VIA University College

Project Report

Group

Pawel Adam Kramarz (189614)
Titas Urbonas (189653)

Supervisor

Erland Ketil Larsen

VIA University College
ICT Engineering

Chr. M. Østergårds Vej 4
8700 Horsens

December 15th, 2015

Contents

Table of figures

Abstract

A problem was found in the "elcon a/s" which is using an older Machine Park for the production. To make the park more efficiently, we design a monitoring solution for maintenance. The system contains 3 parts – embedded system, database, windows application, which is including hardware and software solutions. The core point of the system is the embedded system which is connecting to the machine with protocols (eg. USS, Modbus, H) and sending the data through the serial connection to the database. The system was created specifically for one Machine Park, but with a little effort, it could be expanded to use globally.

1 Introduction

The aim of this project is to create maintenance and monitoring system for "elcon", which could ease the workflow for the technicians and to raise the productivity of the company. The requirements for the system will be based on an interview with the technician and the manager in elcon.

Previously, the company maintained machine when they had a breakdown and that caused loss a lot of productivity. With this system, company will be able to predict the breakdowns, maintain it in advance and monitor workload for the day also for the working period.

The aim of this project report is to provide an insight into the creation of a complete maintenance and monitoring system, based on a set of requirements. The process is split into four major phases: analysis, design, implementation and testing.

Short introduction of the "elcon a/s":

The company is located in Horsens it is a medium size company. The company makes PCB (Printed circuit board) boards mostly for Denmark's and Germany's companies.

As every company also "elcon a/s" is structured with departments, which handle orders just in time.

Orders from customer get to the salesperson by email. The specification is typed into the CRM (Customer Relationship Management) system; the Gerber data will be sent to the CAM (Computer aided manufacturing) department. The CAM has to check the ability to production of the data –it will be called DFM checks. Then the data is stepping up to the customer panel and the production panel. Finally, different formats are exported for different machines. In case when the Gerber data are not repairable or some inconsistency exists, a query is returned to the salesperson, which forwards the query to his customer.

Data ready-for-production is immediately used on the plot-machine to plot films, which will be used for expose. In parallel, the drill data are used to drill the panels. Panels with holes will be metal-plated in a shadow process. Then a resist foil will be applied to the panels, which will be exposed through the pattern-film. During the resist stripping some parts of the resist-foil, which has not been exposed, are washed out.

The next step is the MP-line (Metal plating), where orange-blue panels get copper onto resist-free areas in electrolysis process, which also ends with tin applying. The panels get nice blue-silver color. The tin has to protect the copper against etching in the next step. But before the etching the resist must be completely washed out from the panels, and exposed the copper to the etch fluid. Panels are orange-silver before etching. The speed which the panels are going through the bath, decides about the etch-depth of the copper clad used for the raw material.

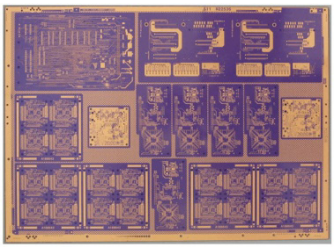


Figure 1 - Production panel with resist

As a supplement to the electrical test, which will follow later in the flow, an AOI test can be carried out just after the etching process. The purpose of this test is to check the differences between the image-data with the real image of the panels. Any short or break in the layout will be observed with the AOI machinery.

Here we can talk about fully functional PCB board. But the board is still far away from to be used by the customer. It needs a protective surface. For non-solder places on the board a green (black, red, white) solder mask will be applied. This is done when the panels are totally coated and exposed through mask films. Finally, the non-exposed places are washed out, and solder pads are ready to get the protective surface as gold, silver or tin. Depends on the surface the panels look green-gold, green-silver or green-glossy silver (tin).

If desired a PCB can be equipped with silkscreen. A silkscreen machine will draw the image of components from the data prepared in the CAM department. As the image is not dry, it must be UV (ultra-violet) exposed before surface treatment.

Until now, the panels have not been tested electrically, but they will be, after routing. Here the production panels are routed out for small size customer panels. Every PCB holds on break-tabs in the customer frame. Every frame has some fiducial marks and tooling holes, which are used for mounting and soldering machine.

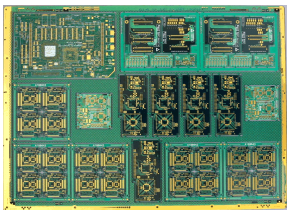


Figure 2 - Production panel completed

Customer panels are better handled on the flying probe tester. However, small pieces used for prototypes are preferred to be tested on big panels. Although the AOI is an image test, the flying probe tester checks the PCB electrical. Any short or break makes the PCB unusable for the customer and must be thrown out.

The final inspection, which is a manual process, rounds up the production process. The PCBs are ready now for delivery.

The flow in the form of rich-pictures is to be found in the analyze part.

2 Analysis

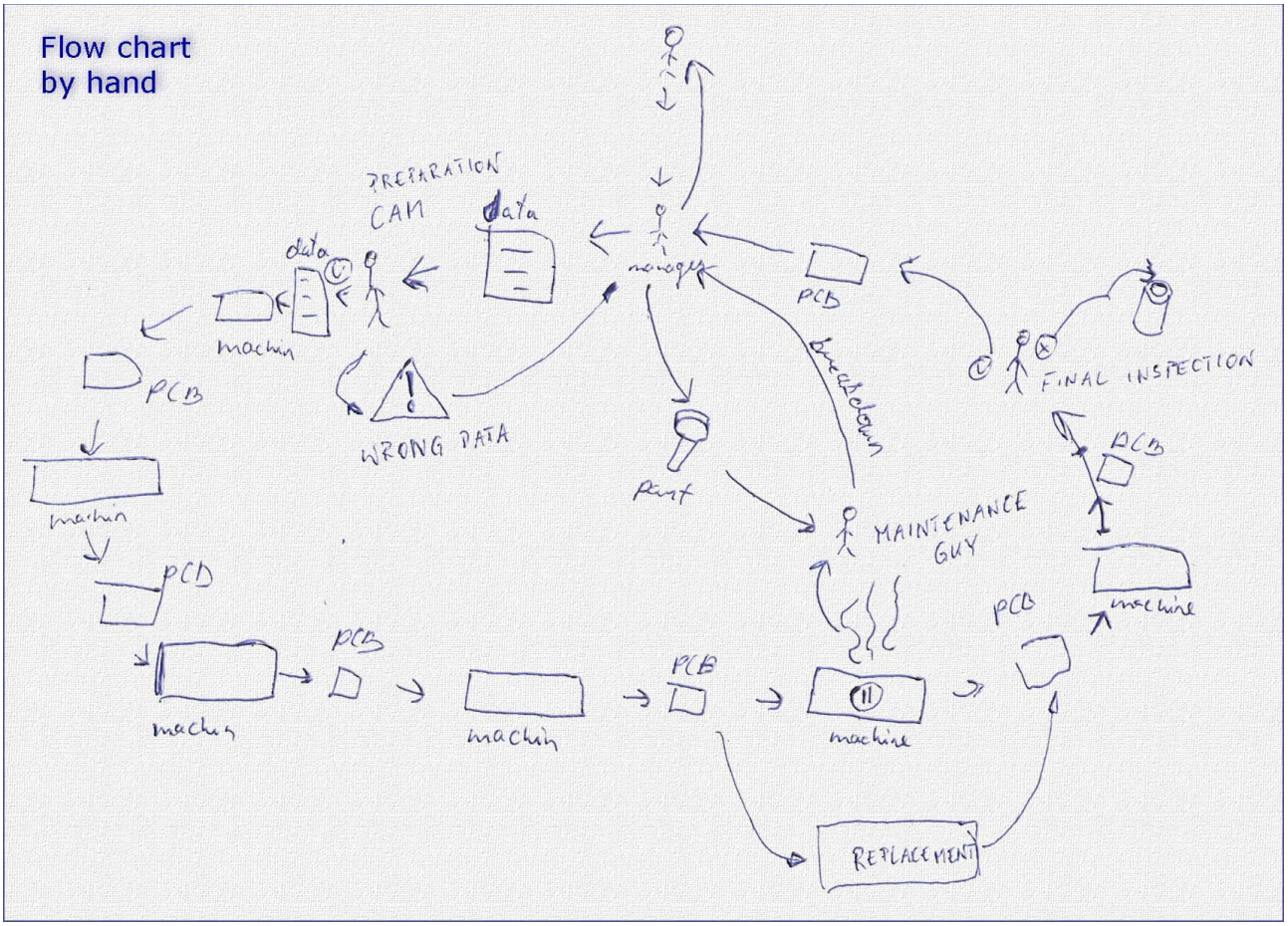


Figure 3 - Rich picture

By analyzing workflow in the company the rich picture (Figure 3 - Rich picture) has been made. In the rich picture it is possible to see the problem which could be solved. The problem is when one of the machines break down the production line is paused and if there is no replacement for the machine which braked down the production line is stopped till when it is fixed.

After understanding the problem of the work flow, there is a necessity to make usecase's based on the problem and the requirements of the staff.

2.1 Usecase diagram

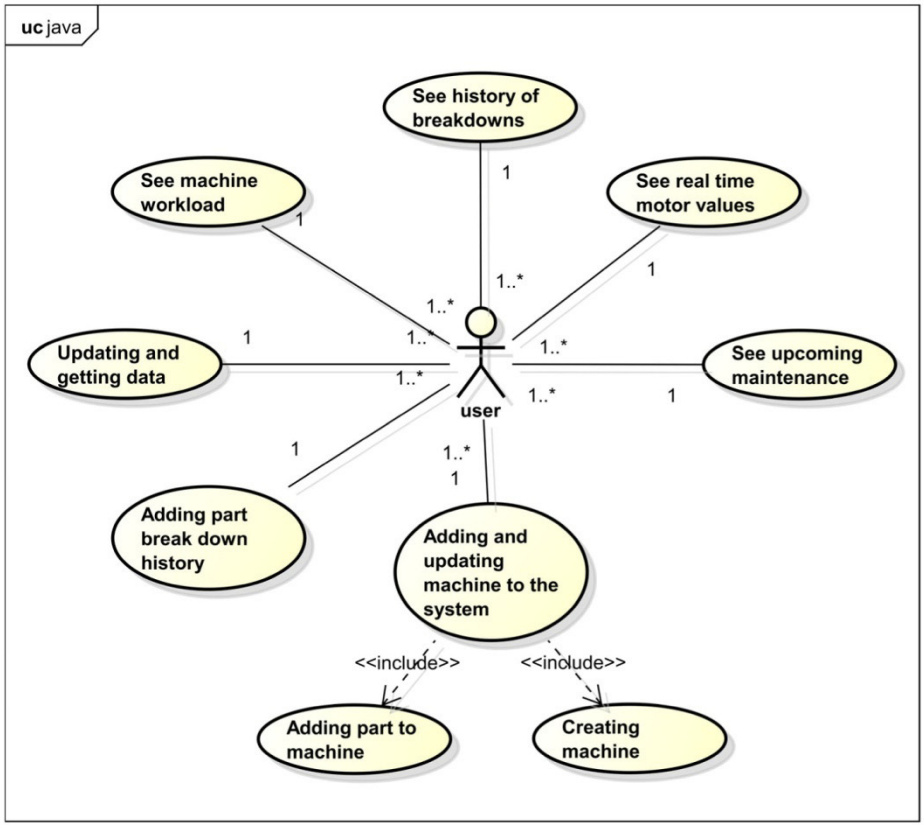


Figure 4 - Usecase

By pinpointing the most important requirements form the interview with the company a usecase diagram (Figure 4 - Usecase) can be created. In the diagram it is possible to see all the functionalities of the system.

2.2 Usecase summary

The following are short summaries of the most important actions based on the usecase diagram.

1. See machine workload: The user will see all machine names and their produced panels in given time.
2. See history of breakdowns: The user will see all machines which had their parts changed and the date when there were changed.
3. See real-time motor values: The user will see the latest values of the machine motors (eg. Frequency, Motor RPM (Revolutions per minute), Motor torque in the present) and the time when they were updated. Updates every few seconds.
4. See upcoming maintenance: The user will see upcoming parts of the machines which should break and the date when it will should happen
5. Adding and Updating machine to the system: When the machine is connected to the system user will have to insert data for the machine to the system.

6. Adding part breakdown history: When a breakdown occurs in the machine the user will have to insert the date into the system which machine and which part was changed after fixing the breakdown.
7. Get real time values: every few second machine is asked to update the real time values for the motor.
8. Get status form sensors: the user will see the specific module of the machine which is working on.

2.3 Usecase Description

See machine workload 2015/12/12

ITEM	VALUE
UseCase	See machine workload
Summary	Possibility to check total and daily work load for all machines.
Actor	user
Precondition	There has to be connection to the database.
Postcondition	List of workloads will be displayed.
Base Sequence	1. User will open the application. 2. User will have a choice to pick daily workload or total workload. 3. The list of all the machines with total workload or daily workload are displayed in one of the tables.
Branch Sequence	
Exception Sequence	If there are no data in the table the database is not connected or there is no data in database.
Sub UseCase	
Note	

Figure 5 - Usecase description

The usecase and summary provides a general idea of what the functions of the system it will be. To further elaborate on this, a usecase description (Figure 5 - Usecase description) is made, which will show more precisely how the function will be implemented.

2.4 Requirements

Based on the problem, equipment that the company uses and the requirements of the usecase diagram the functional and nonfunctional requirements were made. The keystone of the requirements was that the machine cannot connect directly to the companies server, which is used for storing data for the PCB production, based on this requirement the decision was made to include a middle layer between machine and the server which is called in this report embedded board - (Ethernut).

Functional requirements

1. The machines can send status to the server
2. The embedded board can read the status of the machines
3. The embedded board can send the status of the machines to database server
4. The embedded board can see changes of the machines sensors
5. The embedded board can send changes of the machines sensors to database server
6. The database server can add time-stamp to status and sensor change entrance
7. The user application can show the status of all machines that are connected
8. The user application can show the workload of current day
9. The user application can show the work load of the period
10. The user application can show the breakdown history of specific machines
11. The user application can add machines and parts for the machines
12. The user application can add history
13. The user application can show the prediction of upcoming breakdowns
14. The user application can automatically update date from the machines

Nonfunctional requirements

1. The system must use embedded board between machine and server
2. The database server must run on Linux platform

Brief description of the requirements:

To get the requirement the proper cabling is needed. Furthermore, some settings are necessary in machine registers. With a proper RS485 connection and wrong settings of the machine controller (MicroMaster) will not respond any telegram. A signal converter is needed to connect both the machine controller and the embedded board. It will call it an RS485 to RS232 converter.

To get the status of the machine controller as in requirement 2 a special telegram must be prepared. Using a wrong telegram the machine controller will not respond at all.

As mentioned in the req. 3 the forwarding happens in the software of the embedded board. When the data arrive on one serial port, then they will be reformatted and finally send through the second serial port to the server application.

Requirement 4 assumes that some sensors will be attached to some ports of the embedded board. There are different sensors installed in the machine and different ways to read their status: SPI or

TwoWire protocol. Also a simple PIN reading could be enough in some cases. Four examples of sensors in (Figure 6 - Sensors). The proximity switch, the flow switch and the reed switch are ON-OFF switches, and can be connected through the opto-isolators to the input lines of the embedded board.

The thermocouple is a temperature sensor which changes its resistance with increasing/decreasing of temperature. For the thermocouple an analog-digital converter (PORTF on embedded board) have to be used, which must also be implemented in the code.

The DB-service is a program running on the server machine that read continuously the data on the serial port. When the data arrive it is important to parse them accordingly to the content format with them a SQL statement and finally save them onto the database

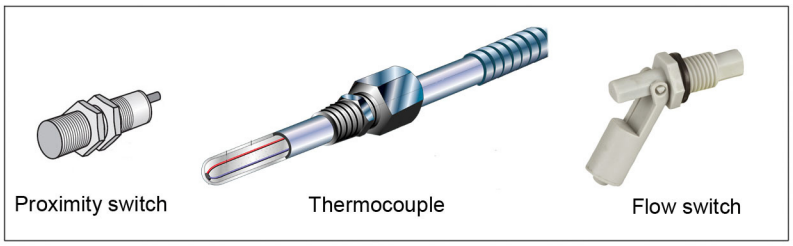


Figure 6 - Sensors

2.5 Domain Class model

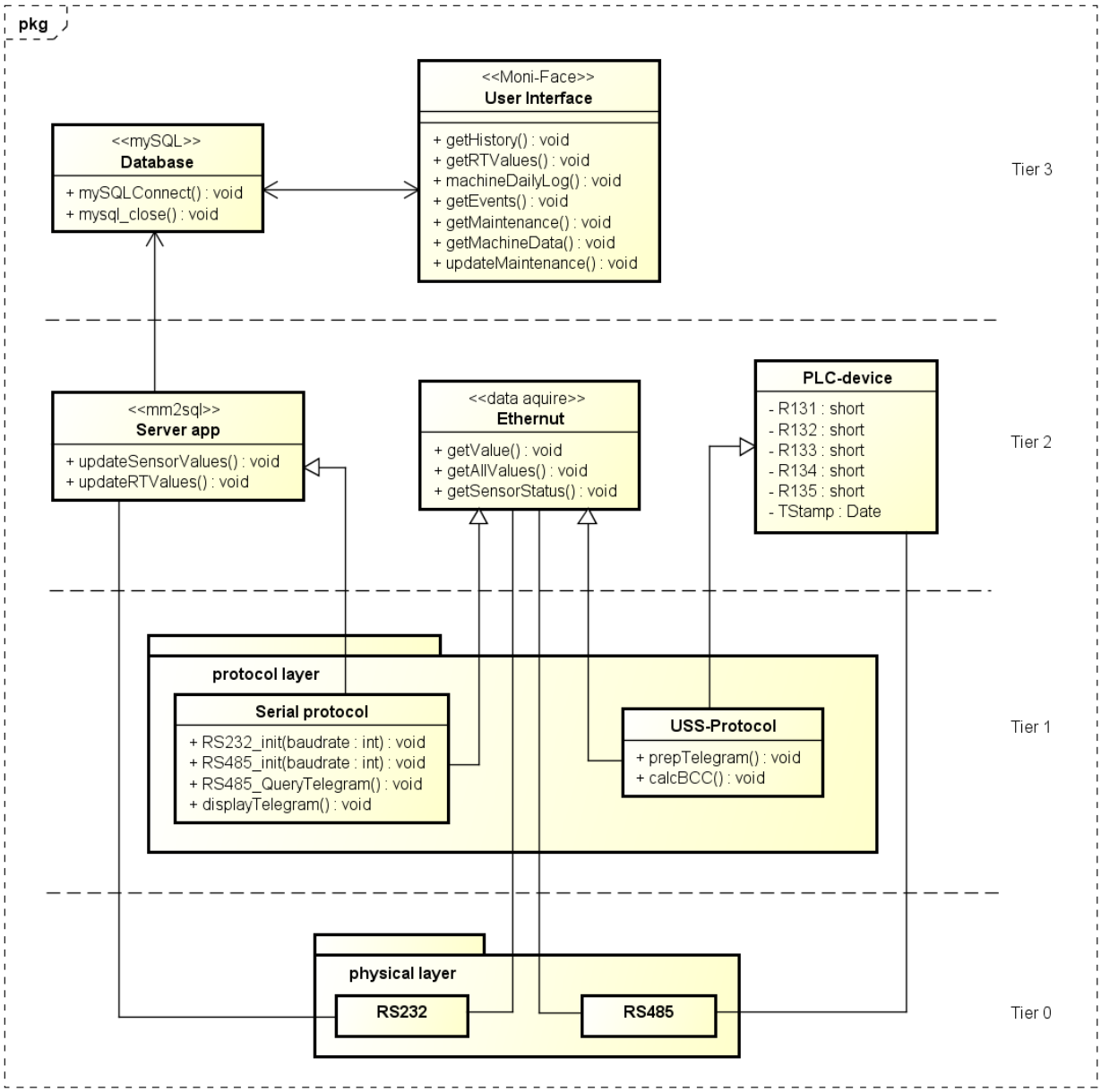


Figure 7 - Domain Class model

After all the requirements and usecase's the domain class model were made (Figure 7 - Domain Class model) it is an early version of the model part of the system. This model shows the basic architecture of the system, how the separate parts are related. It is a draft version of the class diagram.

3 Design

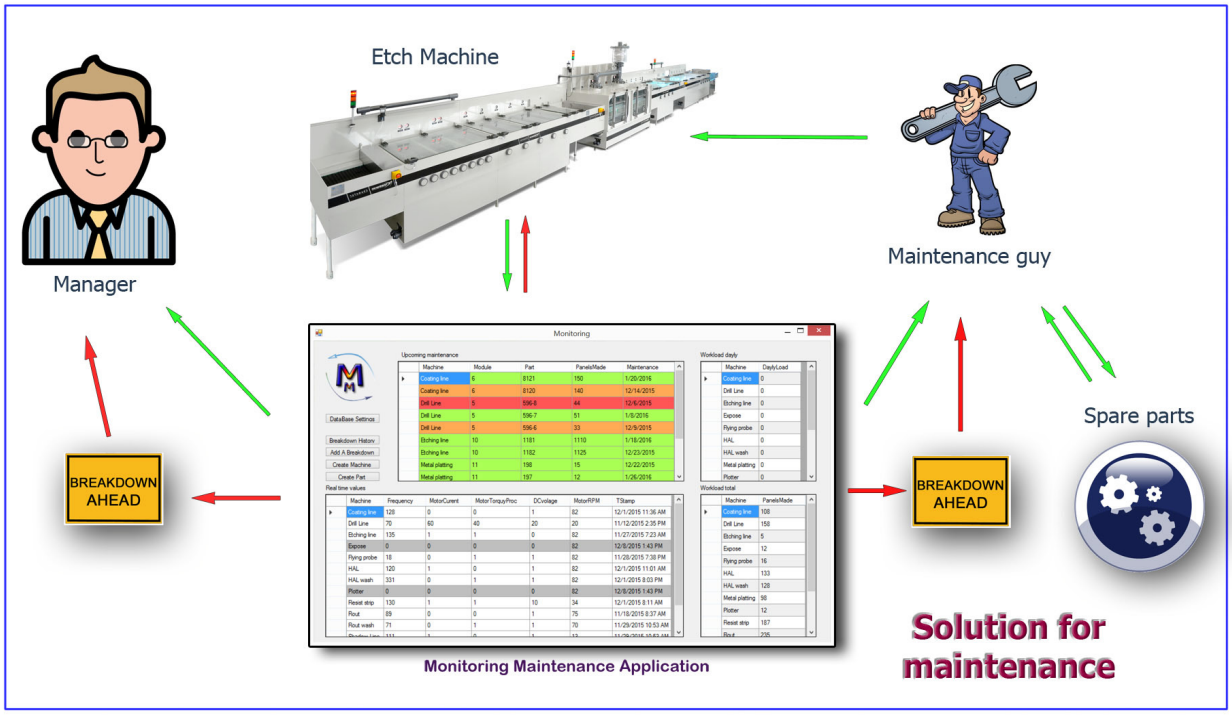


Figure 8 - Rich picture for Solution of maintenance

After the analyzing phase was finished, the system moved to design phase. From the requirements and usecase the system architecture could be made. In order to visualize the solution for the problem the rich picture (Figure 8 - Rich picture for Solution of maintenance).

3.1 Block model system

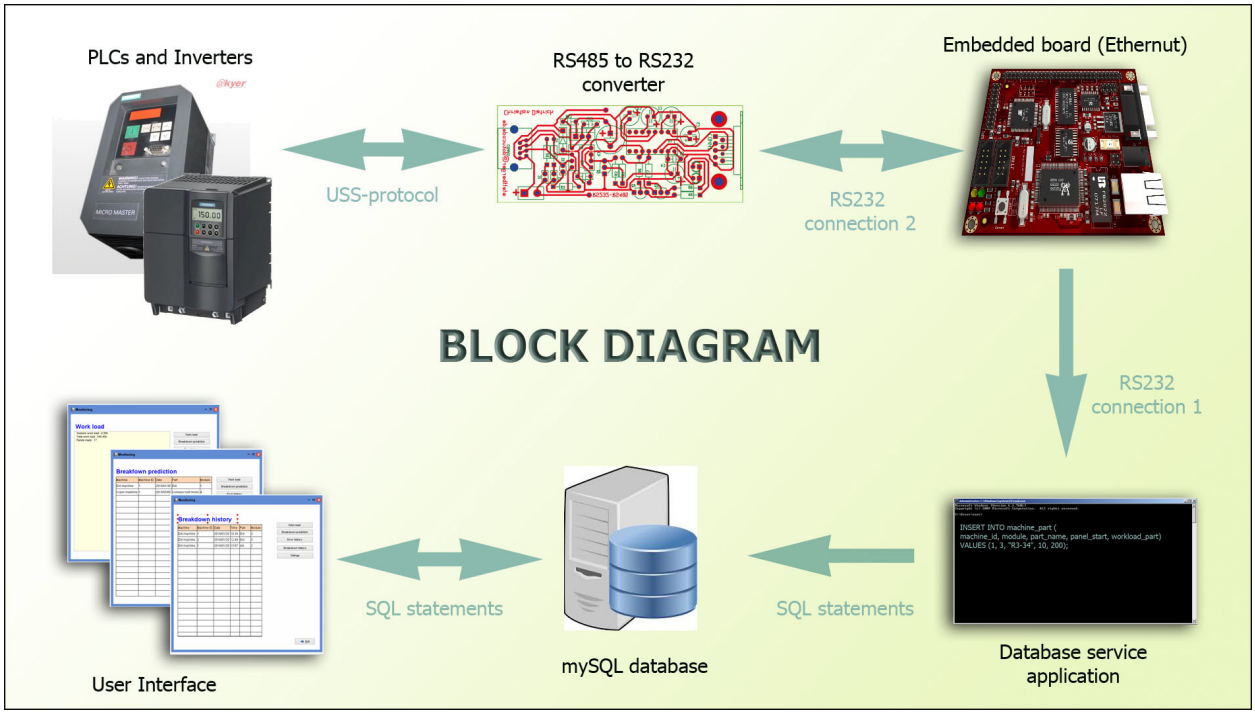


Figure 9 - Block diagram of system

The block diagram (Figure 9 - Block diagram of system) shows the architecture of the system and the sequence for this system is:

1. The embedded board sends a message through the converter to the PLC or inverter.
2. The machine controller (PLC or inverter) sends back through the convert to the embedded board the message with data.
3. The embedded board sends through RS232 connection the data to the database service application.
4. The database service application adds data to the database server and including the data-stamp.
5. User interface gets the data and displays it.

More than one machines connected:

The embedded board will send the message to the controller and receives an answer. After the receiving the message the embedded board will convert the data and send to the database application. This process repeats for all the machines.

3.2 Database design

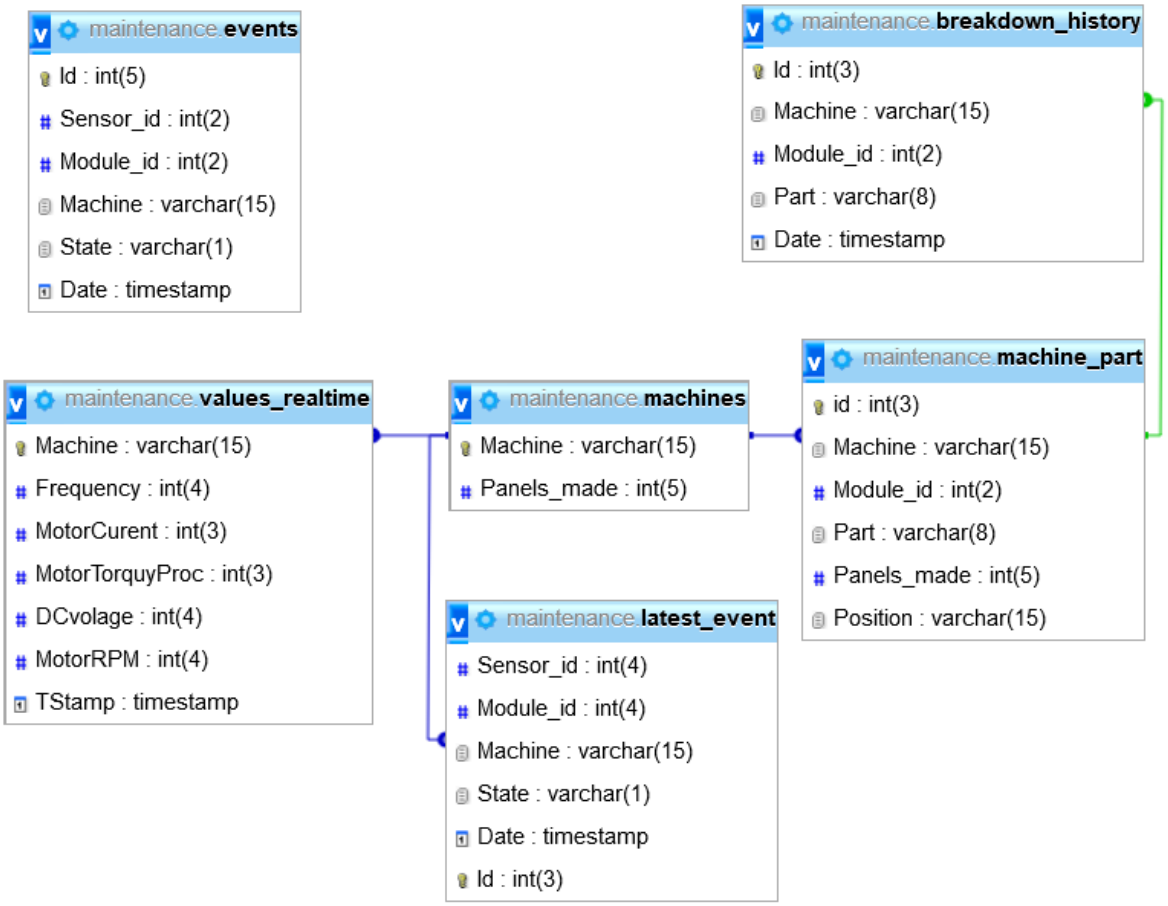


Figure 10 - Database design

After analyzing the data flow from the machines and what data will be needed to store from the application the design of a database (Figure 10 - Database design) was created. Based on the flow of data the decision was made that machine will have access to the events and values_realtime tables and the application has access to all the tables. The "machines" table is the main table form which is possible to connect to all other tables except the events table. Other tables are filled by calculating form events table.

Table:

1. Events: the table is used for storing the data from the machine sensor this table is used for calculation of the machine workload(how much panels were made).
2. Values_realtime: the table is used for storing the data of the machine motor values used for showing the park overview which machine is working and which is on the standby mode.
3. Machines: table is used for storing the data for the machine name and how many panels the machine made in total.

4. Latest_events: the table is used for storing one row of the events. The purpose of this table is to remember which row was the last one for the events table. Used for when the main application shutdowns to remember till which row all data was calculated.
5. Machine_parts: the table is used for storing all parts of the machines which will be added from the machine. Also the data will by change on the run time for the amount of panels it made.
6. Breakdown_history: the table is used for storing all parts that was change when the software was used.

3.3 Class diagrams

With class diagrams it is possible to see all system. The purpose of the class diagrams in this project is to see the overview of the different parts of the system.

3.3.1 Class diagram application

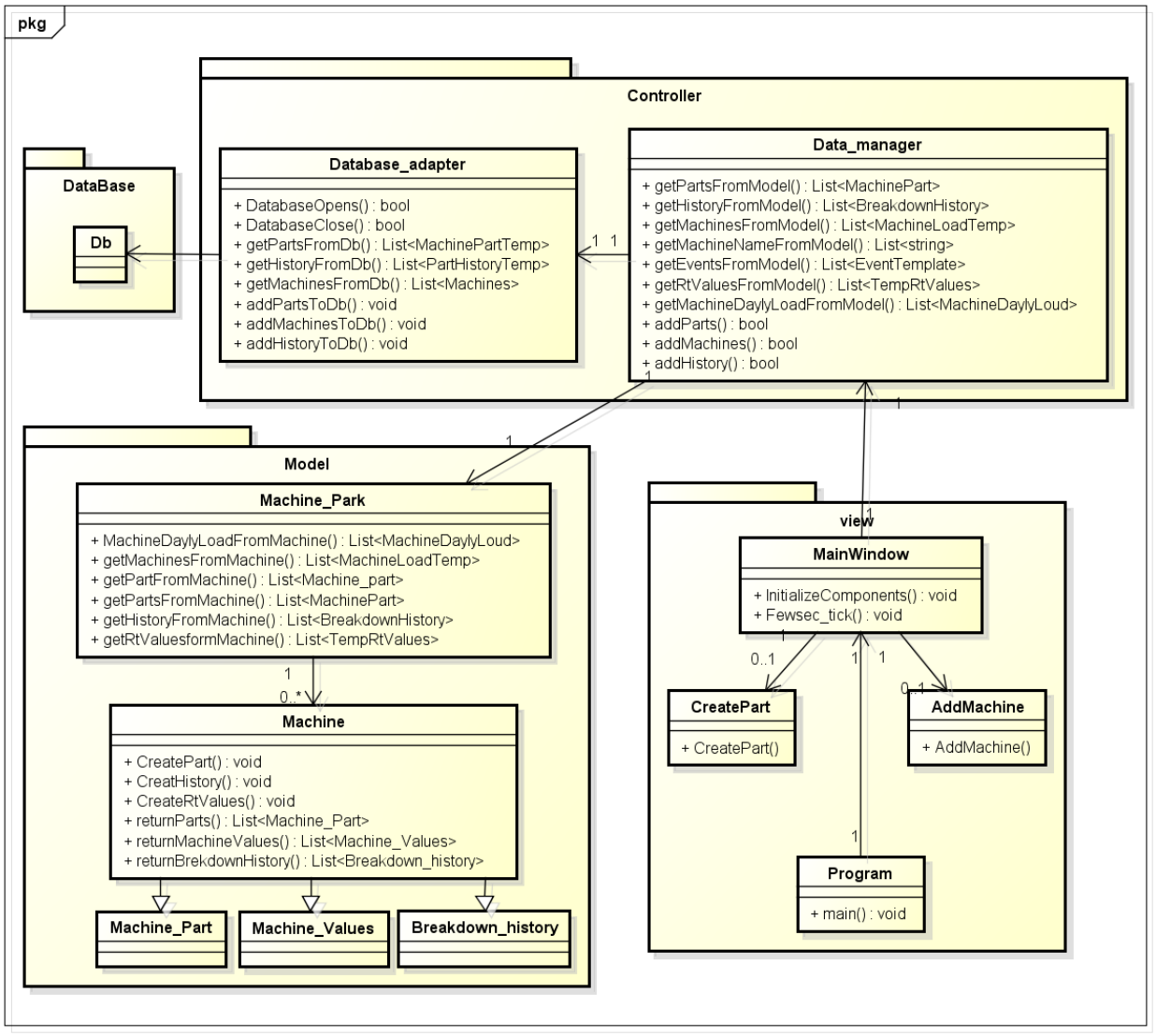


Figure 11 - Class diagram skeleton

The class diagram skeleton (Figure 11 - Class diagram skeleton) for the application is based on pattern MVC (Model View Controller) the full diagram could be found here (Class diagram application).

For this application there are 3 main packages and one additional which is not shown in the diagram. Not shown package is called “ObjectTemplates” it is used for transferring data objects in a different way than it is stored in the data model.

The main packages are “Model”, “View” and “Controller” which are the MVC pattern. The “Model” package is used for the storage of the data in other words dataset. The “View” package is for a UI (User interface) and the “Controller” package is controlling all data flow form database and data from the model to the UI.

3.3.2 Class diagram RS485 communication

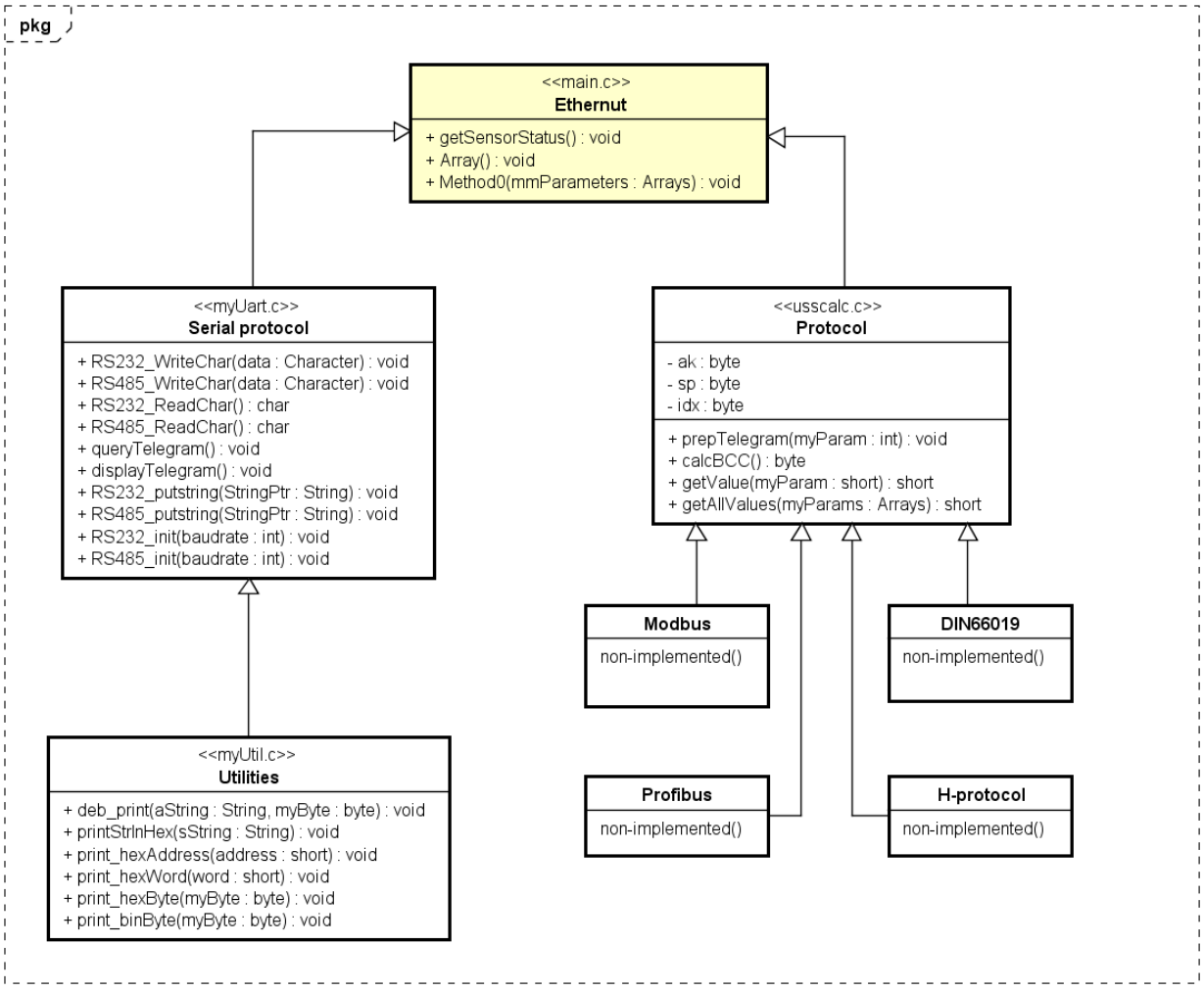


Figure 12 - Class diagram for RS485 communication

In the (Figure 12 - Class diagram for RS485 communication) will be found a structure of the AVR-software for the Ethernut embedded board.

For communicating with the hardware device on software level a protocol is needed. Only USS-protocol has been implemented. The purpose of it is scalability model ensures that an implementing of missing protocols in the future can be done without influence on the other parts of the program. The utilities are used mostly for debugging purpose, but also for displaying information on the used LCD display.

3.3.3 Class diagram application for database

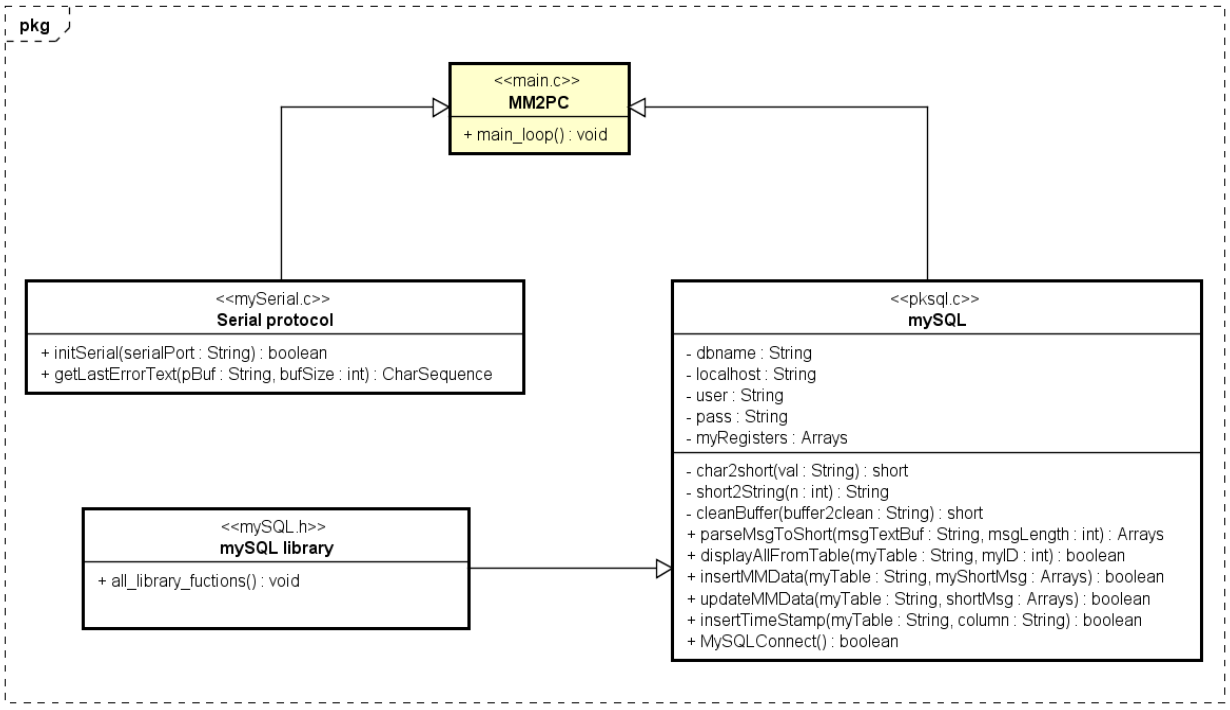


Figure 13 - Class diagram for database application

This diagram (Figure 13 - Class diagram for database application) shows the structure of the program with all functions used inside of it. The code is split into 4 parts:

1. Grab the message from serial port
2. Parse the message
3. Connect to the database
4. Insert data into the database

Check in 4.1.1 for description for every part.

3.4 Sequence diagram

From the class diagrams the sequence diagrams were made to show how data is flowing in each application.

3.4.1 Sequence diagram for update the data from database

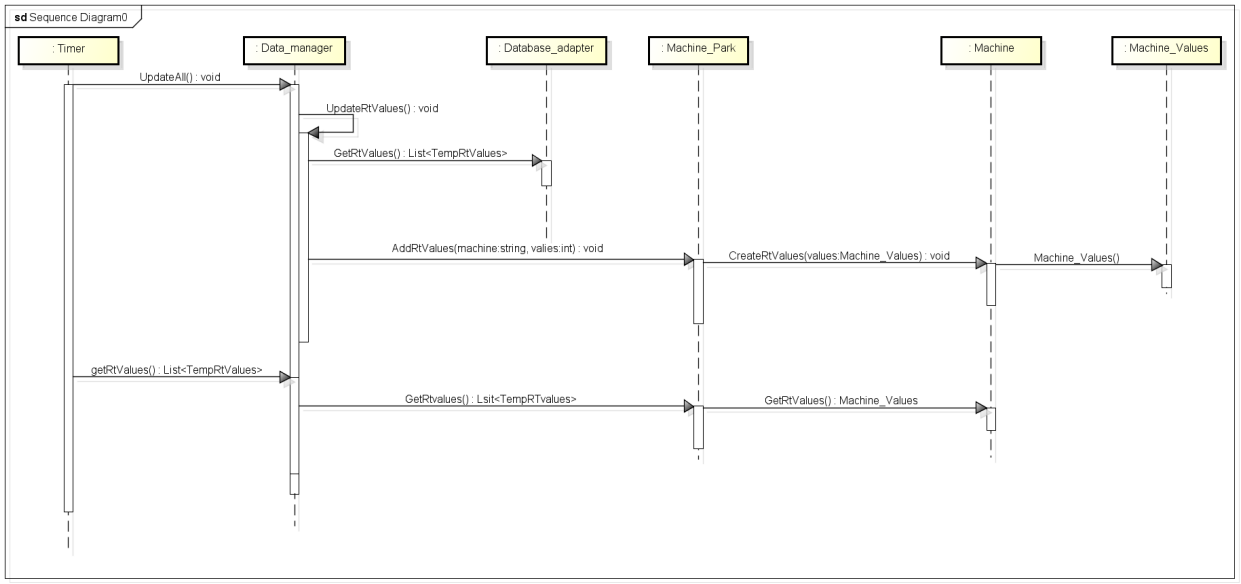


Figure 14 - Sequence diagram for real time values

The Sequence diagram (Figure 14 - Sequence diagram for real time values) represents the flow of the data in the main application for timer. The timer starts when the application starts and it is responsible for updating all datasets in the model every 5 seconds from database. The diagram only shows one part of dataset update there are three more data sets, but the sequence is the same as shown in the sequence diagram only the data sets are saved in a different object in the machine object.
The flow of the data:

1. 5 seconds pasts form the last update timer will activate with starting the process of updating.
2. The request goes to the control package to the Data_manager object into the `Updateall()` method.
3. In the update method there are four types of update machine, machine part, history and real time values. In the figure 18 shows update of the real time values.
4. The sequence continues by going to Database_adapter object into the `GetRTvalues()` which is connecting to the database and returning the list of the all machine real time values.
5. Then returned list is sent to the model package in Machine_park object into the `AddRtValues()` the method takes all the data and divides it for each machine and sends the Machine_Values object to the machine object.
6. Then the data flow to the method `set_Rtvalues()` which is setting the date in the object.
7. For each machine the flow form 5-6 is repeated.
8. For all other updates the sequence repeats 3-7, but the methods names are different.

9. When `UpdateAll()` finish its job the next step is to update the UI (user interface). In the (Figure 14 - Sequence diagram for real time values) it shows only the real times values.
10. The flow comes back to the timer. The request goes to the control package to the Data_manager object into the `getRtValues()` method.
11. The sequence continues by going to model package to Machine_park object into the `GetRtValues()` which is requesting real time date from all machine.
12. Then the request is sent to the Machine object into the `GetRtValues()` which is return the object.
13. The sequence repeats 11-12 for all the machines.
14. For each data set update the sequence repeats 10-13, but the methods names are different.
15. When the list in the Machine_park is ready it comes back to the timer.
16. The timer sets all data in the UI.

3.4.2 Sequence diagram embedded board

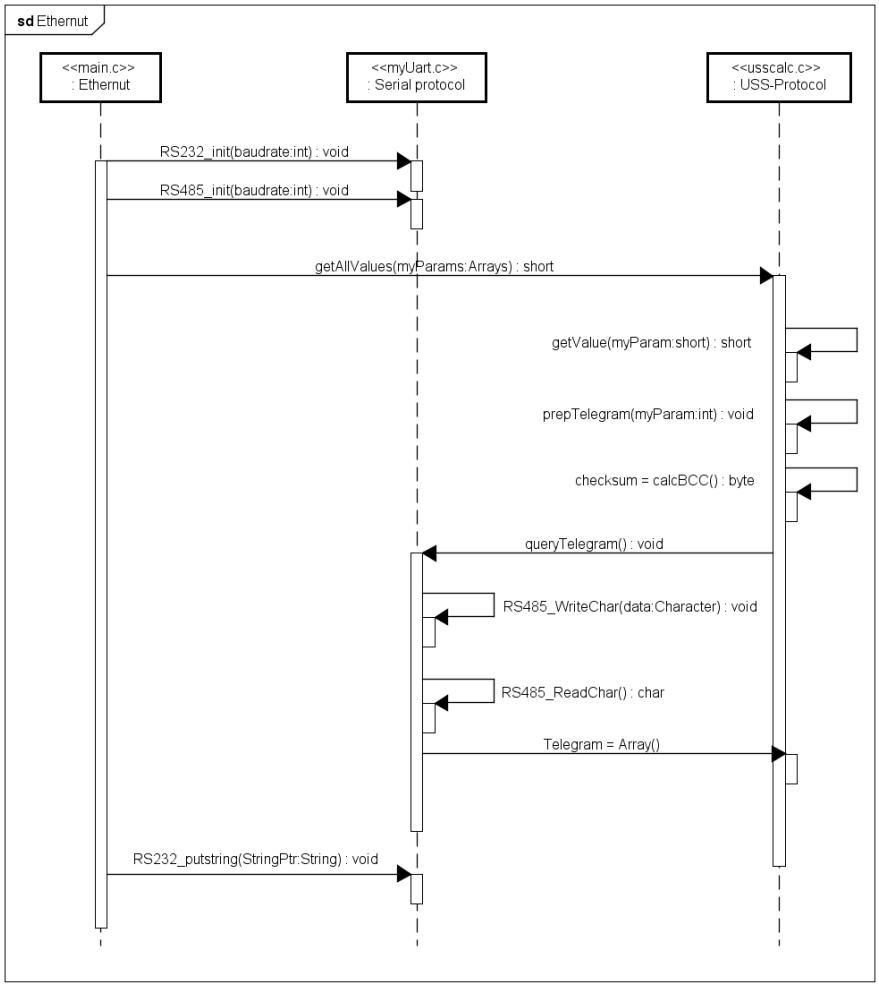


Figure 15 - Sequence diagram embedded board data flow

From the (Figure 15 - Sequence diagram embedded board data flow) it is possible to see the data flow from the PLC or Inverter. Steps of the sequence:

1. First the initialization for the serial ports is required.
2. The main function of embedded board is sending the request to USS protocol function.
3. USS protocol functions starting to make a telegram for the inverter.
4. Firs part of the preparation is finding the first register that the data will be needed.
5. Second part of the preparation is to make the right telegram to the machine controller.
6. Third part of the preparation is checking the sum of the telegram.
7. After these three parts the telegram is ready for sending.
8. The embedded board sends the telegram to the inverter.
9. The machine controller responses with telegram with the values.
10. The sequence repeats 4-10 for all required data.
11. Finally, when all data are ready it is sent to the database application.

3.4.3 Sequence diagram for database application

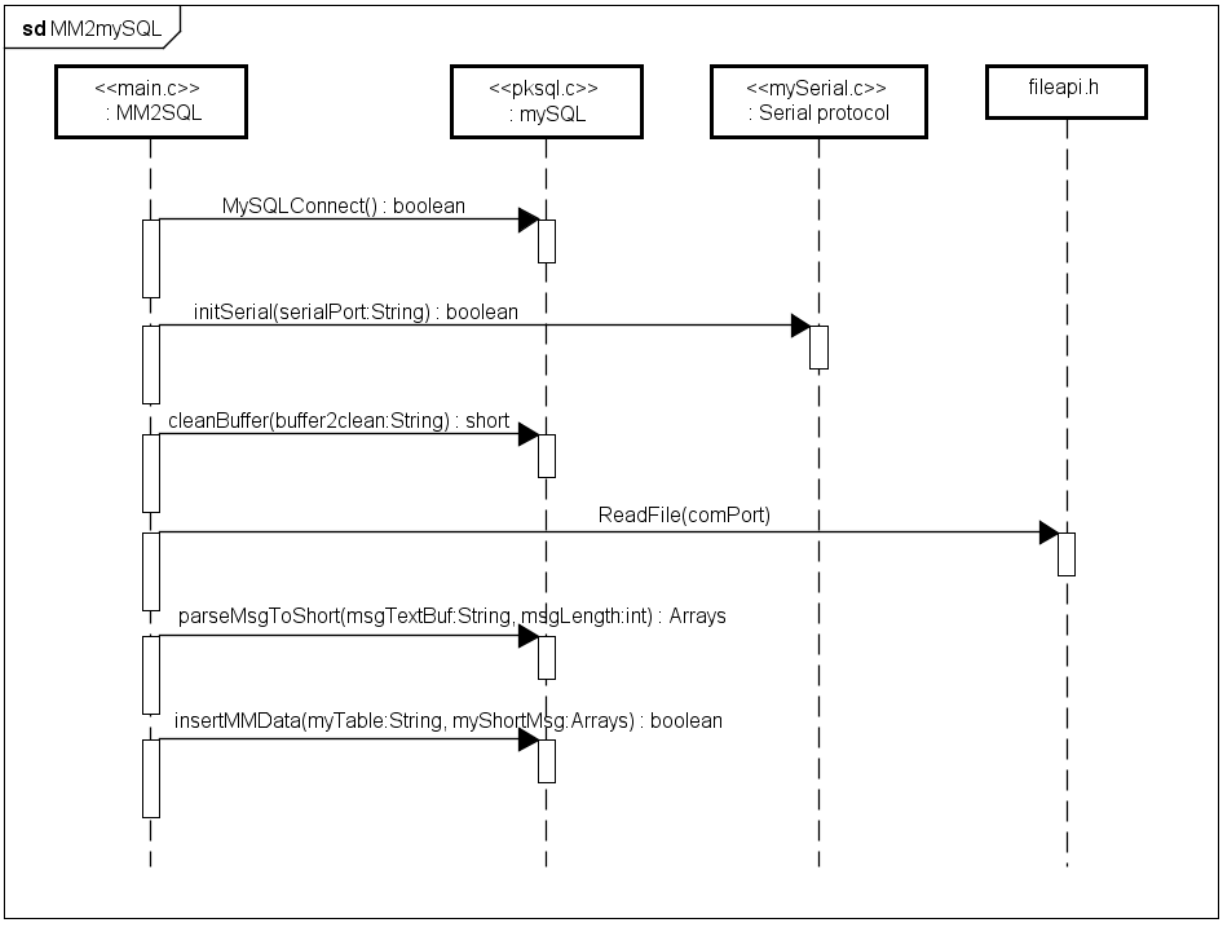


Figure 16 - Sequence diagram of database application

From the (Figure 16 - Sequence diagram of database application) it is possible to see the data flow of the received data from the embedded board. The flow of the sequence is:

1. The flow starts with the initialization of MySQL(`MySQLConnect()`) connection and the serial connection(`initSerial()`).
2. The buffer which is receiving data from the serial port will be cleaned (`cleanBuffer()`).
3. The next step of the flow is reading the data from the embedded board (`readFile()`).
4. Then the date is rephrased and assigned to specified types for the SQL statement (`parseMsgToShort()`)
5. The final step of the flow is inserting data from the machine controller to the database (`InsertMMData()`)

3.5 Activity diagram

Figure 17 - Create machine activity diagram

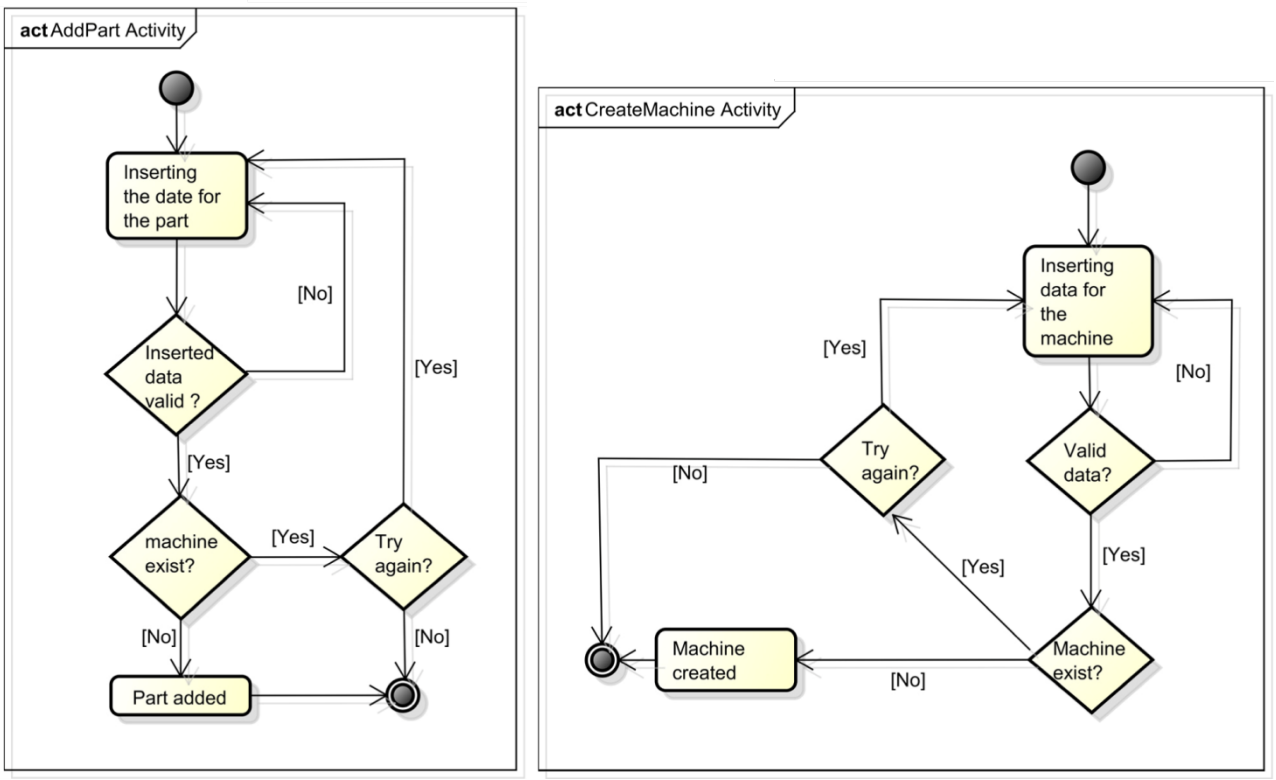


Figure 18 - Part activity diagram

From the usecase diagram few usecase's have Activity diagrams which are showing how the user interacts with the application other usecase's are automatically displayed on the main window.

4 Implementation

The system was developed with eclipse, MySQL and Visual Studio.

The application for a database and embedded board was developed with eclipse environment. It is important to start the environment from a Visual Studio Command Prompt as it sets properly the environmental variables needed to build the executable files.

The database was implemented on MySQL. The database type was chosen based on the group decision.

The main application was implemented in Visual Studio. The user interface was implemented in windows form technology.

The code could be found in the “github” link.

4.1 Software

4.1.1 Database implementation

The database where implemented in two parts. First part is for storing data from the machines and the second part is for storing data for calculated values of the upcoming maintenance and monitoring the machine park work load.

In the (Figure 10 - Database design) is possible to see the design of the database. From the design of the database, it is possible to see that “machine” table is the key table form which is possible to access all other data.

```
CREATE TABLE IF NOT EXISTS `machines` (
  `Machine` varchar(15) NOT NULL,
  `Panels_made` int(5) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

ALTER TABLE `machines`
  ADD PRIMARY KEY (`Machine`);
```

From the code above it is possible to the “Machine” table implementation. It contains a primary key in this particular table the “machine” column is unique and there is no need for new column and it is a keystone table. This table is accessed only by the main application

```
CREATE TABLE IF NOT EXISTS `machine_part` (
  `id` int(3) NOT NULL,
  `Machine` varchar(15) NOT NULL,
  `Module_id` int(2) NOT NULL,
  `Part` varchar(8) NOT NULL,
  `Panels_made` int(5) NOT NULL
)
ALTER TABLE `machine_part`
  ADD PRIMARY KEY (`id`),
  ADD KEY `machine_part_ibfk_1` (`Machine`);
ALTER TABLE `machine_part`
  MODIFY `id` int(3) NOT NULL AUTO_INCREMENT,AUTO_INCREMENT=17;
ALTER TABLE `machine_part`
  ADD CONSTRAINT `machine_part_ibfk_1` FOREIGN KEY (`Machine`) REFERENCES `machines` (`Machine`);
```

From the code above it is possible to see the “Machine_part” table implementation. It contains primary and foreign keys. The table “Latest_event”is implemented the same way and table “Breakdown_history” is implemented similar just the foreign key is pointing to the machine part table. The primary key is id, because there is no unique column in this table. The foreign key is pointing to the main table “Machine”, because the “Machine_part” extends the main table. The “Machine_part”, “Latest_event” and “Breakdown_history” is accessed only by the main application.

```
CREATE TABLE IF NOT EXISTS `events` (
  `Id` int(5) NOT NULL AUTO_INCREMENT,
  `Sensor_id` int(2) NOT NULL DEFAULT '0',
  `Module_id` int(2) NOT NULL DEFAULT '0',
  `Machine` varchar(15) NOT NULL DEFAULT '',
  `State` varchar(1) DEFAULT NULL,
  `Date` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`Id`),
)
```

From the code above it is possible to see the “Events” table. The table is for storing sensor date when and which sensors have changed its status. For the primary key there had to be a new column, because there were not unique columns. The Id column was made with auto increase. This table is accessed by main application which is displaying the data and the database application which are inserting the data.

```
CREATE TABLE IF NOT EXISTS `values_realtime` (
  `Machine` varchar(15) NOT NULL,
  `Frequency` int(4) NOT NULL,
  `MotorCurent` int(3) NOT NULL,
  `MotorTorquyProc` int(3) NOT NULL,
  `DCvolage` int(4) NOT NULL,
  `MotorRPM` int(4) NOT NULL,
  `TStamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`Machine`)
)
ALTER TABLE `values_realtime`
  ADD CONSTRAINT `values_realtime_ibfk_1` FOREIGN KEY (`Machine`) REFERENCES `machines` (`Machine`);
```

From the code above it is possible to the “Values realtime” table. The table is used for strong the values from the machine controller. The table contains both primary and foreign keys. The primary key is the “machine” it is a machine name the system will store only the latest changes of the machine controllers. The foreign is pointing to the main table “Machines”. The table will be accessed by database application for storing the data and main application for displaying the data.

4.1.2 Main application implementation

The application is implemented in C# (C sharp), the technology that application uses is windows forms, and the pattern of the system is MVC (Model View Controller). This application is for the displaying the data from the system and user interactions with the system. From the full class diagram link, it is possible to see the full application architecture. The application is separated into two parts the View and the DataSource the reason for it is the business logic.

The application starts from the main method in the program class and starts the GUI (graphical user interface). From this point the application is waiting for an event there is two types of events in this application. The first event is the timer which is responsible for updating data and uploading to the application. The second event is from the user interaction with the application, it responds to the button clicks.

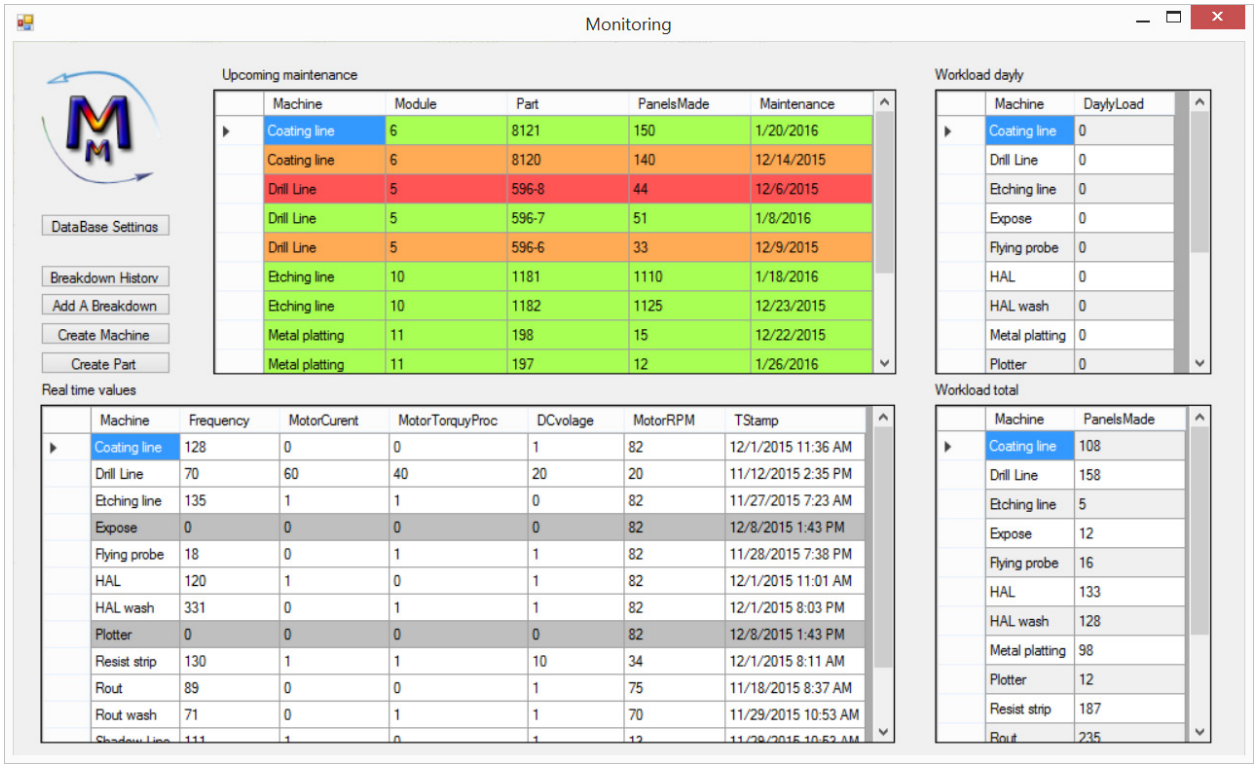


Figure 19 - Main application view

The GUI contains DataGridViews that are displaying data in the tables the GUI is (Figure 19 - Main application view). Also the DataGridViews are modified. The upcoming maintenance table is divided into three colors to signal status of the machine part. The green for the part is still good, the orange for the maintenance is close and the red color is signaling that the part has to be replaced as soon as possible. From the real time values table it is possible to see which machines are working, and which are not. The working machines are in white cells and not working machines are in gray cells. Other two tables are for daily and total workload of the machine.

Second part of the GUI is controls:

- Create Machine and Create Part buttons is for including the parameters of the newly connected machine.
- Add a Breakdown is used when application shows that a part needs maintenance and after maintaining the user has to insert data that it was maintain.
- Breakdown History is for checking all the breakdowns which were registered in the system
- DataBase Setting is for inserting the data of database manually, but it is automatically this button is used only when the connection fails.

The keystone of the program is the control package which contains two classes the Data_Manager and Database_Adapter. From the names it is possible to understand what the classes are doing. The Data_Manager is responsible for all the data flow from the model to the GUI and automatically updating all the data by creating Database_Adapter which is getting the data from the.

The last package is Model which is similar to the database model. The models main class is the Machine_park from which all the data is stored and also all the data is received for GUI.

Considerations:

The application is still in the implementation. Not all functionality is implemented in other words the application is still a skeleton. The functionalities like calculation of the daily load or maintenance calculations will be implemented in the next version of the project.

4.1.3 Application for database implementation

So far the embedded board contacts the machine controller for the real-time values of his registers and finally sends them to the application for database through the serial line of R232.

To update the mySQL database with fresh values it is necessary to introduce a new application, which runs on the server. Its task is to read the serial line, to take the incoming data, to format it (rephrasing) and to make a SQL-statement which will be executed onto the mySQL-server service. The mentioned application is a win32 console program and works together with mySQL library (libmysql.lib).

But before the update, some initialization takes place. `MySQLConnect()` and `initSerial(COMPORT)` are performed to connect to the database and to the serial port respectively. If an error occurs, the application exits with an error value.

The main application runs in the loop forever, where it listens for incoming data on the serial port.

```
	while (TRUE) {
		rc = ReadFile(comPort, msgTextBuf, sizeof(msgTextBuf), &bytesRead,
NULL);
		if (rc != FALSE && bytesRead != 0) {
			shortMsg = parseMsgToShort(msgTextBuf, bytesRead);
			rc = updateMMData(mce_table, shortMsg);
			if (rc == FALSE) {
				printf("Error while insertSQLData(). \n");
			} else {
				printf("MM-data successful updated. \n");
			}
		}
	}
```

Some characters of the incoming data are line controls and must be considered while parsing. In particular, these values are line feed, carriage return, space and colon. Pure values need to be converted from ASCII to binary form and saved into an array of shorts. All this is done in the function `parseMsgToShort()`.

Finally the SQL-statement is built inside of `insertMMData()` and the query is executed with a command `mysql_query()`. If the execution fails, the function returns `FALSE` and exits from the main loop. Otherwise, it returns `TRUE` and waits for the next message on the serial port.

Considerations:

Like in the embedded part, also the DB-app can be blocked forever, when no data arrive to the serial port. This issue will be solved by introducing the timeout variable and should be done in the next version of the program.

Furthermore, it will be necessary to program a Linux version of the existing Win32 DB-application, which should expand usability of different operating systems.

4.1.4 Embedded board software

The Embedded board project has 3 parts responsible for all functionality:

- Serial protocol
- USS protocol
- Utilities

Here is a brief description of the way how the software works.

As the communication with the machine controller and the SQL-application takes place by serial interface, both serial lines must be configured before use. The functions `RS232_init()` and `RS485_init()` are responsible for proper initialization of the serial interface. As a parameter has been introduced the speed-value, because both interfaces would work with different speed.

The main part works in an infinitive '`while`'-loop. The data from the machine controller are being acquired in the function `getAllValues()`, then the data are included with the register name with new line after the value and finally will be send trough the serial port to the SQL-Application.

`GetAllValues()` calls one-by-one the function `getValue()` for each register, and the amount of registers does not matter.

```
voidgetAllValues(uint16_t* mmParametersArr, uint8_t length) {
	uint8_t i;
	for (i = 0; i < length; i++) {
		// wait();
		mmParametersArr[i] = getValue(mmParametersArr[i]);
	}
}
```

This function has to return a value of the called register from the machine controller. To achieve this, a proper telegram must be prepared before sending a query. `PrepTelegram()` prepares an array of values to be sent through the RS485 line. The length of telegram with parameters (PKE, PZD) and BCC code are critical when creating it. As the reader may be interested in the details about the way of preparing, in particular about the USS-protocol, the best recommendation is the document which can be found in 4.1.5.

When the telegram is ready to be sent, the function `RS485_QueryTelegram()` takes and sends it on the serial line using the function `RS485_WriteChar()`. As the length of the telegram is fixed, there is no need to introduce a length variable separately, thus the length of 14 bytes has been hard coded here. Immediately after sending, the receiving buffer is cleaned and the function `RS485_ReadChar()`

try to read the values from the machine controller. The `RS485_QueryTelegram()` returns back with the return-telegram, which is stored into the array of machine controller parameters.

The Utilities part has been introduced mostly for debug purposes and expands possibilities for quick handling of bugs in the code on the terminal window. The response from the board is coming onto the terminal application (version 1.9b by Br@y) connected through the serial port with an RS232 to Serial converter. The function `print_hexWord()` was used mostly during the development process. Working with bits was the `print_binByte()` invaluable.

The embedded board also includes an LCD-display to show variables. In case of connection to both serial ports has been made (one for the MM75 and another for the PC) there is no possible to show anything on the terminal window. At time of delivery the project to the customer, the functionality of the LCD will be changed. It will show the status of working program (RUN or FAIL).

Considerations:

The serial function `RS485_ReadChar()` and the `RS485_WriteChar()` is blocking the microcontroller:

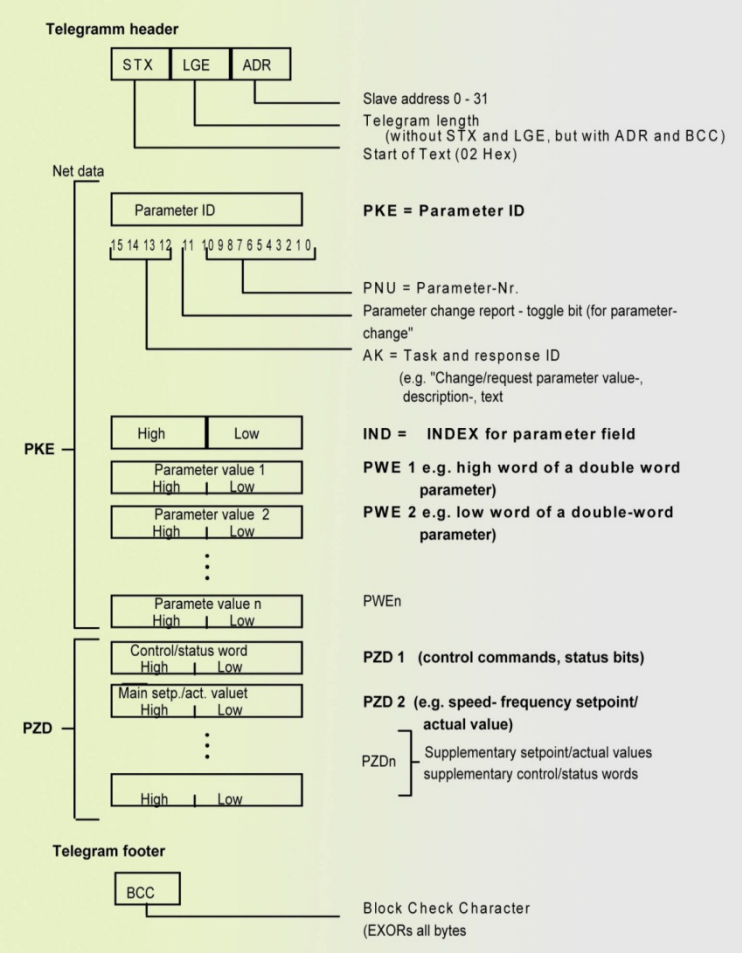
```
Unsigned char RS232_ReadChar(void) {
	while (!(UCSR0A & (1 << RXC0)))		// Wait until a data is available
	{
		asm volatile("nop");
	}
	return UDR0;
}
```

If interrupts were used, it would be a more elegant solution. Trying to improve the code only the read direction could be realized with interrupt `(USART0_RX_vect),` while the `RS485_WriteChar()` stays unimplemented. As a half-solution was rather unacceptable, the blocking functions were used instead.

Another potential issue is a missing timeout for the read-function `RS485_ReadChar().` If there will not follow any response from the machine controller, then this function will block the controlled forever. Both the interrupt and the timeout should be introduced in the next version of the program.

4.1.5 Communication with USS protocol

The structure of the USS protocol can be found in the next page:



This picture has been extracted from the USS specification protocol on page 72 (uss_24178253_spec_76.pdf).

The Telegram consists of header, footer and the net data. The header is in our case always the same, because the length of our telegram is fixed. The footer is the check sum and depends on the content of net data. The net data is the core of the telegram. It consists of PKE and PZD fields. Once the telegram is ready to send the transfer can be started in ASCII mode.

Data transfer for RS485 is always realized in the half-duplex operation as showed here:

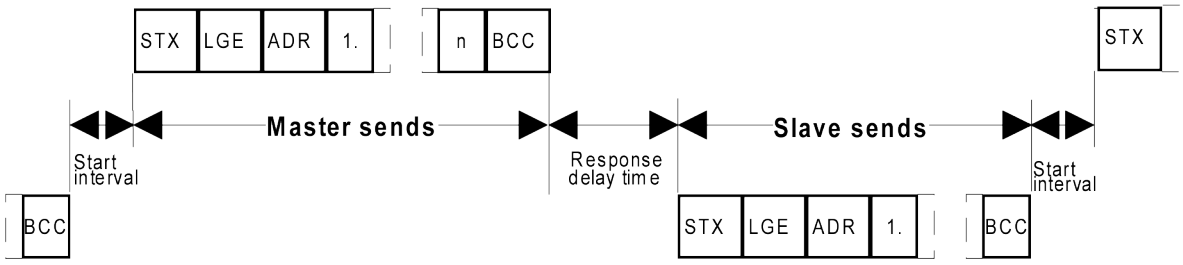


Fig. 5.3: Send sequence

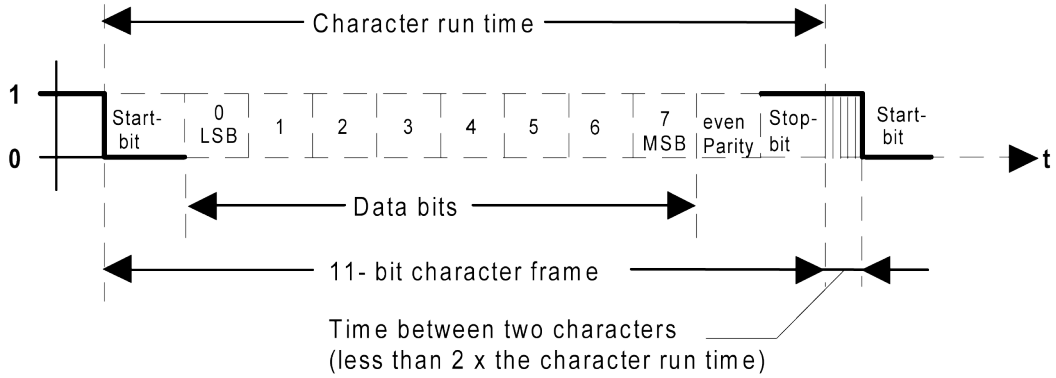
The start interval and the Response delay time need to be monitored by the master and cannot be exceeded. The maximum permissible response delay time is 20ms; however it may not be less than the start interval. The start interval can be found in this table:

Baud rate in bit/s	Start interval in ms
9600	2.30 ms
19200	1.15 ms
38400	0.58 ms
187500	0.12 ms

Table 5.1: Minimum start interval for various baud rates

In our case it is the Slaves responsibility to hold times, however the Master (embedded board) must also meet the times when queries on the values.

To be more precisely how the transfer performs, please refer to the bellowed picture:



Please notice that the frame has a parity bit set to ‘even parity’.

4.2 Hardware

The hardware plays a significant role in this project, for example through it the server application can update the database. An explanation of the hardware components can be split into three parts: Embedded board (Ethernut) with the LCD-display, RS485/RS232 converter and Cabling.

4.2.1 Embedded board with LCD-display

The PCB board for the Ethernut project has been made on the basis of the Egnite-project. Having the eagle data, the PCB was produced at “elcon” and manual soldered with proper electronic components.

Specification:

- ATmega 128 RISC microcontroller with up to 16 MIPS throughput
- Two serial ports, one RS-232 at DB-9 connector
- 4 kByte in-system programmable EEPROM
- 32 kByte external SRAM

- 22 programmable digital I/O lines

The block diagram of the embedded board is displayed in (Figure 20 - Embedded board block diagram) and the schematic diagram here: link.

For this project only the UART functionality has been used, skipping the Ethernet part. In the next version it should contain an implementation of memory access for telegram data and a driver for the RTL8019-Controller for direct connection to the Ethernet network.

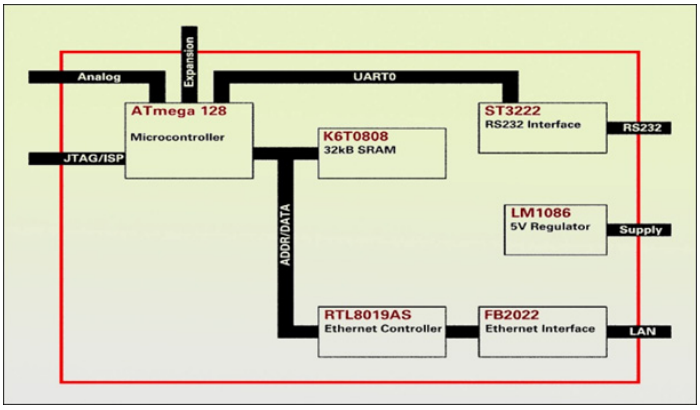


Figure 20 - Embedded board block diagram

The LCD display of Nokia 5110 fulfills the embedded board. As both serial ports are obtained, the display has been introduced for debugging purposes, and finally to display the status of the working board to the user. The driver has been programmed on the basis of SPI (Serial Peripheral Interface) without interrupts. The link displays connections to the embedded board and in this link the more specific description of used pins.

For the data sheet of used PCD8544 chip inside of LCD display the reader may look here: link.

4.2.2 RS485/RS232 converter

To connect to the machine controller it was necessary to build an RS232-RS485 converter. The MM75 operates with RS485 standard, while the embedded board operates with RS232.

RS232 and RS485 are in fact two different standards. The voltage levels and lines itself are not the same. In the simplest version of RS232 three lines are used: TX, RX and GND. In the RS485 standard only two lines A and B transport the information. For the RS232 is the ground the signal reference. The lines in RS485 are differential (like the CAN line).

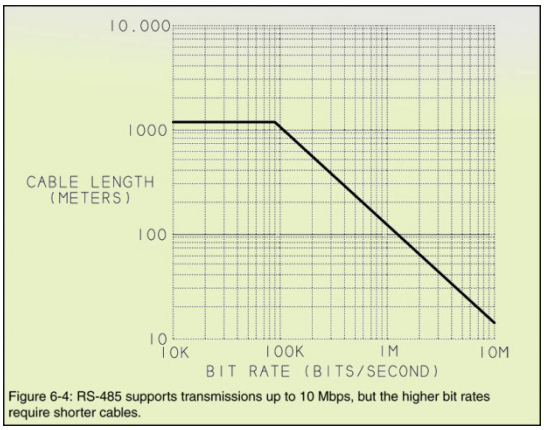


Figure 21 - Dependence of the length and bit rate of RS485

The length of the cables differences significantly. While the RS232 cannot be longer than 15m (20m if shielded), the RS485 can be up to 1200m long. The (Figure 21 - Dependence of the length and bit rate of RS485) table shows the dependence of the length and the bit rate.

Three different boards were made, but finally only one was used. The picture of the final board can be found here link.

The converter is based on the project of Christian Dietrich of Friedrich-Alexander-Universität in Erlangen. The reader can find the schematic diagram in this link. The active components are 75176 as Differential Bus Transceiver, and the MAX232 as the line driver/receiver.

The RJ45 is going to be connected to the MM75, while the DSUB is connected to the embedded board. For expanding the interface with even more RS485-devices this link will be helpful.

Both the embedded board and the RS232/485 converters are supplied with one 12V source.

4.2.3 Cabling

A proper connection of hardware requires cables, plugs and sockets. But not only physical parts are important, but also terms like length of cables, termination or biasing must be considered.

One of the prepared PCB-boards for the signal converting was a full-duplex RS485 converter of the Visgence Company (India). As there are two ways for connection (half and full duplex) in the RS485 standard, but only the half-duplex is used on machine controller. Thus the solution of Visgence was useless for our purposes.

Another problem was encountered during the project was isolation to the ground. While connecting to MM75 and the embedded board to the PC an electrical shock was experienced. It could simply avoid the shock by a proper grounding of the devices.

An example of ground-isolation is here link.

5 Test

For testing purposes was used a simplified V-model, which is showed in the (Figure 22 - V-model).

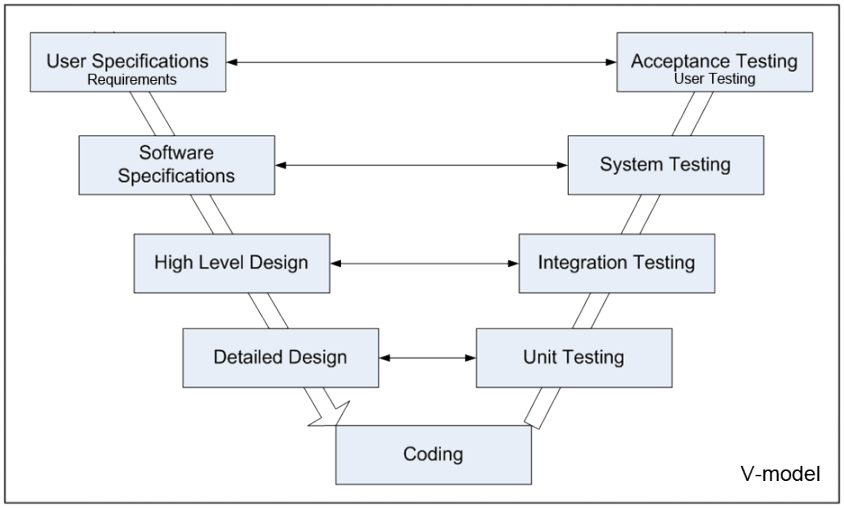


Figure 22 - V-model

5.1 Hardware test for connections on serial port

The testing of cabling between the embedded board and the machine controller requires schematics. Once connected properly this test do not need to be repeated again. Some help are LEDs mounted

on the RS232-RS485 converter board, where both should blink during the connection. An additional check with the oscilloscope will be required if the device does not respond. An example of waveforms for lines A(-) and B(+) of RS485 are provided in this link.

5.2 Coding phase

During the coding phase, one developer has explained his code the other developer. By this explanation it has been ensured, that both the code has been developed correct. This so called Code Walkthrough has been performed for the Quality Assurance.

The source code, which was intended to be tested, has been grouped into a small set of test cases. Here only the MicroMaster Inverter, the RS485 converter and the dedicated program were used. Finally the state of variables has been monitored, to analyze the programmer's logic and assumptions. The results were displayed in the application window.

5.3 Unit testing

An individual unit of source code has been tested each time when they have been introduced into the code. The goal of this Unit Testing was to see, that each part of the program is proper isolated and that the individual parts are coded correct. Some examples can be showed here:

- Can the embedded board talk to the machine controller?
- Can the embedded board talk to the PC?
- Can database app listen on the serial port?
- Can database app talk to the database?
- Can PC app talk to the database?

5.3.1 Unit Test of the connection to the database application

The connection to the database application should also be tested before using. This has been done by a sending of a string as the name of the board to the database application and requires a send back from the database application. By comparing both strings it can be evaluated that the connection is made. When no database is running or some other error prevents the database to connect to the database application, an error will cause the program to exit.

For the PC application, there was made a user test. This is for the design and features of the application. After the test there were useful information what should be added and what should be changed. For the application implementation, there was one test which is for the connection to the database. If the connection fails the user will receive a message “there was an error connecting to the database. Check the database connection settings”.

5.3.2 Unit test of USS-protocol

At the very beginning of the testing of the protocol only the terminal program was used. Based on the theory of USS protocol many different telegrams have been created and send from the Terminal program to the machine controller. Unfortunately, none of them has been responded by the unit. By sniffing the serial lines on RS232 port, while running the ‘Drive monitor’ from Siemens, a proper telegram has been obtained. Having the working telegram also the BCC sum could be calculated free of errors.

Even when the physical connection exists and connection parameters are fine, the machine controller would not respond until it got a proper telegram. This test can be realized in two steps: firstly by inserting a timeout within the telegram must arrive to the embedded board. And secondly by calculating the BCC code from the incoming bytes and comparing them to the BCC code which arrives with the bytes. Both values must be the same.

5.4 Integration Testing

For the Integration Testing the software and hardware components has been combined and tested to evaluate the interaction between them.

5.4.1 MicroMaster Inverter <=> Embedded board

For this purpose the MicroMaster Inverter has been replaced by a software application, which connects to the serial port to the embedded board. In particular the embedded board will send a request telegram to the application every 5 seconds. Then the software application will randomly response on the telegram, another time it wills not response at all. The purpose of this test is to simulate if embedded board reacts correctly to the telegrams generated by the software and how it reacts if nothing comes from the test application.

5.4.2 Embedded board <=> Database application

The integration testing has not been executed on the database application and no results will be presented here. However the testing could be performed this way: The embedded board needs to be reprogrammed with interference in the byte flow. The interferences should simulate misrepresented bytes during transmission, which could happen with 1200m of twisted pair connection. In the sending protocol the first byte will be the length of the whole flow. The checksum generated from the message will be added to the end of the data. Then the database application will receive a message and it will recognize if the message was delivered free of errors based on its length and its checksum.

5.5 System Testing

The System Testing ensures that the code does what it is intended to do. To satisfy the System Testing it was necessary that the functionality specified in the requirement specification works. By the check of the requirements 72% has been well tested, 28% were not. Here are requirements, which are not tested at all because of missing implementation:

- The embedded board can see changes of the machines sensors – not done
- The embedded board can send changes of the machines sensors to database server – not done
- The user application can show the workload of current day – not done
- The user application can show the work load of the period – not implemented

5.6 Acceptance testing (User test) or what elcon said:

The user testing has been performed in the company of “elcon” through the product owner (Jesper Gjøl). This is the feedback that is given to us:

"The interface seems to be structured well. We can focus on the parts of the application, which are important for us. In particular we like the upcoming maintenance and the graphical display of actual status of machines. We think also, that a missing web-functionality could be a disadvantage for us."

A few days after last feedback the CEO of 'elcon' has send this message:

"Hi Guys, as you know already I travel a lot and would like to see the status of machines from outside. If you are able to improve you application a bit, then I would pay you extra money for your effort."

As the web-functionality was not required, but it is expected now – then it has been promised to the customer to develop this missing feature in the next version of the system. In general the feedback was positive and satisfied both sides.

6 Results

The purpose of the tests is to ensure that the code (it could be also the hardware) is working as expected. The tests speed up the developing process. The results from the tests show clearly that the code and the hardware works correctly. Tests with the machine controller when it is powered off and database when it is turned off works as expected.

However, it was quite difficult to define expected results for the USS protocol, because of the lack of experience within the working field.

7 Discussion

Main application:

The main application had two options which technology it should use. The options were Windows forms and WPF (Windows Presentation Foundation). The WPF had nicer UI, but the windows forms were chosen based on the flow of the data. If the main application will not be open for a few days the event table will be filled with large amounts of data. Furthermore the application when starts takes few seconds to start because of the database connection.

The model package contains a model which is similar to the database. The implementation took this part of action based on the large amount of connections to the database. The database application access the database at least every second depends on how many machines are connected to the system and how many there are turned on. Based on this fact the decision was made to implement the model also in the application to reduce the data loss.

Embedded board:

As mentioned before there could be some issues with the connection to the serial port. Suppose that the telegram, which is going to be sent, would not be prepared correct way, so the other party will wait for the incoming message forever. To avoid this situation, it would be necessary to implement a timeout variable. This would require another thread, which in case of timeout will kill the waiting thread.

Two kinds of interrupts can be distinguished: a hardware and a software interrupt. Both kinds are not used in our project. However an implementing of them would increase the value of the project.

An example of the hardware interrupt could be an emergency stop. The interrupt line on the LOW state will set the embedded board into the fail state and send a correspond message to the PC application, which further will set a specified alarm-value into the database. The GUI should be able to show the alarm on screen.

A software interrupt may be used for SPI or serial connection. As mentioned in the implementation part, the microcontroller is blocked because of waiting for byte complete during sending or receiving bytes. By using an interrupt, it is possible to ensure that the processor can process other commands. It would be of great benefit when using threads.

The use of the embedded board may seem to be not significant, if the whole power of the board was not used. This could be true, as the board has much more possibilities than used in this particular version of the project. What if some sensors will be connected to the board? A typical flow switch needs one input line. A thermocouple will use the DAC functionality. A reed switch is also to be connected to the ON/OFF input line. There must be also sensors acting as SPI-slaves or using the TWI bus. All those possibilities cover the embedded board. It should be considered to introduce this in the next version of the project.

The next thing is the use of Ethernet chip RTL8019AS. Unluckily the network connection did not work. Maybe the fault was caused by the chip failure itself or some failure during soldering process. The network would be a powerful feature on this project giving us more flexibility and more likely the direct access to the database. A drawback could be the size of SQL library needed for implementation having in mind the flash size, but it is not ruled, that a solution for this problem would be found quickly.

Testing

Not all of the tests have been implemented into the code yet. However the system is working without any exceptions at this point. Some test will be included further in the implementation.

8 Conclusion

The project was not fully completed; to be precise around 75% was finished. The 25% which were not completed was (Getting the status of the sensors from the machine). Which led to other requirement which were not completed (workload calculation). Based on the work, the project and the number of members in the group it is a realistic result of the project.

All group members achieved knowledge in the specific fields which is leading to self-improvement. Also management and communication skills have improved.

The customer was satisfied with the solution. Also, he suggested us to continue with this project after the group will have time, to fulfill all the requirements.

Future of the project:

Database: including part position to be able insert two the same name parts. Include an extra table for machine id configuration for handling the machine names.

User application: include an overview of the machine, include web server for checking machine park overview

Embedded board: include two ways communication with database to get all information about where and which machine is connected. Include implementation of other protocols, which will enable to connect more devices. Include implementation of network on Ethernut and the web server for direct access. Simplifying of transmission data on RS232 line.

9 References

Axelson, J. (2007). *COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems* (Second Edition ed.). Madison WI 53704: Lakeview Research LLC.

Beeby, S., Ensell, G., Kraft, M., & White, N. (2004). *MEMS mechanical sensors.* Norwood, MA 02062: ARTECH HOUSE, INC.

Capehart, B. L., & Middelkoop, T. (2011). *Handbook of web based energy information and control systems.* Lilburn, GA 30047: The Fairmont Press, Inc.

Frank, R. (2013). *Understanding Smart Sensors (Third Edition).* Boston-London: Artech House.

Hersent, O., Boswarthick, D., & Elloumi, O. (2012). *The internet of things : key applications and protocols.* Chichester, West Sussex, PO19 8SQ, United Kingdom: John Wiley & Sons Ltd.

Hosek, P., & Diblik, M. (2009). Implementation of Siemens USS Protocol into LabVIEW. Liberec, Czech Republic.

Huang, J., & Zhang, X. (2014, September). The Design and Realization of Ship Power Management System Based on Modbus. *Applied Mechanics and Materials* , pp. 561-566.

Möller-Nehring, W., & Bohrer, W. (1994, September). Universal Serial Interface Protocol, Specification. (09.94). Erlangen, Germany.

Reynders, D., & Wright, E. (2003). *Practical TCP/IP and Ethernet Networking.* Burlington, MA 01803: IDC Technologies.

Reynders, D., Mackay, S., & Wright, E. (2005). *Practical Industrial Data Communications, Best Practice Techniques.* Oxford: Elsevier Ltd.

Righini, G. C., Tajani, A., & Cutolo, A. (2009). *An Introcuction To Optoelectronic Sensors, Series in Optics and Photonics - Vol. 7.* USA: World Scientific Publishing Co. Pte. Ltd.

Sinclair, I. R., & Dunton, J. (2007). *Practical Electronics Handbook.* Oxford: Elsevier Ltd.

Strauss, C. (2003). *Practical Electrical Network Automation and Communication Systems.* Burlington, MA 01803: IDC Technologies.

Sun, J. S., & Guo, Z. P. (2014, Jun). Web-Based Monitoring for Frequency Converters with USS Interface. *Advanced Materials Research* , pp. 1866-1869.

Tipsuwanporn, V., Numsomran, A., Samaimak, S., & Harnnarong, S. (2014). Development of redundant bus library for arduino to apply in SCADA system. *14th International Conference on* (pp. 42-46). Seoul: IEEE.

Using USS Protocol with MICROMASTER MM420. (2001, April). Germany.

Xunwen, S., Shaoping, W., Dongmei, Z., & Qishen, Z. (2010). RS-485 Serial Port Pseudo-full-duplex Communication Research and Application. *Prognostics & System Health Management Conference.* Macau: IEEE.