse "+cpuString);
                }}}
    } catch (Exception ex) {
        System.out.println(ex.toString());}
   return AVG;
}
```

Figure 4.30: The Java code of the CPUAvg operation



```
ManagementWS
    Web Pages
    Source Packages
        management
            NetworkDeviceEntity.java
            Operations.java
            UtilsNetwork.java
            WakeUtil.java
    Libraries
    Web Services
        Operations
            CPU: float
            CPUAvg: float
            Cpu5sAvg: float
            Shutdown: String
            WOL: String
    Configuration Files
```

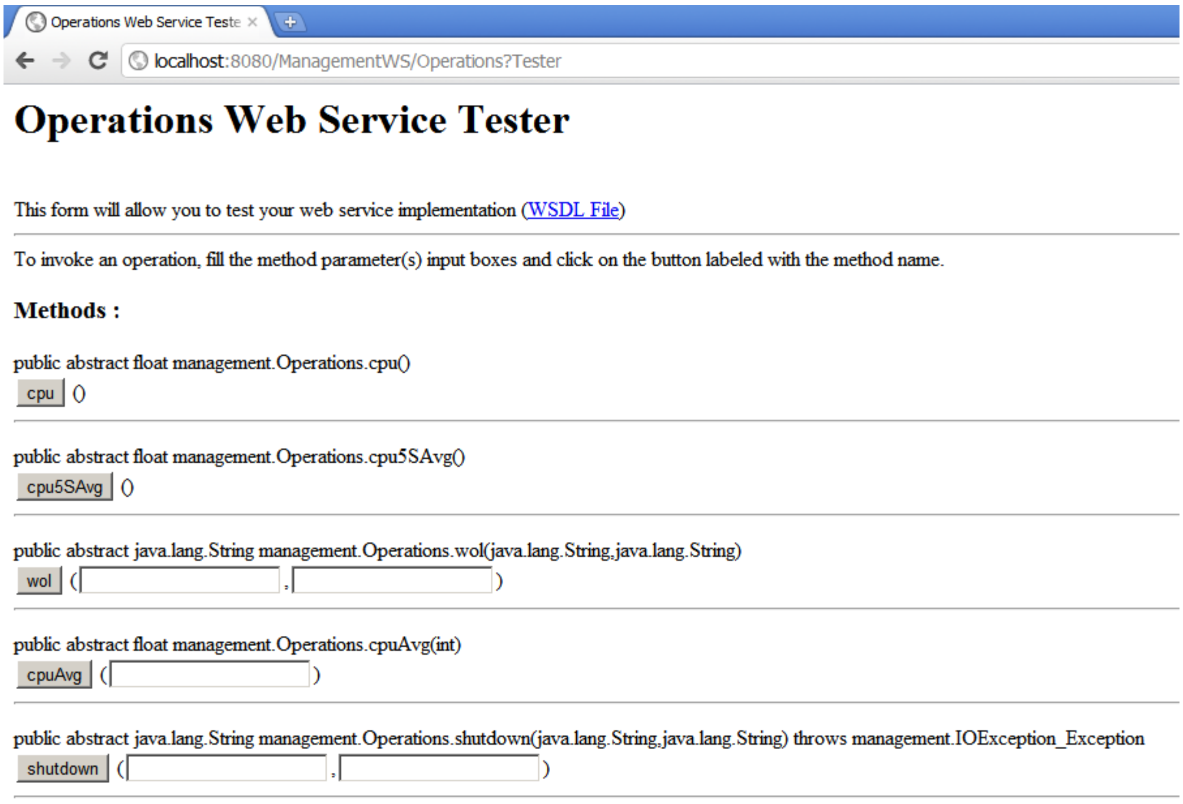
Figure 4.31: Final structure of ManagementWS

Testing Web Services

In order to test the web application's web service, it is required initially to deploy it using one of the servers provided by NetBeans (e.g. Glassfish) or in another server available. Afterwards it can be consumed by developing a Client application using the WSDL file location provided or by using the "Test Web Service" functionality provided by NetBeans and Glassfish. In the second method the web service can be tested (i.e. consumed) using a simple browser. In the case of ManagemetWS applications the deployment was done using Glassfish 3.1. After deploying it the operations were tested one by one using the location of where the service was developed, in this case in http://localhost:8080/ManagementWS/Operations?Tester. The tester is only available in Glassfish, not in Tomcat. The web page displayed is shown in Figure 4.32. This page can be accessed also in other PC's inside the LAN using the IP address instead of localhost (e.g. http://192.168.2.13:8080/ManagementWS/Operations?Tester) or even accessed by a WAN or the Internet. In case is needed to access this page over the Internet it is required to have a public IP address on the deployed PC (i.e. server). Another solution is using Network Address Translation (NAT) techniques to link the private IP inside the LAN, with the Public IP provided by the Internet Provider in a specific port of the public IP.

To test each operation, initially the user must enter the parameter/s, if needed, next the operation can be executed (i.e. the java code will be executed) and last a value will be returned in another page.

Example 1: In the page shown in Figure 4.32 the IP and the MAC address of a computer in the LAN was entered, in the WOL operation. The IP was 192.168.2.9 and the MAC address was 00:08:74:AA:DC:03. After pressing the "wol" button the page shown in Figure 4.33 was retrieved and in addition the computer in the LAN "woke up". The returned page indicates the parameters given as inputs and the outputs by the operation. Additionally the SOAP request and Response Messages were shown. In the request SOAP message the input elements, IP and MAC, and the payload of the SOAP message were marked in red. Respectively in the response SOAP message the return element and its payload was marked in red.


Operations Web Service Teste

localhost:8080/ManagementWS/Operations?Tester

Operations Web Service Tester

This form will allow you to test your web service implementation (WSDL File)

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract float management.Operations.cpu()

cpu ()

public abstract float management.Operations.cpu5SAvg()

cpu5SAvg ()

public abstract java.lang.String management.Operations.wol(java.lang.String,java.lang.String)

wol (,)

public abstract float management.Operations.cpuAvg(int)

cpuAvg ( )

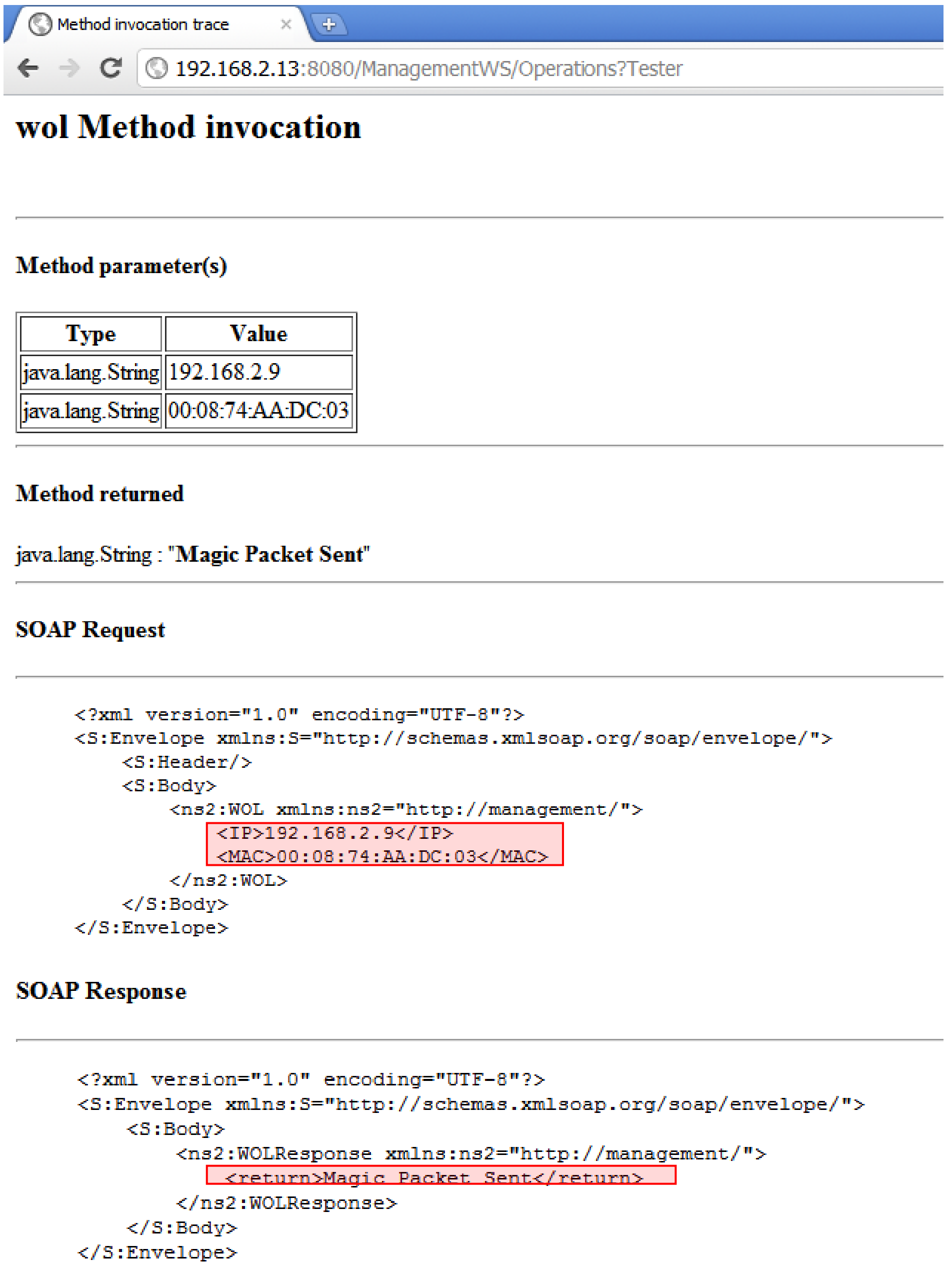
public abstract java.lang.String management.Operations.shutdown(java.lang.String,java.lang.String) throws management.IOException_Exception

shutdown (,)

Figure 4.32: Screenshot of the Web Service Tester

Example 2: In the test page the number 10 is entered in the box in the cpuAvg. Part of the retrieved page containing the input given and the result returned is shown in Figure 4.34. Additionally the SOAP Request/Response messages were displayed but they are not displayed in that figure. In this example the average CPU utilization of the server where the service was deployed, in 10 samples was 11.487%.

To conclude all the operations were tested and it was noticed that each one worked perfectly by returning the expected values (e.g. float, string) while executing the implemented java application and a result performing actions (e.g. measure CPU utilization, send Magic Packet).


Method invocation trace

192.168.2.13:8080/ManagementWS/Operations?Tester

wol Method invocation

Method parameter(s)

Type	Value
java.lang.String	192.168.2.9
java.lang.String	00:08:74:AA:DC:03

Method returned

java.lang.String : "**Magic Packet Sent**"

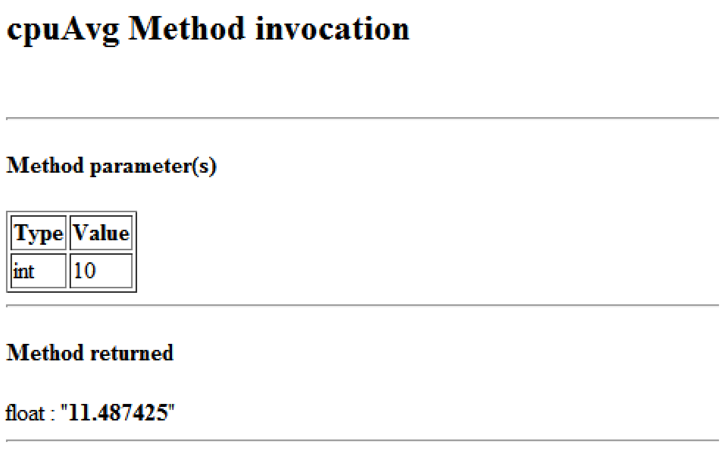
SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:WOL xmlns:ns2="http://management/">
            <IP>192.168.2.9</IP>
            <MAC>00:08:74:AA:DC:03</MAC>
        </ns2:WOL>
    </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:WOLResponse xmlns:ns2="http://management/">
            <return>Magic Packet Sent</return>
        </ns2:WOLResponse>
    </S:Body>
</S:Envelope>
```

Figure 4.33: Results after invoking the wol method


cpuAvg Method invocation

Method parameter(s)

Type	Value
int	10

Method returned

float : "**11.487425**"

Figure 4.34: Results after invoking CPUAvg

4.3 Correlation between CPU and Power Consumption

The results of the experiments conducted in sub section *4.1.2 Experiments Results and Comparisons* have shown that that the power dissipation increased while the server was active. Assumptions could be made regarding the reasons why this increase was noticed.

Initially it can be assumed that, since Plugwise's Stick was active and sending/receiving, part of the increase can be connected with it. Indeed part of the increase could be addressed with this device. Examining the values given it is shown that in the case of FoxBoard, when active there are spikes increasing from 2.13W to approximately 4W, respectively for the Mini Note Book an increase from 15W to 17W and in PC2 there was an increase from 48W to 90W. The nominal power dissipation of Stick is 0.375W as given by the manufacturer. Therefore the increase can be correlated also with other parameters regarding the PC operation (e.g. CPU utilization, network traffic, graphics etc.). This is expected since when the PC is idle some of these circuits are not used or are used less then when active. Therefore the power dissipation is lower. In the opposite way, when the PC is active, some of these circuits are operating and requiring additional power. Another factor that can influence the increase of the power dissipation is temperature. If the temperature of these devices will increase which results in requiring more power to cool (e.g. the fans operate faster when temperature is increased therefore requiring more power) in addition when the temperature of the of a switching PSU increases the also the thermal loses.

In present there are many research projects which correlate the CPU utilization and other parameters with the power dissipation (i.e. energy consumption). Therefore in this section the CPU utilization and the power dissipation will be investigated in order to detect patterns which correlate these two parameters.

Experiment

An experiment was conducted using applications and devices shown in Table 4.1 and Table 4.2, in sub section *4.1.2.* The applications used were PwScript to measure power and a .bat file containing the typeperf command to export the percentage of the CPU utilization in a CSV file. The devices used were one Stick, one Plugwise Smart Plugs and PC1. PC1 was running both these applications while the Plugwise Stick was connected in one of the USB ports and the smart plug was connected its plug to measure the power dissipation. This experiment was conducted on 27th of August 2011 from 14:24 to 14:42. During this time the utilization was increased, running applications, and turning the PC idle for a while. This increase was caused eight times as shown in the peaks of Chart 4.15. Additionally the power measured is illustrated in Chart 4.16.

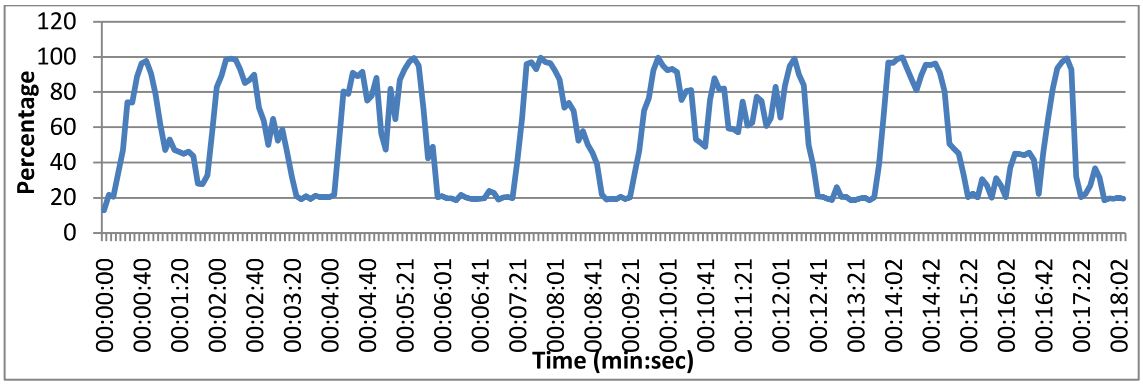


Chart 4.15: CPU utilization in PC1

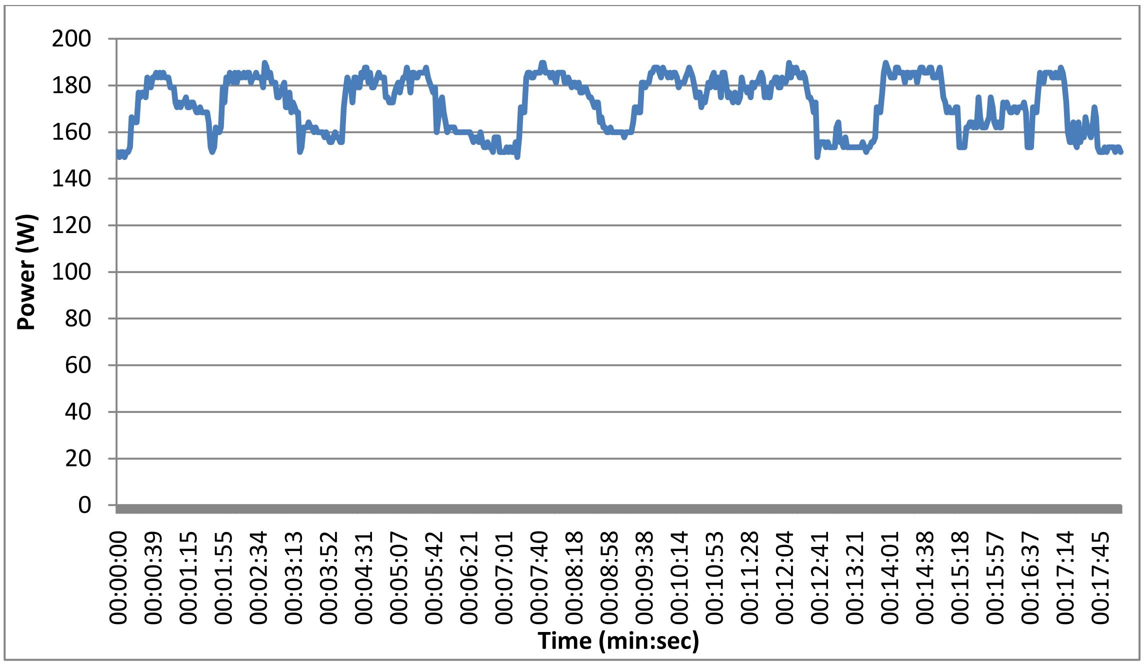
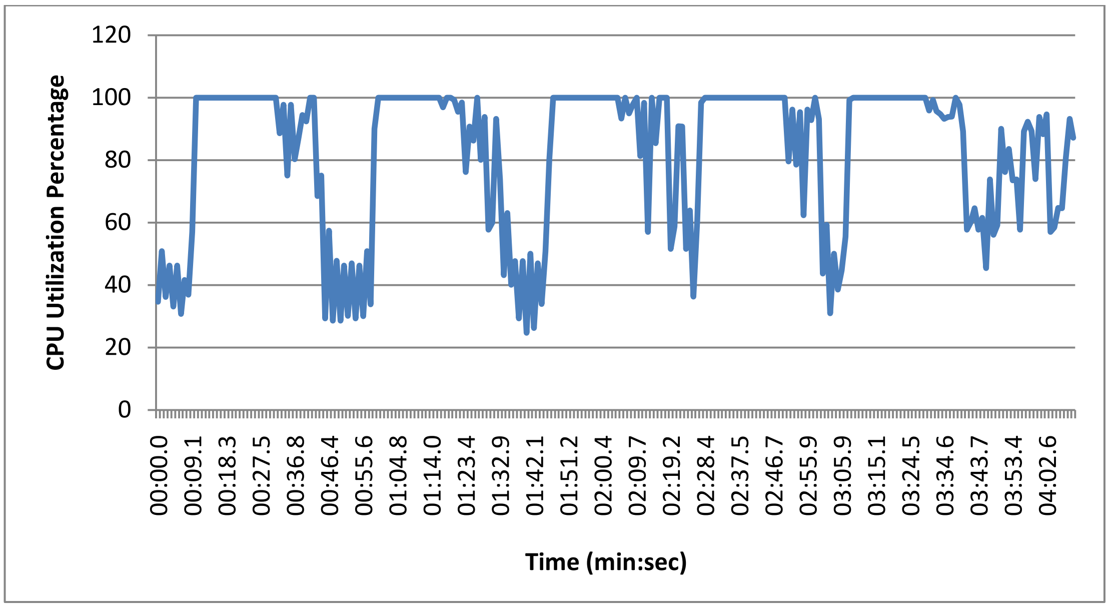


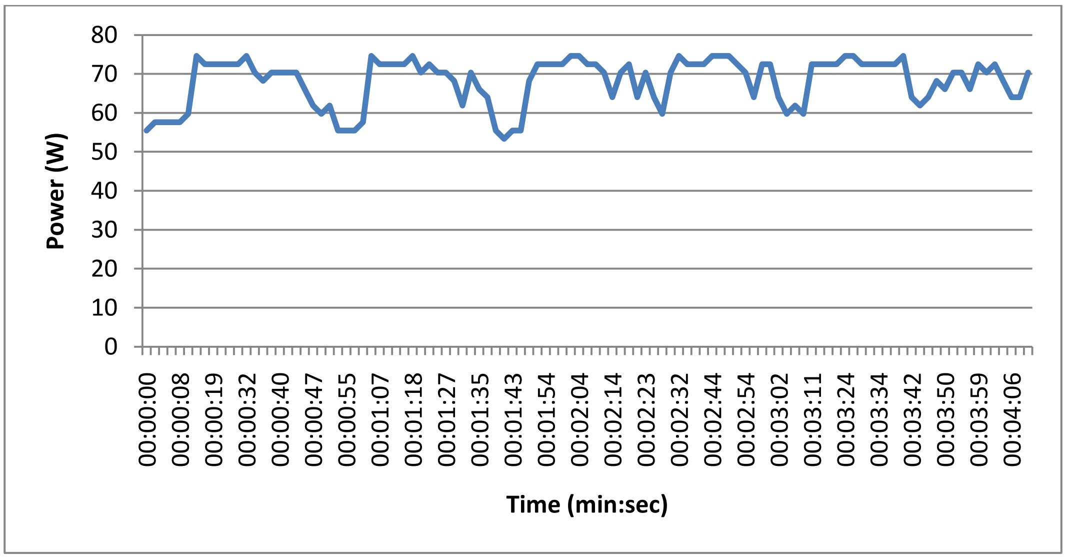
Chart 4.16: Power dissipation in PC1

Though the amount of the increase is not proportional, (i.e. the CPU fluctuates from 19% to 100 % and the power dissipation from 154W to 189W) by viewing these charts in temporal association, it was clearly noticed that the power dissipation is correlated to the CPU utilization, and therefore with the activity of the PC.

In addition, the same experiment was conducted in another PC with different CPU and memory resources. The second PC was a Dell Optiplex 330 and its parameters are: *Processor:* Intel(R) Core(tm) 2 duo E440 @ 2.00GHz; *RAM*: 2GB; 2X250GB hdd; *OS:* Windows 7; PSU: 305W max. The results of this experiment are shown in 4.17 and 4.18. Again the by observing these charts, the correlation between CPU utilization and power dissipation it is clearly shown.



4.17: CPU Utilization in Dell Optiplex 330



4.18: Power Dissipation in Dell Optiplex 330

5 Conclusions

The rapid developments in computer science, computing devices and the internetworking infrastructure have helped the implementation of concepts like Service Oriented Architecture (SOA) and Ambient Intelligence (AmI).

A Significant increase in the processing power and memory resources has added intelligence to devices, enabling the execution of more sophisticated algorithms. This evolution and development was also due to size reduction and energy efficiency, facilitating portability and extending the energy autonomy. In addition the reduced cost has driven in the implementation of such features in everyday devices, introducing intelligence to the surrounding environment. The lower cost of electronic devices and sensors has facilitated their massive deployment in various environments.

This progress requires an interconnection infrastructure for devices which process large amounts of data and have high bandwidth requirements, and devices with low bandwidth requirements equally. The research and development of protocols have increased the capacity and efficiency of the signal transport medium. Portability of devices has leaded in many improvements in wireless networks. In addition the energy autonomy is not only related to the hardware improvements but also with the development and implementation of energy efficient wireless protocols. The existence of distributed sensors in places where there is no power supply has leaded to the development of energy aware protocols in wireless sensor networks (e.g. ZigBee) which extend the battery lifetime and therefore the energy autonomy of these devices.

The developments in the previous sectors are also related with progress in computer science where complex and efficient algorithms are developed and implemented in the existing systems. Another aspect is the evolution in the available application's architecture while implementing new technologies and concepts like SOA. Advances in artificial intelligence in combination to machine to machine communication could result in the creation of devices which would operate independently providing services to humans and existing systems.

Combining the previously mentioned advances could lead in the creation of “high end systems” which could improve people’s quality of life and overall experience while introducing interacting systems that adapt to the environment, noticing patterns and acting upon user’s request or by making decisions based on their intelligence. In addition, such systems could also reduce the operational costs and the overall efficiency of the system.

Many research projects have focused in implementing AmI in system which would perceive, reason and act based on its intelligence. These projects involve the development and implementation of both software and hardware components. Smart Homes assist and alert the users in their everyday activities providing a better quality of life. Applications in transportations, in energy monitoring, in enterprise operations could create and intelligent environment.

Such a case of an AmI system is the Smart IHU research project which is based on the previously mentioned concepts, technologies, applications and hardware in order to monitor energy, and other parameters (e.g. movement, temperature etc.). The collection of these data will not only display this information to the administrating personnel, but in combination with complex algorithms that make decisions and take actions in order to improve quality of life and reduce the operational costs while saving energy and reducing CO2 emissions. The integration of the different wireless sensor networks available, in a single application platform facilitated the aggregation of data related by distributed nodes in a central point for further processing. Afterwards the implementation of SOA using web services has enabled the use of functions and data to be used by other systems and also manage the available hardware remotely.

After studying the available literature regarding the previously mentioned concepts and available devices two main improvements were suggested in the Smart IHU project. These two suggestions involved hardware, to directly reduce the energy consumption of the network infrastructure, and software by developing applications in order to monitor and manage the existing PCs in a LAN.

Hardware

In regards to hardware three devices was investigated: an ARM9 based and Linux embedded single board computer (SBC), named FoxBoard, a Toshiba NB250 Mini notebook, and a Dell Pentium 4 PC. These devices were used as a gateway in scenarios using the web application of the Smart IHU project. While performing experiments it was noticed that the PC and Notebook was performing faster than the SBC. There was a significant difference in power dissipation between these devices while executing the exactly same procedure. The PC was the most energy inefficient. This can be related to the old technology equipment which is less energy efficient than in recent devices. The PC was excluded and cannot be used as a gateway due to the significantly high energy consumption in comparison to the other two devices. Between the remaining two devices the Mini Notebook consumes six times more energy than FoxBoard and is approximately from 2.5 to 5 times faster than FoxBoard, depending on the operation of the Plugwise Web Service (e.g. switch on device or read power). The fact that FoxBoard's response time is high, therefore is slow, must be further investigated using an SBC with more than 140MB of RAM, because FoxBoard had limited memory resources and the Swap technique was used in it. Therefore the slower operation of FoxBoard may be related to swap since flash memory, which is considered "slow", was used as RAM. In case a SBC with that amount of RAM could have the same response time with the Notebook then the SBC could be considered as a perfect solution since the power dissipation is significantly low (e.g. less than 5W in most cases of the ARM based SBC) . Another factor would be the investigation of the aggregation of multiple clients and devices. Experiments regarding the response time could be performed. Under the existing circumstances FoxBoard could be considered as a solution but it would be more suitable applications which are not time sensitive and a low client aggregation. An intermediate system (e.g. pico-ITX) can be estimated to be placed in between FoxBoard and Mini Notebook in regards to power dissipation. In general in order to decide the appropriated device, estimations must be done about the aggregation of Clients.

Software Applications

The software developed in this work consisted of applications to perform the following remote operations to a PC in a LAN: wake up, shutdown/reboot/hibernate, receive CPU utilization. Even though these software utilities are very helpful in environments with distributed PCs there are two main problems that were noticed while using them. First the interoperation between different OSs and second the acknowledgement if the operation was successfully executed. In the former, Wake on LAN can be excluded since depends on the NIC and also it was functional after testing it between different OSs, but the CPU utilization and the remote shutdown can only be executed in Windows OS in regards to the client. The solution to this could be the investigation of SSH and other techniques in order to develop a more sophisticated application that could at least operate from Windows to Linux OS and vice versa.

In regards to the second problem, the CPU utilization could be excluded since it returns a float value and can be noticed if it was executed or no. The WOL and the Shut down return only a message that the packet or the command was sent. There is still feedback but it is between the client and the server, not between the client and the listening PC. In the WOL operation it is possible that the UDP packet would not be received by the listening PC. Therefore to notice that the PC woke up supplement techniques (e.g. PING using ICMP messages) must be used and serve as positive acknowledgements (ACKs). The shutdown command, as monitored by wire shark, uses TCP and there is a response on the terminal but when sending a command to an IP which does not correspond to a host, the system responds with and error after approximately 25seconds. This could cause a serious problem if this ACK would be used in the Web Application, since it could bind the server for a reasonable time period where other users wouldn't be served.

The development of software that addresses, detects and solves the previously mentioned issues, would result in a more efficient and scalable software application.

Bibliography

Aarts, E. H., & Encarnacao, J. L. (2008). *True Visions: The Emergence of Ambient Intelligence* (Second ed.). Berlin: Springer.

ACME Systems. (n.d.). *FOX Board G20 - Linux Embedded SBC*. Retrieved August 25, 2011, from http://www.acmesystems.it/?id=FOXG20

Baldauf, M., Dustdar, S., & Rosenberg, F. (2007). A survey on context-aware systems. *Int. J. Ad Hoc and Ubiquitous Computing* (pp. 263-277). Inderscience Enterprises Ltd.

(1999). *Bluetooth Specification Version 1.0A, Part A : Radio Specification.*

Cerami, E. (2002). *Web Services Essentials.* O'Reilly.

Chappell, D., & Jewell, T. (2002). *Java Web Services.* O'REILLY Media.

Christin, D., Mogre, P. S., & Hollick, M. (2010). Survey on Wireless Sensor Network Technologies for Industrial Automation: The Security and Quality of Service Perspectives. *Future internet* , 96-125.

Cook, D. J., Augusto, J. C., & Jakkula, V. R. (2007). *Ambient Intelligence:Technologies, Applications, and Opportunities.* Elsevier B.V.

Dario, P., Mazzolai, B., & Laschi, C. (2011, May 13). *P. Dario, B. Mazzolai and C. Laschi – Dustbot*. Retrieved August 9, 2011, from http://www.smart-urban-stage.com: http://www.smart-urban-stage.com/rome/ideas/paolo-dario-barbara-mazzolai-and-cecilia-laschi-dustbot/

De Amicis, R., Conti, G., Piffer, S., & Prandi, F. (2011). *Service oriented computing for Ambient Intelligence to support management of transport infrastructures.* Springer.

Dwivedi, K. A., & Vyas, P. O. (2010). Network Layer Protocols for Wireless Sensor Networks: Existing Classifications and Design Challenges. *International Journal of Computer Applications (0975 – 8887)* , 30-34.

Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., et al. (2004). *Patterns: Service-Oriented Architecture and Web Services.* IBM International Technical Support Organization.

Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., & Jansen, E. (2005, March). The Gator Tech Smart House:A Programmable Pervasive Space. *IEEE Computer Society* , pp. 64-74.

Helal, S., Winkler, B., Lee, C., Kaddourah, Y., Ran, L., Giraldo, C., et al. (2003). *Enabling location-aware pervasive computing applications for the elderly.* IEEE.

IEEE Computer Society. (2006). *Part 15.4: Wireless Medium Access Control MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs).* New York: IEEE.

IHU School of Science and Technology. (2010, March). *Projects: Smart International Hellenic University Application of ICT Systems for Sustainable Growth, Energy Efficiency and Better Quality of Life*. Retrieved September 13, 2011, from IHU School of Science and Technology: http://www.tech.ihu.edu.gr/index.php/projects/smart-international-hellenic-university.html

Jabjone, S., Chatchaiyadej, S., & Chantavichean, T. (2009). SMART HOUSE MANAGEMENT SYSTEM WITH RFID. *International Conference on the Role of Universities in Hands-On Education* (pp. 793-799). Chiang-Mai: Rajamangala University of Technology Lanna.

LEWIS, F. L., & Grant, A. R. (2004). *Wireless Sensor Networks.* New York.

Mahmoud, Q. H. (2005, April). *Articles: Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI).* Retrieved August 19, 2011, from Oracle Technology Network: http://www.oracle.com/technetwork/articles/javase/soa-142870.html

NetBeans Community. (2011). *NetBeans IDE - The Smarter Way to Code*. Retrieved September 21, 2011, from NetBeans IDE 7.0 Features: http://netbeans.org/features/index.html

Neves, J., Santos, M. F., & Machado, J. M. (2007). *Argumentation-Based Decision Making in Ambient Intelligence Environments.* Berlin: Springer.

Nordin, H. (2008). *Your Computer and the Climate: Make a change today – Save the planet tomorrow.* TCO Development.

OASIS. (n.d.). *OASIS UDDI Specification TC*. Retrieved September 8, 2011, from OASIS Advancing open standards for the information society: http://www.oasis-open.org/committees/uddi-spec/faq.php

Ort, E. (2005). *Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools.* Sun Microsystems.

Padmanabhuni, S., Chaudhari, A. P., Bharti, S., & Kumar, S. (2007, June 7). *WSDL 2.0: A Pragmatic Analysis and an Interoperation Framework*. Retrieved September 9, 2011, from Web 2.0: Article: http://soa.sys-con.com/node/219029

PHILIPS. (2005). *Philips Research - Technologies Robotics*. Retrieved August 09, 2011, from www.research.philips.com: http://www.research.philips.com/technologies/projects/robotics/index.html

Raghavendra, C. S., Sivalingam, K. M., & Znati, T. (2006). *Wireless Sensor Networks.* Springer.

Rahman, A., El Saddik, A., & Gueaieb, W. (2008). *Wireless Sensor Network Transport Layer: State of the Art.* Heidelberg: Springer.

Rakotonirainy, A., & Tay, R. (2004). In-Vehicle Ambient Intelligent Transport Systems (I-VAITS): Towards an Integrated Research. *Procedings of 7th international IEEE conference on intelligent transportation systems (ITSC 2004)*, (pp. 648-651). Washington DC.

Roy, A., & Sarma, N. (2010). Energy Saving in MAC Layer of Wireless Sensor Networks: a Survey. *National Workshop in Design and Analysis of Algorithm (NWDAA).* India: Tezpur University.

Saffiotti, A., & Broxvall, M. (2005). PEIS Ecologies: Ambient Intelligence meets Autonomous Robotics. *Proceedings of the sOc-EUSAI (Smart Objects and Ambient Intelligence) conference*, (pp. 275-280). Grenoble.

Weber, V., Vickery, G., & OECD. (2009). Smart Sensor Networks: Technologies and Applications for Green Growth. *ICTs, the environment and climate change.* Helsingør: OECD.

Wilson, J. (2005). *Sensor Technology Handbook.* Oxford: Elsevier.

Appendix A: Terms

Acronym	Definition
AC	Alternative Current
AES	Advanced Encryption Standard
AI	Artificial Intelligence
AmI	Ambient Intelligence
AODV	Ad hoc On Demand Distance Vector
API	Application Programming Interface
ARM	Advanced RISC Machine
ASK	Amplitude Shift Keying
BIOS	Basic Input Output System
BPSK	Binary Phase-Shift Keying
CAD	Computed Aided Design
CAM	Computer Aided Manufacturing
CSMA	Carrier Sense Multiple Access
CSMA/CA	Carrier Sense Multiple Access/ Collision Avoidance
CSV	Comma-Separated Values
DC	Direct Current
DDR	Double Data Rate
DLL	Data Link Layer
DNS	Domain Name System
DPI	Debug Port Interface
DSSS	Direct Sequence Spread Spectrum
ebXML	Electronic Business using XML
ESP	Electronic Stability Program

FCC	Federal Communications Commission
FDMA	Frequency Division Multiple Access
FSK	Frequency Shift Keying
FTP	File Transport Protocol
GPRS	General Packet Radio Service
GPS	Global Positioning System
HCI	Human-Computer Interaction
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IHU	International Hellenic University
IP	Internet Protocol
ISM	Industrial Scientific Medical
ISOS	Intelligent Sensors' Operating System
J2EE	Java 2 Enterprise Edition
JAX-WS	Java API for XML Web Services
JDK	Java Development Kit
JRE	Java Runtime Environment
JSP	Java Server Page
JVM	Java Virtual Machine
LAN	Local Area Network
LVDT	Linear Variable Differential Transformer
MAC	Medium Access Control
MACA	Multiple Access with Collision Avoidance
NIC	Network Interface Controller
O-QPSK	Offset Quadrature Phase-Shift Keying
OASIS	Organization for the Advancement of Structured Information Standards

OSI	Open Systems Interconnection
PAN	Personal Area Network
PC	Personal Computer
PDA	Personal Data Access
PEIS	Physically Embedded Intelligent Systems
POW	Plugwise Over Web
PSK	Phase Shift Keying
PSU	Power Supplying Unit
QoS	Quality of Service
QTY	Quantity
RAM	Random Access Memory
RFID	Radio Frequency Identification
RISC	Reduced Instruction Set Computing
RTD	Resistance Temperature Detector
SMP	Sensor Management Protocol
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPIN	Sensor Protocols for Information via Negotiation
SQDDP	Sensor Query and Data Dissemination Protocol
SSH	Secure Shell
SDB	Subnet Directed Broadcasts
TADAP	Task Assignment and Data Advertisement Protocol
Tar	tape archive
TDMA	Time Division Multiple Access
VGA	Video Graphic Array
W3C	World Wide Web Consortium
WA	Web Application

WAN	Wide Area Network
WAR	Web Application ARchive
WiFi	Wireless Fidelity
WLAN	Wireless Local Area Networks
WOL	Wake On LAN
WOW	Wake On WAN
WPAN	Wireless Personal Area Networks
WSAN	Wireless Sensor and Actuator Networks
WSDL	Web Service Description Language
WSN	Wireless Sensor Network
XML	eXtensible Markup Language
UDDI	Universal Description Discovery and Integration
USB	Universal Serial Bus

Appendix B: Configuring FoxBoard

Create A Bootable microSD with Debian Lenny

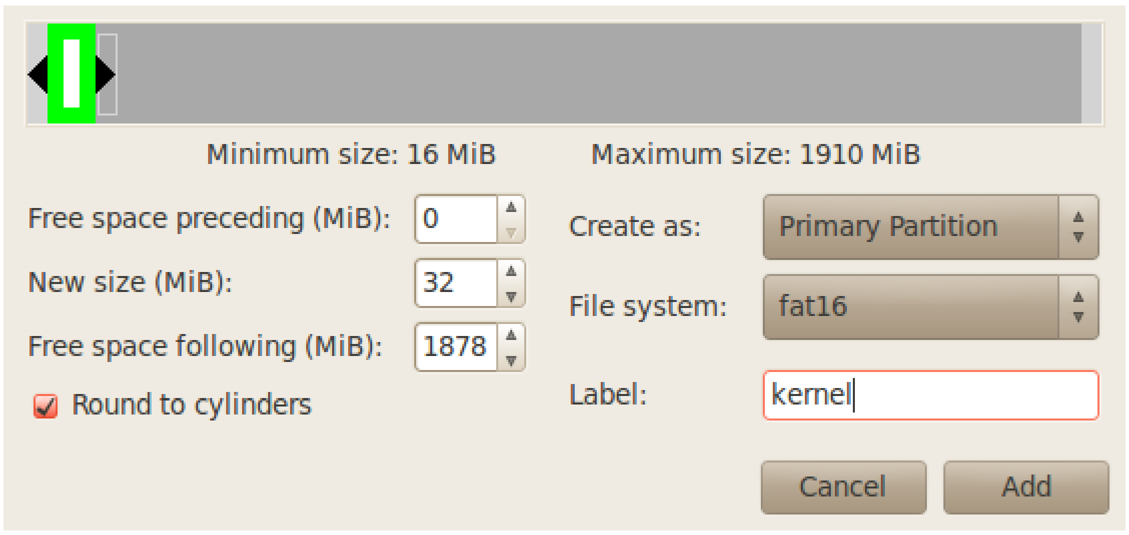
The procedure to create a bootable MicroSD card is described by ACME Systems[35]

In a new microSDs typically are formatted with a unique big FAT partition so the operations to do are :

- Delete the factory default big FAT partition
- Create the four partition requested by the FOX Board G20
- Copy the file inside the new partitions

Create a new partition using the command right click -> New with these parameters:

- New size: 32MB
- File system: fat16
- Label: kernel
- Leave all the other fields at default values



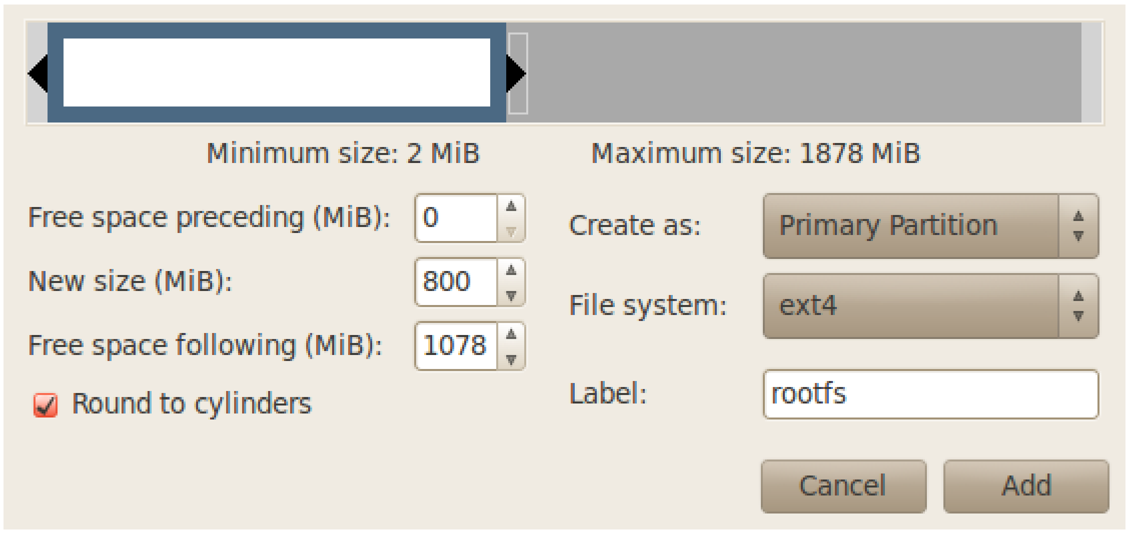
Minimum size: 16 MiB   Maximum size: 1910 MiB

Free space preceding (MiB): 0
New size (MiB): 32
Free space following (MiB): 1878
☑ Round to cylinders
Create as: Primary Partition
File system: fat16
Label: kernel

Create a new partition using the command right click -> New with these parameters:

[35] http://www.acmesystems.it/?id=foxg20_microsd_create

- New size: 800MB or more if you intend to install a lot of Linux packages
- File system: ext4
- Label: rootfs
- Leave all the other fields at default values

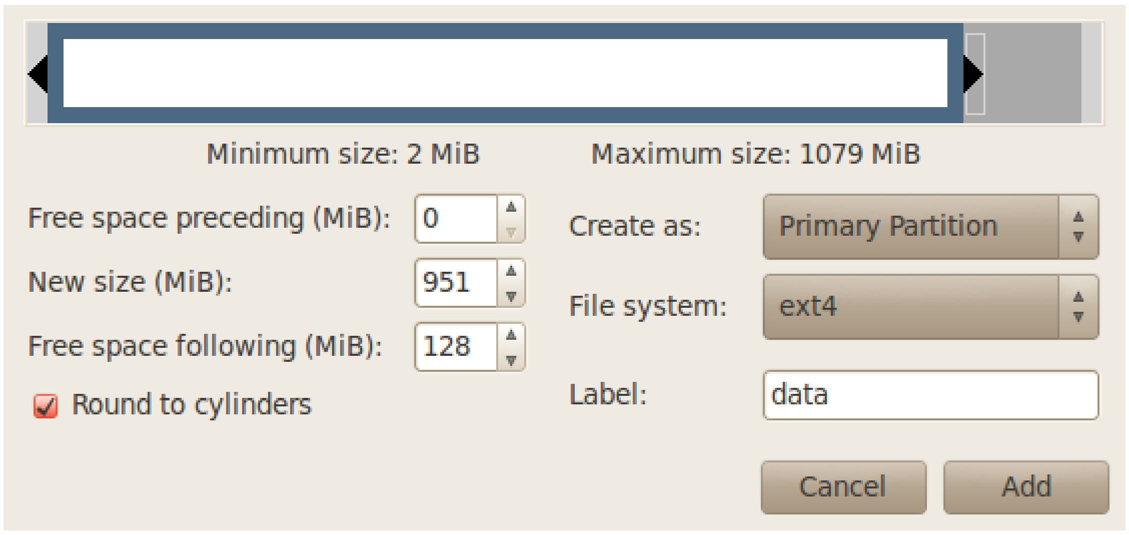

Minimum size: 2 MiB
Maximum size: 1878 MiB

Free space preceding (MiB): 0
New size (MiB): 800
Free space following (MiB): 1078
☑ Round to cylinders

Create as: Primary Partition
File system: ext4
Label: rootfs

Create a new partition using the command right click -> New with these parameters:

- File system: ext4
- Free space following: 128MB
- Label: data
- Leave all the other fields at default values


Minimum size: 2 MiB
Maximum size: 1079 MiB

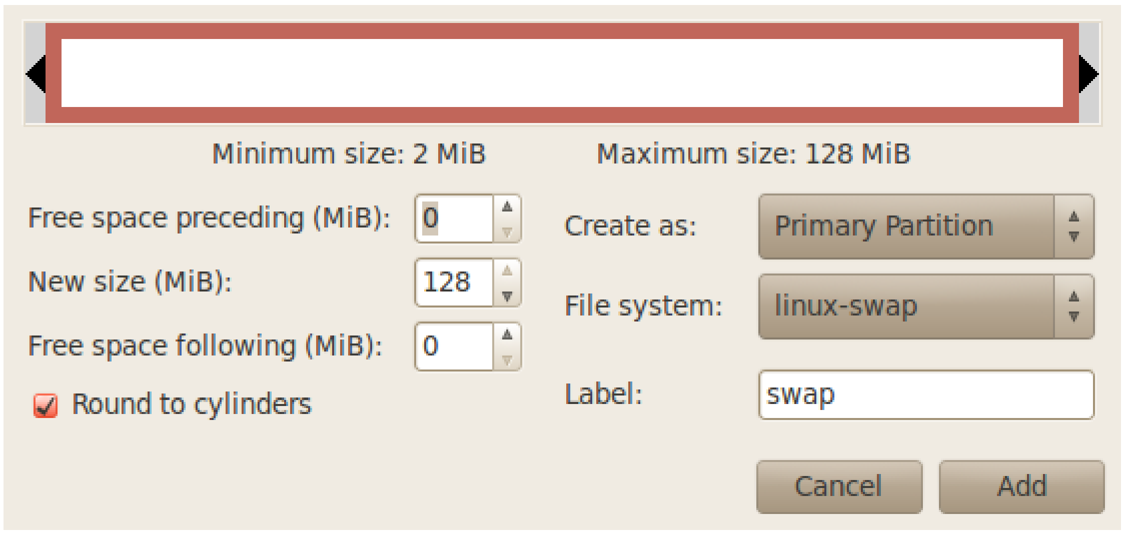
Free space preceding (MiB): 0
New size (MiB): 951
Free space following (MiB): 128
☑ Round to cylinders

Create as: Primary Partition
File system: ext4
Label: data

Create a new partition using the command right click - New with these parameters:

- New size: 128MB
- File system: linux-swap
- Label: swap
- Leave all the other fields at default values

- Click on the green sign to apply all the operations and exit from Gparted.


Minimum size: 2 MiB Maximum size: 128 MiB

Free space preceding (MiB): 0

New size (MiB): 128

Free space following (MiB): 0

Round to cylinders ✓

Create as: Primary Partition

File system: linux-swap

Label: swap

Remove the microSD, wait about 10 sec and insert again. Tree new partitions will be mounted automatically on:

- /media/kernel
- /media/rootfs
- /media/data

Now proceed to fill these partitions with the contents required by the FOX Board G20.

Kernel uImage and rootfs contents

Download the last snapshot from the binary repository then:

Copy the Linux Kernel uImage and parameters files in /media/kernel.

```
$ cp uImage /media/kernel
$ cp machtype.txt /media/kernel
$ cp cmdline.txt /media/kernel
```

Un-tar and copy the rootfs contents in /media/rootfs:

```
$ sudo tar xvjpSf rootfs.tar.bz2 -C /media/rootfs
```

Synchronize the microSD contents:

```
$ sync
```

Unmount all the microSD partition from your PC:

```
$ sudo umount /media/kernel
$ sudo umount /media/rootfs
$ sudo umount /media/data
```

Remove the microSD, insert it in your FOX Board G20 and try to boot it.

How to set a static IP address

Provided by ACME Systems (http://www.acmesystems.it/?id=foxg20_set_static_ip)

By default the FOX Board G20 gets the IP address from the DHCP server on your LAN this article explains how to set a static IP address

If you have a Linux PC simple insert the FOXG20 microSD card. If you are using Ubuntu Linux it will mount the microSD file system on /media directory.

Edit the file /media/etc/network/interfaces placing # chars on these lines:

```
#auto eth0
#iface eth0 inet dhcp
```

Then uncomment the *iface eth0 inet static* line and tailor the details for your local set-up, for example:

```
auto eth0
iface eth0 inet static
  address 192.168.1.90
  netmask 255.255.255.0
  gateway 192.168.1.1
```

Setting the System Clock

The read the currently System Clock type:

```
smartihu:~# date
Fri Oct  8 17:44:42 CEST 2010
```

To set it type:

```
debarm:~# date -s "8 OCT 2010 18:45:00"
Fri Oct  8 18:45:00 CEST 2010
```

This time is valid until the board is on. When you turn-off it the system clock is lost.

Setting the Real Time Clock

The read the Hardware CLock type:

```
debarm:~# hwclock -r
Fri Oct  8 17:46:43 2010  -0.004115 seconds
```

This time is read at startup from the Linux Kernel and mantained with the on-board RTC Lithium battery.

To set the Hardware Clock with the System Clock value type:

```
debarm:~# hwclock -w
```

Now check it typing:

```
debarm:~# date
Fri Oct  8 18:49:02 CEST 2010
debarm:~# hwclock -r
Fri Oct  8 18:49:10 2010  -0.004076 seconds
debarm:~#
```

Change the hostname on a running system

The hostname in saved in /etc/hostname.

Change it for example with myfox typing:

```
debarm:~# echo "myfox" > /etc/hostname
debarm:~# /etc/init.d/hostname.sh start
```

Then logout typing:

```
debarm:~# logout
```

Login again and your prompt will be:

```
myfox:~#
```