Journal of
Information Technology & Software Engineering

Review Article Open Access

Software Development of Integrated Wireless Sensor Networks For Real Time Monitoring of Oil and Gas Flow Rate Metering Infrastructure

Ayuba John[1*], Safwana Haque[2], Abdulazeez Yusuf[1] and Mu'azu Dauda[1]

[1]Department of Computer Science, Federal University Dutse, Jigawa State of Nigeria, Nigeria
[2]Department of Computer Science and Engineering, College of Engineering and Technology (CEAT), IUBAT - International University of Business Agriculture and Technology, Bangladesh

Abstract

This research work describes the software development of an integrated wireless sensor network (WSN) for real-time remote monitoring of oil and gas flow rate metering infrastructure. The wireless sensor network includes flow sensors, pumping machines, microcontrollers, Wi-Fi or wireless links and a database server. The pumping machine first pumps oil and gas, then a flow rate sensor measures the quantity and communicates the flow rate data in meter cube per second via a wireless link to the database server for further processing. The resulting data is stored on the database server while the continuous flow rate is displayed in real time on the website designed as the human-computer interaction (HCI) interface through which, the quantity of the oil and gas lifted can easily be accessed without having to collect the data from several stations manually. The cost-effective indices for reporting the quantity of oil and gas lifting are of significance to any accountable and transparent system. However, manual processes have not been able to do that in real time. In this paper, our main focus is to provide the software development life cycle involved in the design and implementation processes of the system prototype using iterative incremental mode and the complete software development architecture for the wireless sensor network.

Keywords: Wireless sensor networks; Software development; Embedded devices; Real time monitoring and sensor node

Introduction

National resources like oil and gas are two very important assets to the economic development of oil producing countries. Maintaining the economic progress of these countries strongly depends on having good control over its oil and gas industries [1,2]. The rapid development of oil producing countries and increase in demands for energy; petroleum and natural gas have become important assets for these countries. The oil wellheads to export terminals flow rate monitoring for oil and gas are important parts of the national treasure vital to the national economy [1,3,4]. Therefore, maintaining the economic progress of these countries strongly relies on having good control over these resources.

The report of the auditing firm Price Water House Cooper Limited (PWC) expressed a lack of transparency and good accountability by the Nigerian National Petroleum Corporation (NNPC) [1,5]. It also showed that the Central Bank of Nigeria (CBN) cannot validate how much money is remitted to the federation account, which signifies the presence of corruption in the oil and gas sector in Nigeria. The former Governor of CBN Sanusi Lamido Sanusi threw down the gauntlet in 2013, when he alleged that $20 billion oil money was unremitted to the national treasury [1,6]. A committee set up by finance minister Ngozi Okonjor Iweala said about $10 billion was the figure. When the auditing giant, price water house cooper Limited was brought in, it reported that according to the government, only $1.48 billion should have been remitted to the treasury. The report which the presidency released apparently to clear itself of shielding corrupt officials said the oil giant should have refunded $4.29 billion. The fact that the central Bank of Nigeria, federal ministry of finance, Budget office of the federation and NNPC cannot agree on revenue realized from crude oil sales (Nigeria's major revenue source) presents a compelling need to improve transparency and accountability processes in NNPC.

However, in an oil sector reform agendum of the present administration, it highlights that one of the biggest challenges that has remained unaddressed since the very first Nigeria Extractive Industries Transparency Initiative (NEITI) audit report and which has kept reappearing in subsequent ones is the absence of metering infrastructure for measuring the quantity of crude oil flows from wellheads to export terminals [1,7], and without metering at both ends of the pipeline, no one can say how much oil the country is producing, let alone what it is losing through theft and vandalism. Therefore, in this research work, the researchers designed a prototype system that aimed at solving the above challenges with less stress by conveniently monitoring the flow rate of the oil and gas, calculating the amount lifted and remotely sending the data to a database server. The results of the data can then be easily accessed through a web based interface where the detailed records of the oil lifted for a period of time can be easily retrieved [1,2,8]. This system also helps identify the exact amount of money expected to be remitted into the revenues treasury account [1].

A Wireless Sensor Network is an infrastructure comprised of an interactive computing wireless-based communication element that is characterized by a task oriented sensor node which gives an administrator the capability to implement, detect, and respond to proceeding occurrences in an identified location [1]. The location actually examined; can be a corporeal domain, a natural system, or an information technology (IT) structure while the administrator is typically a civic legislative, commercial, or industrial body [9]. The sensor network constitute of some basic modules such as: an assembly of disseminated sensors, communicating wireless network, a central point for information gathering; and established computing resources at the central point capable of handling parallel data occurrences, status enquiring, and data mining [1]. Nodes in the sensor network usually require one or more sensors, a wireless communication device, a

***Corresponding author:** Ayuba John, Department of Computer Science, Federal University Dutse, Jigawa State of Nigeria, Nigeria, Tel: 2348060486709; E-mail: ayuba.john@fud.edu.ng

Received March 09, 2018; **Accepted** April 25, 2018; **Published** April 30, 2018

Citation: Ayuba J, Safwana H, Abdulazeez Y, Ma'azu D (2018) Software Development of Integrated Wireless Sensor Networks For Real Time Monitoring of Oil and Gas Flow Rate Metering Infrastructure. J Inform Tech Softw Eng 8: 233. doi: 10.4172/2165-7866.1000233

microcontroller and an energy source; normally a battery [3]. Wireless technology has the potency to be useful in many aspects; eradicating the requirement for cables, contributing to the abridged installation and effective costs. It facilitates installations in isolated regions and permits cost effective, momentary and portable systems. It also, allows a different variety of uses; mainly in the area of health, safety and environment. For Oil and Gas Industry, the corporate and vital applications are correlated to the monitoring of real-time process control, safety, maintenance and production performance [1,9]. Therefore, a wireless sensor network can be defined in this scenario as a network of embedded devices that can communicate the information gathered from a monitored field through wireless links. The WSN in this work is designed to maintain a near real-time visibility of oil and gas flow rate measurement for good transparency and accountability.

Literature Survey

Incremental software development addresses the time of delivery of software products instead of providing a massive, unchanging system that does not permit individual variation after a long development time, smaller portions are implemented consecutively. If appropriate, this method has many benefits over the traditional waterfall method [10]. First of all, requirements can be ordered so that the most significant ones are delivered first and benefits of the original system increased earlier. Subsequently, less essential requirements are left until later and thus, if the schedule or financial plan is not adequate the least significant requirements are the ones more likely to be omitted. Subsequent here means that customers obtain part of the system earlier on and are more likely to support the system and deliver feedback on it. Another advantage is that, being smaller; the time frame and cost for each delivery phase is easier to estimate, and that consumer feedback can be acquired at each phase and strategies adjusted consequently. Lastly and possibly most essentially, being an incremental approach tolerates for a much better feedback to changes for requirements.

These benefits have predominantly been exploited in agile methods [11]. Agile methods shared the same idea of release planning. For instance, in Extreme Programming [12], a software product is first described in relations to 'consumer feedback' which are informal depictions of user requirements. In the planning process, these feedbacks are arranged by means of apparent value to the consumer and fragmented into a sequence of releases based on estimates of the period each feedback in an increment will take to implement. An iteration plan is developed for delivering that release. Each increment released is a complete product for use to the client. At any stage, a feedback may be added and integrated into a future release. Delivering software incrementally requires a process of evaluating requirements and incorporating them into increments. The distinctive advice from an advocate of incremental delivery is to resolve the increments and then deliver these accordingly to user values [13]. User values may be measured in relation to cost-benefit calculations or a combination of these and risk evaluations [11]. This accepts that requirements are now allocated to increments and only then are the increments ordered. However, often the case is that any given requirement could be delivered in one, several or even all releases.

Therefore, in this research work, a system prototype was designed of a wireless sensor network which can simply monitor's the flow rate of oil and gas in a real time, calculates the amount being lifted and remotely send the data to a database server to a distance control center or base station which can be easily access in order to get the detail records of the oil lifted for a period of time [1]. In this paper, our main focus is to provide the software development life cycle involved in the designed and implementation processes of the system prototype designed using iterative incremental model.

Sensor Nodes Software Development

The sensor nodes are controlled by the microcontroller which is programmed using C language to monitor the flow rate of oil and gas. An iterative model design was adopted and the process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. For each iteration design, modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental). As with any real-time embedded system, the code must be elevated to achieve results within certain constraints. Owing to the sensitivity of the performance, effectiveness and lifespan of a wireless sensor network, the algorithm guiding network configuration, an iterative design, implementation, and test phase is vital [14,15]. Since the applications of wireless sensor networks differ so momentously, the algorithms involved in a particular kind of wireless sensor network must be elevated over a huge amount of calculation and design.

The development form for node application proposed in favor of the extremely iterative design configuration method; strictly follows distinctive software engineering practices of a node's component interface optimization that is implemented in the design phase shown in Figure 1. the evaluation by monitoring the outcomes leads to further iterations. the development scheme results in a definite application for the sensor nodes involved in particularly tailed parts.

Components requirement stage

To realize greater flexibility and enhance actual client requirement satisfaction, there is a cumulative tendency to develop and release software in an incremental mode [10]. In adopting this process, requirements are delivered in several releases and therefore, a decision should be made on which requirements to deliver in which release. Three main considerations which have been taken into account are the procedural priority's essentials in the requirements, the classically conflicting significances as determined by the representative stakeholders, as well as the stability between vital and accessible effort. The procedural priority constraints relate to circumstances where one requirement cannot be implemented until another is completed or where one requirement is implemented in the same increment as another one. Stakeholder preferences may be based on the apparent value or resolution of delivered requirements to the diverse stakeholders involved. The procedural priorities and distinct stakeholder priorities may be in conflict and hard to reconcile [10].

Therefore, in this section, several important aspects of the system requirement for the prototype design were considered, including component selection. Several factors were considered when selecting hardware components. The first consideration was the selection of the component package to use for each part. Most of the electronic components in this design were selected in a surface mount package. This reduced the foot print required for each component and the overall space required on the printed circuit board. The power and voltage rating for each of the components was considered to make sure that none of the components would become overstressed during use. Finally, cost and availability were considered; several of the components are as follows: Circuit design diagram, Power supply unit, Flow sensor unit, DC machine pump unit, Microcontroller unit, Display unit, WiFi module and the Repository central server unit.

Design and edit

The system design describes the desired structures and processes in detail, comprising rules, process diagrams, pseudo-code and other documentation. In this stage, the system software for the sensor nodes were coded using ‘C’ programming language by incorporating all the functionalities at the requirement stage, a database was created and integrated with the website designed for the implementation [1]. The flow rate software coded was embedded in a microcontroller; the codes are shown below:

```
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit Net_Wireless_MCW1001_Chip_RST at RD0_bit;
sbit Net_Wireless_MCW1001_Chip_RST_Direction at TRISD0_bit;


// Global variables
char myIpAdd[4] = {192, 168, 1, 1};
char myMacAdd[6] = {0x22, 0x33, 0x44, 0x55, 0x66, 0x88};
char netMask[4] = {255, 255, 255, 0};
char gatewayAdd[4] = {192, 168, 1, 1};
char remoteIpAdd[4];
unsigned int remotePort, localPort;
char strSSID[13] = "WiFi Control";
char channels[11] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
char wifiStatus, bindResponse, backLog, listenResponse;
char socketHandle, socketChild;
char response;
unsigned int wifiTmr = 0;
//////////////////////////////////////////////////////////////////////////
sbit flow_input at RC0_bit;
sbit led at RC5_bit;
sbit buzzer at RC4_bit;
sbit low at RA0_bit;
sbit high at RA1_bit;
sbit NEXT at RA2_bit;
sbit ENTER at RA3_bit;
sbit power at RA4_bit;
sbit led2 at RD3_bit;
sbit pump_output_Direction at TRISC2_bit;
sbit flow_input_Direction at TRISC0_bit;
sbit led_Direction at TRISC5_bit;
sbit buzzer_Direction at TRISC4_bit;
sbit low_Direction at TRISA0_bit;
sbit high_Direction at TRISA1_bit;
sbit NEXT_Direction at TRISA2_bit;
sbit ENTER_Direction at TRISA3_bit;
sbit power_Direction at TRISA4_bit;
sbit led2_Direction at TRISD3_bit;
//////////////////////////////////////////////////////////////////////////
unsigned long old_cumulative_cnt, cumulative_cnt, cumulative_val;
char min, min2, erase_num ;
unsigned int sec_result;
bit show, write_enable, write, read,inc, erase_bit,NEXT_bit, dec, onLoad, cnter, toggle, IsOn ;
unsigned short current_duty, old_duty;
unsigned char byte_1, byte_2, byte_3, byte_4;
/////////////////////////////////////timer variables///////////////////////////////
char Freq_txt[15];
unsigned int tmr1_Val=0;
unsigned short tmr0_cnt = 10;
bit flag;
char *Disp = "0000.00" ;
char *Disp_min = "00" ;
char *Disp_sec = "0.00" ;
char Total_reading []= "0000.00";
char K;
///////////////////////////////Function declarations//////////////////////////////////
void button_state();
void flow_convert();
void extract_byte();
void eeprom_rd_write();
void eeprom_erase();
void pwm_function();
void sys_init();
void button_delay();
void Lcd_setString(char row, char column, const char text[16]);
void Erase_Disp_Menu(char i);
void Standby_Disp();
void Background_Process();
void Timer_init ();
void Disp_Total (unsigned long total,unsigned int sec);
void InitWiFi();
/////////Interrupt routine which increment Time variable every second.//////////
void interrupt() {
 if(INTCON.F2==1) //if timer0 is overlfowed every 20ms
 {
 led2 =~ led2;
  if(--tmr0_cnt==0) //decrement and check tmr0_cnt until zer0 which means a 0.2 sec
 {
 min++;
 min2++;
 wifiTmr++;
 if(min2 >= 2){ T1CON.F0=0; flag=1; min2=0; }
 if( wifiTmr >= 2){ Net_Wireless_MCW1001_Time++ ; wifiTmr = 0; }
 if(min>=10){ // 5sec
 if(write_enable){
 write =1;
 old_cumulative_cnt = cumulative_cnt;
 write_enable = 0;
 }
 min =0;
 }
 tmr0_cnt = 49; //reset count variable
 }
 INTCON.F2=0; //reset timer0 flag
 TMR0=96; //Timer0 starts counting from 97 again
 }
 if(PIR1.F0==1) //if timer1 is overlfowed
 {
 PIR1.F0=0; //reset timer1 flag
 }}
//////////////////////////////button routine///////////////////////////////////
void button_delay(){
 delay_ms (50);
}
void button_state(){
 if (!low){
 delay_ms(20);
 dec = 1;
 }
 if (!high){
 delay_ms(20);
 inc = 1;
 }}
///////////////////////////Flow convert routine////////////////////////////////
void flow_convert(){
 if((flag==1)andand(!NEXT_bit)) //if one 0.2 sec elapsed
 {
 tmr1_Val = ((TMR1H << 8) + TMR1L); //get high and low from timer1
 TMR1L=0; TMR1H=0;
 T1CON.F0=1; //reenabling counting
 cumulative_cnt = cumulative_cnt+tmr1_Val;
 sec_result = (tmr1_val*60)/450;
 Disp_Total(cumulative_cnt, sec_result);
 if (OnLoad){ Lcd_Cmd(_LCD_CLEAR);}
 Lcd_Out (1,1,Disp_sec);
 Lcd_Out (1,10,Disp_min);
 Lcd_Out (2,1,Disp);
 tmr1_Val=0;
 flag=0; //reset variables
 }}
//////////////////////Extract bytes from long//////////////////////////////
void extract_byte(){
 byte_1 = (cumulative_cnt and 0xff );
 byte_2 = ((cumulative_cnt >> 8) and 0xff );
 byte_3 = ((cumulative_cnt >> 16) and 0xff );
 byte_4 = ((cumulative_cnt >> 24) and 0xff );
}
////////////////////////EEPROM WRI
```

```
TE//////////////////////////////////////
void eeprom_rd_write(){
 if (write){
 extract_byte();
 EEPROM_Write(1,byte_4);
 EEPROM_Write(2,byte_3);
 EEPROM_Write(3,byte_2);
 EEPROM_Write(4,byte_1);
 write = 0;
 led = 0;
 }
if (read){
 byte_4 = EEPROM_READ(1);
 byte_3 = EEPROM_READ(2);
 byte_2 = EEPROM_READ(3);
 byte_1 = EEPROM_READ(4);
 cumulative_val = (byte_4 << 24)+ (byte_3 << 16)+(byte_2 << 8)+
byte_1;
 read = 0;
 }}
/////////////////////////////EEPROM ERA
SE//////////////////////////////////////
void eeprom_erase(){
 if (!NEXT){
 button_delay();
 NEXT_bit =1;
 erase_num++;
 Erase_Disp_Menu(erase_num);
 }
 if ((!ENTER)andand(NEXT_bit)){
 button_delay();
 if (erase_num ==2){
 erase_bit =1;
 erase_num = 0;
 NEXT_bit =0;
 }
 if (erase_num ==3){
 onLoad = 0;
 NEXT_bit =0;
 Erase_Disp_Menu(0);
 delay_ms (1500);
 onLoad = 1;
 }}
 if (erase_num >=4){
 NEXT_bit =0;
 erase_num =0;
 onLoad = 1;
 }
 if (erase_bit){
 onLoad = 0;
 for(K=1;K<=4;K++){
 EEPROM_Write(K,0);
 Delay_ms(1);
 }
 cumulative_cnt = 0;
 cumulative_val = 0;
 Lcd_Cmd(_LCD_CLEAR);
 Lcd_setString(1,1,"CLEARING EEPROM!#");
 Lcd_setString(2,3,"PLEASE WAIT...#");
 Delay_ms(1500);
 erase_bit = 0;
 onLoad = 1;
 }}
///////////////////////////////////Lcd display
function///////////////////////
void Lcd_setString(char row, char column, const char text[16]){
 char k;
 for(k=0;k<=32;k++){
 if('#'==text[k])
 break;
 Lcd_Chr(row,column+k,text[k]);
 } //Lcd_setString(2,7,"CANCEL?#");
}
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / P W M
Routine////////////////////////////////////
 void pwm_function(){
 if ((dec)andand(current_duty>75)){
 current_duty -=10;
 }
 if (!power){
 button_delay();
 toggle =~ toggle ;
 }
 if (toggle){
 pump_output_Direction = 0;
 PWM1_Start();
 IsOn = 1;
 }
 if (!toggle){
 PWM1_Stop();
 pump_output_Direction = 1;
 IsOn = 0;
 }
 if ((inc)andand(current_duty<=245)){
 current_duty +=10;
 }
 if (current_duty != old_duty){
 old_duty = current_duty ;
 PWM1_Set_Duty(current_duty);
 }
 dec = 0;
 inc = 0;
 }
```

//////////////////////////////////Format Display Total/////////////////////////////

```
void Disp_Total (unsigned long total, unsigned int sec){
int per_sec;
total = (total*100)/450;
Disp[0]= total/100000 + 48;
Disp[1]= (total/10000)%10 + 48;
Disp[2]= (total/1000)%10 + 48;
Disp[3]= (total/100)%10 + 48;
Disp[5]= (total/10)%10 + 48;
Disp[6]= total%10 + 48;
//sec = sec*1000;
Disp_min[0] = (sec/10) + 48;
Disp_min[1] = (sec)%10 + 48;
sec = sec*100;
per_sec = (float)sec/60;
Disp_sec[0]= per_sec/100 + 48;
Disp_sec[2]= (per_sec/10)%10 + 48;
Disp_sec[3]= per_sec%10 + 48;
}
```

/////////////////////////////////Erase display menu//////////////////////////

```
void Erase_Disp_Menu(char i){
Lcd_Cmd(_LCD_CLEAR);
if (i ==0){
Lcd_setString(1,5,"CLEARING#");
Lcd_setString(2,4,"CANCELLED..#");
}
if (i ==1) {
Lcd_setString(1,2,"DO YOU WANT TO#");
Lcd_setString(2,3,"CLEAR METER?#");
}
if (i ==2) {
Lcd_setString(1,4,"CLEAR METER #");
Lcd_setString(2,8,"YES?#");
}
if (i ==3){
Lcd_setString(1,4,"CLEAR METER #");
Lcd_setString(2,6,"CANCEL?#");
}}
```

//////////////////////////////STANDBY SCREEN//////////////////////////////////

```
void Standby_Disp (){
if (onLoad){
Lcd_Cmd(_LCD_CLEAR);
Lcd_setString(1,6,"L/s#");
Lcd_setString(1,13,"L/m#");
Lcd_setString(2,10,"Ltr Tot#");
onLoad = 0;
}}
```

////////////////////////////////Background process/////////////////////////////

```
void Background_Process(){
OnLoad = 0;
button_state();
pwm_function();
eeprom_rd_write();
eeprom_erase();
if(cumulative_cnt > old_cumulative_cnt)
write_enable = 1;
}
```

///////////////////////////////Timer init///////////////////////////////

```
///
 void Timer_Init() {
  INTCON.F7=1; //enable global interrupt
  INTCON.F6=1; //enable pref interrupt
  INTCON.F5=1; //timer0 overflow interrupt
  OPTION_REG.T0CS = 0; //Internal instruction cycle clock
  OPTION_REG.PSA = 0; //0 = Prescaler is assigned to the Timer0
module
  OPTION_REG.PS2 = 1;
  OPTION_REG.PS1 = 1;
  OPTION_REG.PS0 = 0; //prescalar is set to 128
  tmr0_cnt=49;
  TMR0=96; //Timer0 starts counting from 96(TMR0 Preload)
  //timer1 config
  PIE1.F0=1; //Enables the TMR1 overflow interrupt
  T1CON.F1=1; //External clock (on the rising edge)
  T1CON.F3=0; //Oscillator is shut-off
  T1CON.F2=0; //Synchronize external clock input
  TMR1L=0; TMR1H=0; //clear timer1
 }
 ///////////////////////////////////////////////////////////////////////////////
 // Initialization of WiFi module
 void InitWiFi() {
  volatile char response;
  Net_Wireless_MCW1001_TimeToWait = 1;
  response = 1;
  while(response != 0)
    response = Net_Wireless_MCW1001_SetMode(_NET_
WIRELESS_MCW1001_CP_1,_NET_WIRELESS_MCW1001_
MODE_ADHOC); // Set "Connection Profile 1" mode (Ad-Hoc mode)


  //Net_Wireless_MCW1001_TimeToWait = 10;
  Net_Wireless_MCW1001_SetChannelList(1, channels); // Choose
only channerl one for communication
  Net_Wireless_MCW1001_SetSecurity_Open(_NET_WIRELESS_
MCW1001_CP_1); // Set open security level
          Net_Wireless_MCW1001_SetSSID(_NET_WIRELESS_
MCW1001_CP_1, strSSID); // Set SSID of network
  // Set network parameters
  Net_Wireless_MCW1001_SetIP(myIpAdd);
  Net_Wireless_MCW1001_SetNetworkMask(netMask);
  Net_Wireless_MCW1001_SetGatewayIP(gatewayAdd);
  Net_Wireless_MCW1001_SetMAC(myMacAdd);
  Net_Wireless_MCW1001_InitDHCP();
  Net_Wireless_MCW1001_SetArpTime(1); // Set gratuitous ARP
timing
  Net_Wireless_MCW1001_SetRetryCount(5, 0); // No retries for
the connection if the first attempt fails,
  onLoad = 1; // in order to establish a new network
 }
 ///////////////////////////////////////////////////////////////////////////////
 // Decoding command from TCP packet
 void CommandDecode(char command) {
 char txBuffer[17];
 unsigned int numOfSentBytes;
  switch(command) {
  case '1' : {
  txBuffer[0] = Disp[0];
  txBuffer[1] = Disp[1];
  txBuffer[2] = Disp[2];
  txBuffer[3] = Disp[3];
  txBuffer[4] = Disp[4];
  txBuffer[5] = Disp[5];
  txBuffer[6] = Disp[6];
  txBuffer[7] = ';';
  txBuffer[8] = Disp_min[0];
  txBuffer[9] = Disp_min[1];
  txBuffer[10] = ';';
  txBuffer[11] = Disp_sec[0];
  txBuffer[12] = Disp_sec[1];
  txBuffer[13] = Disp_sec[2];
  txBuffer[14] = Disp_sec[3];
  txBuffer[15] = ';';
  txBuffer[16] = '1';
      Net_Wireless_MCW1001_TCP_SendBytes(socketChild,
andtxBuffer, 17, andnumOfSentBytes);
  break;
  }}}
 ///////////////////////////////////TCP
sender//////////////////////////////////////////////
 // Polling TCP receiving buffer to see is there some received packet
 void CheckTcpReceive() {
  char response, dat[10], i;
  unsigned int numOfReceiveBytes = 0;
    response = Net_Wireless_MCW1001_TCP_
```

```
ReadBytes(andsocketChild, 10, dat, andnumOfReceiveBytes);
  // If there is no error, and number of received bytes is greater than zero
   if( (response == _NET_WIRELESS_MCW1001_NO_ERROR) andand (numOfReceiveBytes != 0) ) {
  for(i = 0; i < numOfReceiveBytes; i++) {
  CommandDecode(dat[i]);
  }}}
 ////////////////////////////////System Init/////////////////////////////////
///
 void sys_init(){
 OSCCON = 0b01111001;
 ANSEL = 0; // Configure AN pins as digital I/O
 ANSELH = 0;
 C1ON_bit = 0; // Disable comparators
 C2ON_bit = 0;
 TRISB = 0;
 TRISD = 0;
 PORTB =0;
 Delay_ms (100);
 Lcd_Init ();
 Delay_ms (100);
 current_duty = 105;
 old_duty = 0;
 toggle = 0;
 PWM1_Init(1000);
 Delay_ms (100);
 UART1_Init(19200);
 Delay_ms (1000);
 Net_Wireless_MCW1001_HwReset();
 Delay_ms (100);
 Lcd_Cmd(_LCD_CLEAR);
 Lcd_Cmd(_LCD_CURSOR_OFF);
 Lcd_setString(1,2,"FLOWRATE METER#");
 Lcd_setString(2,1,"Designed By John#");
 delay_ms (800);
 ////////////////////////////// pinout 1/0
config//////////////////////////////
 led_Direction = 0;
 led2_Direction = 0;
 flow_input_Direction = 1;
 //pump_output_Direction = 0;
 low_Direction = 1;
 high_Direction = 1;
 NEXT_Direction = 1;
 ENTER_Direction = 1;
 power_Direction = 1;
 buzzer_Direction =0;
 read = 1;
 write = 0;
 erase_bit = 0;
 Lcd_Cmd(_LCD_CLEAR);
 Lcd_setString(1,1,"System Initialzn#");
 Lcd_setString(2,3,"Please Wait..#");
 delay_ms (200);
 eeprom_rd_write();
 cumulative_cnt = cumulative_val;
 }
 void main() {
 sys_init();
 Timer_init();
 InitWiFi();
 Net_Wireless_MCW1001_Connect(_NET_WIRELESS_MCW1001_CP_1, andwifiStatus);
  Lcd_Cmd(_LCD_CLEAR);
  Lcd_setString(1,1,"Setting Up Wifi#");
  Lcd_setString(2,3,"Please Wait..#");
  delay_ms (2000);
  localPort = 10002;
  cnter = 0;
  onLoad = 1;
 while (1){
  /////////////////////////////////////////////////////////////////////
  socketHandle = 0;
  backLog = 1;
    Net_Wireless_MCW1001_SocketCreate(andsocketHandle, _NET_WIRELESS_MCW1001_SOCKET_TYPE_TCP); // Create TCP socket
          Net_Wireless_MCW1001_SocketBind(socketHandle, andlocalPort, andbindResponse); // Bind socket to the listen port
          Net_Wireless_MCW1001_TCP_Listen(socketHandle, andbackLog, andlistenResponse); // Prepare the socket to listen for connection
  // with one children socket.
  while(1) {
```

```
socketChild = socketHandle;
Net_Wireless_MCW1001_TCP_Accept(andsocketChild, andremotePort, remoteIpAdd); // Accept incoming conncetion
if (socketChild != 254) {
if (cnter == 0) { // Accept function set socketChild.
Lcd_Cmd(_LCD_CLEAR);
Lcd_setString(1,3,"Wifi Set Up#");
Lcd_setString(2,4,"Completed..#");
delay_ms (2000);
cnter = 1;
} // Connection established
break;
}}
// Connection established
while(1) {
Standby_Disp ();
Background_Process ();
flow_convert();
CheckTcpReceive();
if(Net_Wireless_MCW1001_TCP_Status(socketChild, 0xff) != 0) { // If connection closed improperly from remote side
Net_Wireless_MCW1001_SocketClose(socketChild); // Close sockets
Net_Wireless_MCW1001_SocketClose(socketHandle);
break;
}}}}
```

Testing and compilation

After the pseudo-code is developed, it is tested in line with the requirements to be certain of the product and to ensure that it is essentially solving the needs addressed and collected throughout the requirements phase. During this stage, all sorts of functional testing like unit testing, integration testing, system testing and acceptance testing are done as well as non-functional testing.

Evaluation

The system was developed and all the processes involved were evaluated. The questions asked and answered were:

- Does the system designed meet the initial requirements and the objectives?
- Is the system reliable and fault-tolerant?
- Does the system function according to the approved functional requirements?

However, these questions have been answered properly, then in tallying the evaluation of the software that was released; it is vital to evaluate the efficiency of the developmental process. This was done accordingly and it was observed from the result of the prototype that; the pumping machine first pumps oil and gas, then a flow rate sensor measures the quantity and communicates the flow rate data in meter cube per second via a wireless link to the database server for further processing as programmed in the microcontroller. The resulting data is stored on the database server while the continuous flow rate is displayed in real time on the website designed as the human-computer interaction (HCI) interface as shown in the Figure 2.

The flow sensor node software implementation

During this stage, the software developed was installed and evaluated in the system hardware's operational frame work. The main security actions for this phase comprises the following: integrating the software developed into its environment, planning and conducting system authorization activities in synchronization with testing of security controls and complete system certification activities. This phase also includes the maintenance of the system and any future updates or expansion of the system per say.

Software Architecture

The components requirement of the wireless sensor network demand for a service based architecture in a hierarchical cluster base for the sensor nodes. To realize greater flexibility and to enhance concrete customers requirement satisfaction, there is an increasing tendency to develop and deliver software in an incremental mode [11,12]. In adopting this process, requirements are delivered in releases and so a choice has to be made on which release. In Figure 2, the distinct flow sensor nodes interact with the middleware embedded in a microcontroller to perform the function dictated by the wireless sensor network application. The administration terminal and the user terminal are the connection point independent external actor; which interacts with the web designed interface to evaluate the results from the wireless sensor network application. The architecture in Figure 2 explicitly explains the behavior of a single node sensor network application, it cannot assign tasks to individual nodes and it can be deduced that the middleware embedded in the microcontroller is the one handling the task for the wireless sensor network and acts as flow rate monitoring coordinator.

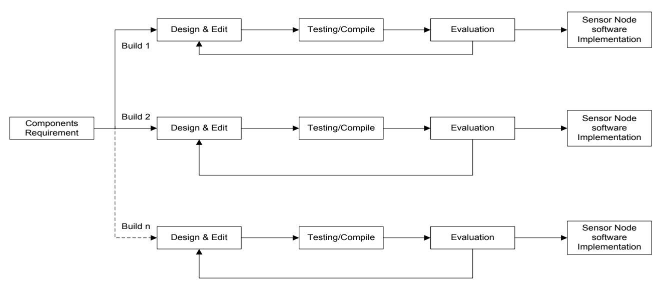
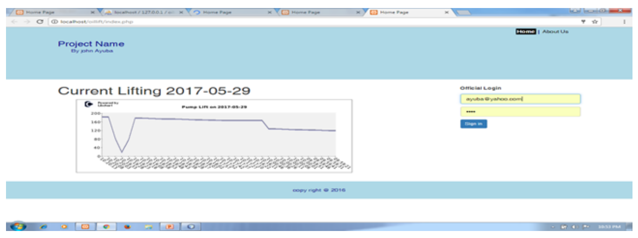


Figure 1: Block diagram for the sensor nodes software development.



Current Lifting 2017-05-29

Figure 2: The home page of the web design showing the current lifting of oil in real time.

When taken an integrated sensor nodes software development into account, the issues mentioned above can affect the software architecture of the wireless sensor networks [13]. The architecture proposed in Figure 3 uses clustering to handle the issues and provides a good application quality of service (QoS) management. A cluster base wireless sensor networks is a set of adjacent sensors which are grouped together and interfaced with the rest of the network through a gateway or cluster head [14]. Gateways are high energy nodes which maintain the network in the cluster, perform data aggregation and organize sensors into subsets [16-18] (Figures 3 and 4).

The Architecture of A Sensor Node

The architecture of the sensor node consists of a pumping machine, flow sensor, microcontroller, communication module and a memory. The pumping machine pumps the oil and gas; then a flow sensor monitors the flow rate and remotely reports the data through the communication module after being processed by the central unit. The resulting data is stored on the database server in form of real time data which can easily be accessed in order to know the amount of oil and gas being lifted without seeking to collect the data from the several stations manually (Figure 5).

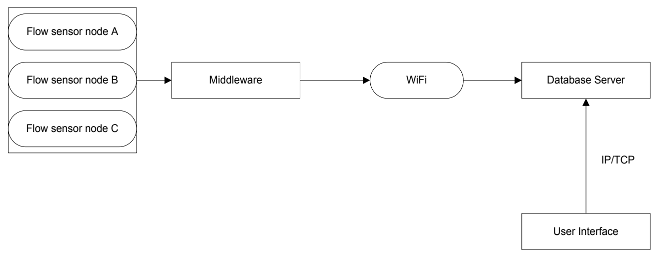


Figure 3: The proposed wireless sensor network software architecture.

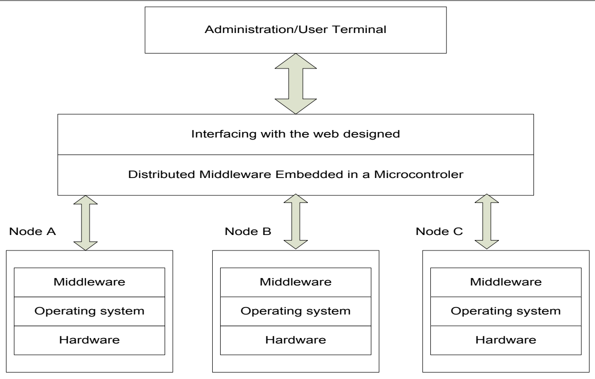


Figure 4: The proposed cluster based wireless sensor networks software architecture.

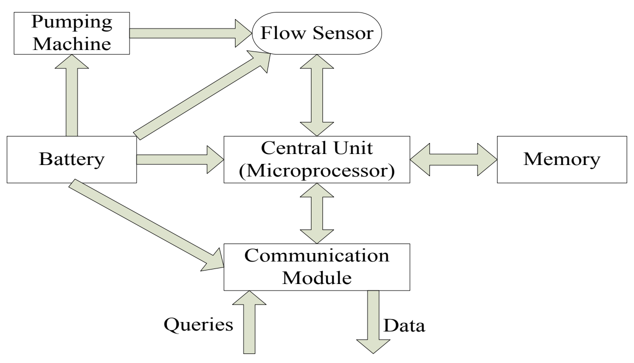


Figure 5: The architecture of a sensor node.

Conclusion

In this research paper, the software development life cycle (iterative incremental model) involved in the design and implementation processes for the system prototype of the integrated wireless sensor network for real-time remote monitoring of oil and gas flow rate metering infrastructure was provided. Complete software architecture for the wireless sensor networks was also developed. The first proposed architecture specifically illustrates the behavior of a single node sensor network application while the second proposed architecture uses clustering to handle the effects of issues from the previous one, thus, providing good application quality of service (QoS) management.

References

1. Ayuba. J, Igimoh J (2017) The Design of Wireless Sensor Network for Real Time Remote Monitoring of Oil & Gas Flow Rate Metering Infrastructure. IJSR 6: 425-429.
2. Beck K (2001) Extreme Programming Explained. Addison-Wesley, Reading, MA.
3. Cockburn A (2002) Agile Software Development. Pearson Education, UK.
4. Ezeigbo C (2014) Sustaining the Debate for Improved Transparency in Nigeria's Oil and Gas Sector. Nigeria Natural Resource Charter.
5. Garuba D (2015) An Oil Sector Reform Agenda for the Buhari Administration by Dauda Garuba. Premium Times.
6. Gilb T, Susannah (1988) Principles of Software Engineering Management. Addison Wesley, US. pp: 1-30.
7. Greer D, Ruhe G (2004) Software Release Planning: An Evolutionary and Iterative Approach. Information and software technology 46: 243-253.
8. Greer D, Bustard D, Sunazuka T (1999) Prioritization of System Changes using Cost-Benefit and Risk Assessments. Fourth IEEE, pp: 180-187.
9. Henkel J (2016) Software Architecture Design of Wireless Sensor Networks.
10. Reza Akhondi M, Talevski A, Carlsen S, Petersen S (2010) Applications of Wireless Sensor Networks in the Oil, Gas and Resources Industries. 24th IEEE International Conference on Advanced Information Networking and Applications, pp: 941-948.
11. Ruparelia NB (2010) Software Development Lifecycle Models. ACM SIGSOFT Software Engineering Notes 35: 8-13.
12. Sobral T (2012) Wireless Sensor Network for Oil & Gas Industry.
13. Sohraby K, Minoli D, Znati T (2007) Wireless Sensor Networks: Technology, Protocols, and Applications. John Wiley & Sons, US.
14. Blumenthal J, Handy M, Golatowski F, Hasse M, Timmermann D (2003) Wireless Sensor Networks New Challenges in Software Engineering. Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference 1: 16-19.
15. Hill Jason, Szewczyk R, Woo A, Hollar S, Culler D, et al. (2000) System Architecture Directions for Networked Sensors. ASPLOS, pp: 1-18.
16. Yu Y, Krishnamachari B, Prasanna VK (2004) Issues in Designing Middleware for Wireless Sensor Networks. IEEE Network 18: 15- 21.
17. Gupta G, Younis M (2003) Fault-Tolerant Clustering of Wireless Sensor Networks. Wireless Communications and Networking, 2003. WCNC 2003. IEEE 3: 1579-1584.
18. Ayuba J, Igimoh JA (2017) Implementation of Wireless Sensor Networks for Real Time Monitoring of Oil and Gas Metering Infrastructure. SCIRJ.