

STUDENTWORK

Final report

Data acquisition system with database using OPC

25.05.2010

Group:

Michał Bochenek

Kamil Folkert

Final report HO2-300 05/2010

<http://prosjekt.hisf.no/~10opcdp/>



www.hisf.no

 HØGSKULEN I
SOGN OG FJORDANE

TITLE: Final report	DATE: 25.05.2010	REPORT NUMBER:
PROEJECT TITLE: Data acquisition system with database using OPC	ACCESS:	NUMBER OF PAGES: 50
	URL: http://prosjekt.hisf.no/~10opcdp/	
GROUP: Michał Bochenek Kamil Folkert	SUPERVISORS: Marcin Fojcik Joar Sande	
EMPLOYER: HSF		
SUMMARY: Our project has been developed as a Senior Design Project. The order for the project came from HSF, and would be considered as a part of researches pursued by HSF in cooperation with Silesian University of Technology. Realisation of the project took place on the premises of Sogn og Fjordane University College in Forde. The subject was to write the OPC client for the Linux platform, that would send the data to the server and to design the database, holding the processed data. The communication should be reliable and the data presentation should be clear and understated.		
KEYWORDS: HO2-300, Senior Design Project; Data acquisition system with database using OPC		

Foreword

This report is the final document for the project which was made as the Senior Design Project. Project's title is Data acquisition system with database using OPC. The order placement company was the Høgskulen i Sogn og Fjordane University College, represented by Joar Sande and Marcin Fojcik. The whole project was developed on the premises of the HSF. Main purpose of the work was to create the system, divided into two parts, according to the runtime platform that would acquire, send and store required data.

Looking at the project from the perspective of the whole studying period it required the largest amount of time spent on the development. The most difficult part was the Linux node software, as the work connected with it included going through the NPE, OPC UA documentation which took a lot of time and required writing some interfaces, procedures and functions from the scratch.

We are really grateful to the Høgskulen i Sogn og Fjordane University College for their support and giving us the opportunity to work on the project in Førde as exchange students. Development of the project allowed us to acquire more pieces of information about using OPC systems not only within the Windows environment and also gave us the chance to build a system consisting of industrial controller, OPC server and MSSQL Server database.

Table of contents

Foreword	3
Table of contents.....	4
1. Abbreviations and symbols	6
2. Summary	8
3. Introduction.....	10
3.1. Project background.....	10
3.2. Division of work	10
4. Theory background	11
4.1. Description of the current situation	11
4.2. Database	11
4.3. Database management system	13
4.4. OPC Historical Data Access	14
5. Objectives.....	16
6. Tools	18
6.1 Programming language.....	18
6.2 Development environment	19
6.3 Source code management.....	19
6.4 Additional tools – Software Development Kits (SDK).....	19
7. Windows part implementation	21
7.1. Database operations.....	21
7.2. XML operations.....	22
7.3. OPC read	23
7.4. Stored procedures and functions	24
7.5. User interface	25
7.5.1. Server name form.....	25
7.5.2. Main form.....	25
7.5.3. Database server settings form	27
7.5.4. Database variables’ write form	28
7.5.5. OPC Variables form	29
7.5.6. About form	30
7.6. Additional libraries	30
7.7. Additional software	31
7.8. Further development.....	32
7.9. Configuring the operating system	33
7.10. Principle of operation	33
8. Linux part implementation.....	35
8.1. NPE scripts	35
8.2. Mediation node application	37
8.3. OPC UA Server application	38
9. Organization	40
9.1. Developers.....	41
9.2. Project supervisors	41
9.3. Order placement company.....	41

10.	Project administration.....	42
10.1.	Carrying out project according to plan.....	42
10.2.	Gantt diagram	42
10.3.	Week schedule	43
10.4.	Meetings schedule	44
11.	Budget of the project	45
12.	General project evaluation.....	47
13.	List of figures	49
14.	List of tables	49
15.	List of appendixes.....	49
16.	Reading list	50

1. Abbreviations and symbols^[4]

OLE	Object-Linking and Embedding - a technology developed by Microsoft that allows embedding and linking to documents and other objects
DCOM	Distributed Component Object Model - a proprietary Microsoft technology for communication among software components distributed across networked computers
OPC	OLE for Process Control – communication standard developed and maintained by OPC Foundation
OPC UA	OPC Unified Architecture – the most recent OPC specification
OPC HDA	OPC Historical Data Access – protocol used for retrieving the historical data from the appropriate HDA server, based on the DCOM model
COM	Component Object Model – a standard, developed by Microsoft Corporation, of creating programming interfaces
DLL	Dynamic link library – Library of software components, which can be shared by different applications
SDK	Software Development Kit, set of development tools that helps create applications for a certain software package
IDE	Integrated Development Environment - also known as integrated design environment or integrated debugging environment is a software application that provides comprehensive facilities to computer programmers for software development.
GNU Project	Free software, mass collaboration project, announced on September 27, 1983, by Richard Stallman at MIT
GCC	The GNU Compiler Collection - a compiler system produced by the GNU Project supporting various programming languages
ANSI C	is the standard published by the American National Standards Institute (ANSI) for the C programming language. Software developers writing in C are encouraged to conform to the requirements in the document, as it encourages easily portable code.
.NET	Microsoft .NET Framework – software developing platform developed by Microsoft Corporation
C#	One of the most popular high-level programming languages used to write programs for .NET platform
SQL	Structured Query Language - database computer language designed for managing data in relational database management systems

T-SQL	Transact – SQL - proprietary extension to SQL, developed and used by Microsoft and Sybase, including additional features, like: Control-of-flow language, local variables or string, date, mathematics support functions.
RDMS	Relational Database Management Service – a set of applications used to manage data, describe data structures inside the database and retrieve data queried by a client application or the end-user.
MSSQLSERVER	Microsoft SQL Server is a relational model database server produced by Microsoft. Its primary query languages are T-SQL and ANSI SQL.
MSDN	The Microsoft Developer Network - the portion of Microsoft responsible for managing the firm's relationship with developers
GPRS	General Packet Radio Service - packet oriented mobile data service available to users of the GSM systems
GSM	Global System for Mobile Communications (originally from Groupe Spécial Mobile) is the most popular standard for mobile telephone systems. The standard contains packet data capabilities (GPRS)
NPE	Dedicated microcontroller with embedded GSM modem, running on Linux, meant to act as a mediate node in our system
NTP	Network Time Protocol, protocol for synchronising the clocks of computer systems over packet-switched, variable-latency data networks.
HSF	Sogn og Fjordane University College
JNI	Java Native Interface – enables adaptation of low level functions, to be accessed from the Java code as the class members
XML	Extensible Mark-up Language - set of rules for encoding documents electronically. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards. There are many programming interfaces that software developers may use to access XML data, and several schema systems designed to aid in the definition of XML-based languages.
JVM	Java Virtual Machine - enables a set of computer software programs and data structures to use a virtual machine model for the execution of other computer programs and scripts.
HO2-300	Code of the Senior Design Project subject at the HSF

2. Summary

Many fields of research connected with different kinds of studies require automated systems gathering and processing information. Specialists can think of some vital solutions for the problems they come across every day. However to perform their researches they need specific data. The reason for commissioning this project is in tight connection with those needs.

Manufacturers give opportunity to use equipment that could fulfil many tasks with the dedicated software. But for the moment, there is not any solution including the data acquisition and storage.

Devices that are able to work in the field, sometimes in extreme weather conditions have to be as portable as possible, which applies to their dimensions as well as the power supply. Techbase company from Poland created the NPE microcontroller, using Linux operating system, being able to stand the low temperature, acquire data from sensors with digital and analogue interface and send it further using the GSM modem. This type of device is the appropriate microcontroller to become the node of our system. The software for the node, however, must be written from the scratch.

The best way to store the data and possess flexible access to it is undoubtedly the database and the database management service. The Microsoft SQL Server is a well-known solution for this kind of application. The extended management software provides graphic user interface for the database administration but also gives easy-to-use interface to the software developer.

In order to easily access and browse the stored data and manage the frequency of database write, there must be particular software created. One of its features has to be the possibility of its further development. Due to the usage of .NET platform any changes that would be of vital importance in future, shouldn't give any problems to the developer. The same goes to the database changes.

We mentioned two separate parts of the system. The question, how they should be connected with each other can be answered with the OPC server.

OPC UA is one of the most modern communication standard used to participate in data exchange. One of the most important advantages of using the OPC server is the independence from the hardware working as the data acquisition nodes.

This is why the OPC server is a part of the project - it combines two operating system platforms as well as increases system scalability. To make both, Linux and Windows applications work properly as the data sender and receiver they were equipped with some basic functions of the OPC clients.

Although setting up the whole system took a lot of time, there is a set of results at the end of the project. It contains the Linux OPC client application, reports made after each meeting, preliminary project and this very report.

Crucial to our work was the work division, so that every person could spend the same amount of time at the project development. As this was a very big project, it required wise administration. Management abilities should not be overlooked, as they can turn out to be of the same importance as the knowledge about the OPC client-server application and database application development.

3. Introduction

3.1. Project background

This project was made within the framework of Senior Design Project subject, with code HO2-300, in the spring semester 2010. The company that ordered the project is the Høgskulen i Sogn og Fjordane University College, represented by Marcin Fojcik. The project name is Data acquisition system with database using OPC. Our main task was to create a set of applications that would be used to acquire, pack, send, and store the data. There is the application to view the stored data provided as well.

3.2. Division of work

Group working on the project consists of two persons. It was a big and important project, so it required a lot of effort. To properly fulfil tasks that the project was divided into, there had to be an appropriate work division. Since two different hardware platforms were chosen, two parts of the project seemed obvious. That sort of work distribution was a good choice, because the development could be run concurrently and there was not any problem with interfering with each person's task. Additionally, each person was able to document the part of the project that was completed, during the growth of the whole system.

4. Theory background

4.1. Description of the current situation

Due to the natural terrain shape in Norway, there are a lot of areas where the possibility of an avalanche/mudslide is high. Geology specialists run some research about predicting the place and time of a disaster. They consider usage of particular sensors, measuring the rain amount, temperature and the pressure in the soil. At the very moment the way of gathering data is very uncomfortable. Data from each sensor must be downloaded at the point where the sensor is placed. The amount of data gathered this way is undoubtedly too little.

In order to that, the development of the appropriate algorithm of predicting avalanches it is needed to possess some more data to analyse.

Projects connected with geology, HSF is conducting with cooperation with Silesian University of Technology and so that our project is to be the universal way of data acquisition, sending and storing. In future it could be used with the geological data system or any other, with the need for data processing provided by our system.

Our system is going to provide solution for both the part of acquiring the data from the sensors and data storage for further analysis.

4.2. Database

A database consists of an organized collection of data for one or more multiple uses. One way of classifying databases involves the type of content, for example: bibliographic, full-text, numeric, and image. Other classification methods start from examining database models or database architectures. Software organizes the data in a database according to a database model. Nowadays, we can notice that the relational model is the most common. Other models such as the hierarchical model and the network model use a more explicit representation of relationships.

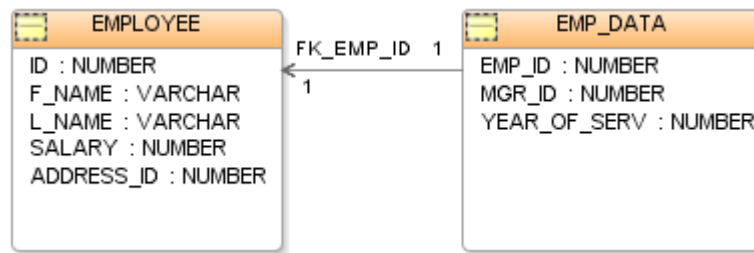


Figure 1 - Simple relational database

A relational database matches data by using common characteristics found within the data set. The resulting groups of data are organized and are much easier for many people to understand. The software used to do this grouping is called a relational database management system (RDBMS). The term "relational database" often refers to this type of software. Relational databases are currently the predominant choice in storing financial records, manufacturing and logistical information, personnel data and much more. The term relational database was originally defined and coined by Edgar Codd at IBM Almaden Research Center in 1970. Relational database theory uses a set of mathematical terms, which are roughly equivalent to SQL database terminology.

Table 1 – Relational database terms and their SQL equivalents

Relational term	SQL equivalent
relation	table
derived relvar	query result, result set
tuple	Row
attribute	column

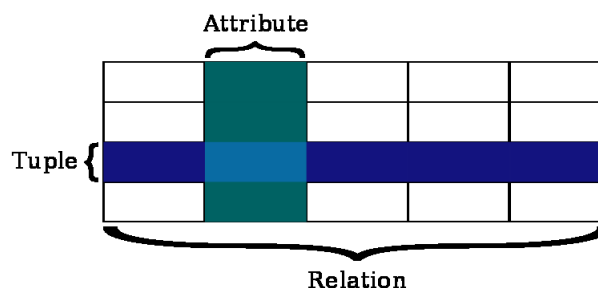


Figure 2 - Relational database terms

Queries made against the relational database, and the derived relvars in the database are expressed in a relational calculus or a relational algebra. In his original relational algebra, Codd introduced eight relational operators in two groups of four operators each. The first four operators were based on the traditional mathematical set operations:

- union operator combines the tuples of two relations and removes all duplicate tuples from the result.
- intersection operator produces the set of tuples that two relations share in common.
- difference operator acts on two relations and produces the set of tuples from the first relation that do not exist in the second relation.
- cartesian product of two relations is a join that is not restricted by any criteria, resulting in every tuple of the first relation being matched with every tuple of the second relation.

The remaining operators proposed by Codd involve special operations specific to relational databases:

- selection, or restriction, operation retrieves tuples from a relation, limiting the results to only those that meet specific criteria, i.e. a subset in terms of set theory.
- projection operation retrieves tuples containing only the specified attributes.
- join operation defined for relational databases is often referred to as a natural join. In this type of join, two relations are connected by their common attributes.
- relational division operation is a more complex operation, which involves essentially using the tuples of one relation (the dividend) to partition a second relation (the divisor).

Other operators have been introduced or proposed since Codd's introduction of the original eight including relational comparison operators and extensions that offer support for nesting and hierarchical data, among others.

4.3. Database management system

Data base management system (DBMS) is the system in which related data is stored in an efficient and compact manner. Efficient means that the data which is stored in the DBMS is accessed in very quick time and compact means that the data which is stored in DBMS covers very less space in computer's memory.

Thus, the DBMSs of today roll together frequently needed services or features of attribute management. By externalizing such functionality to the DBMS, applications effectively share code with each other and are relieved of much internal complexity. Features commonly offered by database management systems include:

- Query ability
- Backup and replication
- Rule enforcement
- Security
- Computation
- Change and access logging
- Automated optimization

A relational database management system (RDBMS) is a database management system that is based on the relational model as introduced by E. F. Codd. A short definition of an RDBMS may be a DBMS in which data is stored in the form of tables and the relationship among the data is also stored in the form of tables.

Components of a DBMS are:

- DBMS Engine
- Data Definition Subsystem
- Data Manipulation Subsystem
- Application Generation Subsystem
- Data Administration Subsystem

Most popular database management systems are: Microsoft SQL Server, IBM DB2, Oracle, MySQL, and PostgreSQL.

4.4. OPC Historical Data Access

The OPC Historical Data Server provides a way to access or communicate to a set of Historical data sources. The types of sources available are a function of the server implementation. The server may be implemented as a standalone OPC Historical Data Server that collects data from an OPC Data Access server or another data source. It may also be a set of interfaces that are layered on top of an existing Proprietary Historical Data Server. The clients that reference the OPC Historical Data server may be simple trending packages that just want values over a given time frame or they may be complex reports that require data in multiple formats.

The OPC Specification specifies COM interfaces, but not the implementation of those interfaces. It specifies the behavior that the interfaces are expected to provide to the client applications that use them.

The OPC Historical Data server objects provide the ability to read data from a historical server and write data to a historical server. The types of historical data are server dependent. All COM objects are accessed through Interfaces. The client sees only the interfaces. An OPC Historian Client application must implement a callback interface to support a shutdown request. The client may also implement interfaces for the various asynchronous connections that a server may provide. If the client expects to use (and the server provides) a particular asynchronous interface, the client must implement the matching callback. [1][6]

5. Objectives

Before the beginning of our work, there was no solution like the one we prepared, available on the market. The cost analysis that we had made led us to choose the particular hardware solution and to use the selected development software.

The final product possesses the following features:

- Linux part:
 - Java code
 - GSM/GPRS over the TCP/IP data transfer protocol
 - implementation of algorithms needed for synchronising time in mediation nodes and central node to ensure good quality of data and possibility of data transfer errors detection.
 - set of algorithms (packing and optimising data structures) used to prepare the data before sending
- Windows part
 - .NET C# code
 - Relational database (MSSQL Server)
 - OPC UA server – making the current data accessible
 - Graphic user interface for data presentation and setting up the database write
 - OPC HDA server – making the historical data accessible
 - Bridge between OPC UA Server and the database application

Although there have been some changes made to the list of elements of our project, the amount of time dedicated for the project development turned out to be enough to implement all necessary features we decided to be included in the system. All of the features listed above have been tested towards most often errors and the group is proud to present the fully operational system.

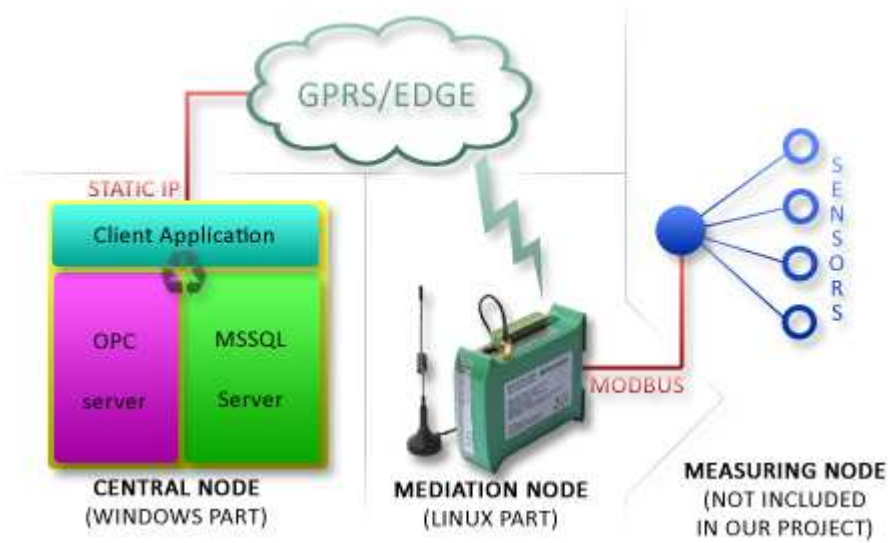


Figure 3 - System overview

6. Tools

Before starting the work, the developers group had to pick the tools to use. Firstly, a programming language had to be chosen, appropriate for the platform used in a particular project part. After the programming language had been selected, a development environment was a tool to choose. According to the language and platform, the variety of tools was not so wide. However, the improvement of quality and efficiency dictated the terms that chosen software had to possess.

6.1 Programming language

The first thing we had to think about was the programming language, appropriate for the used platform. As for the Windows part, the choice was relatively easy, as the .NET platform with its C# language is popular, and we have some knowledge and experience with applications written in C#. Of course we took into consideration other .NET languages like Visual Basic, however, our knowledge about C# is definitely bigger. Also, comparing to the C++ quite complicated memory management, type casts and sophisticated pointer usage, the advantages of C# seemed not to leave any doubts. Besides, it would not be wise to risk the inappropriate memory management, as C# possesses a built-in garbage collection program. The availability of libraries and cooperation with MSSQL Server database tools have their meaning as well.

The Linux part, as NPE controller is going to be used, should be implemented in Java language, because of its object-oriented nature, availability of free external libraries, support of TCP/IP communication and possibility of rapid application development. As NPE has only tiny core Linux system inside, we cannot compile source code on it and external compiler must be used for that purpose. For accessing NPE's hardware resources there is a C library delivered by producer, and for MODBUS communication open-source libraries are going to be used. To encapsulate C functions from mentioned library, we may use JNI to make them accessible as an object's methods.

6.2 Development environment

C# is one of the programming languages co-operating with the Microsoft .NET platform. In order to that, the most suitable environment for programming in the mentioned language would be the Microsoft Visual Studio. It is a commercial tool, however, as students of Silesian University of Technology we have the right to use it for non-commercial purposes free of charge. Microsoft gives the opportunity to test the latest version of Visual Studio – the Visual Studio 2010 RC, but the developers' group decided to use the previous Visual Studio 2008 version as the more reliable software.

NetBeans is one of the most popular IDEs. For that reason it is also the most reconfigurable and has very well provided system of updates. It supports various programming languages like Java, C++, C#, PHP. NetBeans IDE comes with multiple useful features, like code completion, snippets, javadoc analysis tool, and many more. It supports multiple Java platforms and has JAR-generating scripts built-in. NetBeans IDE is free, so we can use it without limitations.

6.3 Source code management

During the work on a project including code writing it is recommended to use a version control system. It improves work efficiency and makes it available to perform changes to the project almost on every computer. However, dividing the project according to the operating system platform and the fact that the group consists of two persons, the version control system was not necessary. Different programming language and structure of the project inside the development environment prevented the group from interfering each member's part of work. The whole process of creating the system did not bring up any problems with keeping all parts of the project up-to-date.

6.4 Additional tools – Software Development Kits (SDK)

Historical data that is stored in the database must be accessible for users that want to analyse or process the data for their purposes. The local viewing of the data is provided by the OPC2DB application written in C#. Apart from this solution, as it may be not so

convenient for all users, we added the OPC HDA server, written in C++. To avoid writing the server software from the very beginning we used the OPC HDA Software Development Kit, which was modified to suit all the functionality that our project must have possessed.

Looking at the HDA Server from the point of view of an administrator of the main server, it requires some attention at the beginning, to prepare operating system for cooperation, but later on, it is running as the background service, requiring no attention at all.

OPC Unified Architecture applications are not very common in Linux environment. However, we decided to use Prosys' Java SDK for UA server process development. It is platform-portable, so in the future it may be integrated with NPE data providing part and embedded into NPE as a complete stand-alone OPC UA server. For the research purposes the Prosys' Java SDK is free of charge. It is the only SDK which is supported by complete documentation and on-line help. We can recommend it to anyone who is interested in rapid OPC UA portable applications development.

7. Windows part implementation

The part of the project running under Microsoft Windows environment consists of many different elements, which are independent in some ways and simultaneously related to each other. In order to make the set of our applications work with system's other elements following sub-parts had to be implemented. Each point describes details of implemented functionality.

7.1. Database operations

The OPC2DB application that has been developed performs basic database operations which are the read of data already stored in the database and the opposite – writing acquired data. All operations that can be performed over the database are gathered within *DBOper* class.

Class structure is as follows:

```
public class DBOper
{
    protected String SqlConnectionStr;

    public DBOper()
    public DBOper(String SrvName, String DbName, String UserName, String
        Pass, Boolean PersistSecInfo)
    public void changeConStr(String SrvName, String DbName, String UserName,
        String Pass, Boolean PersistSecInfo)
    public void getValuesBetweenTime(DateTime startTime, DateTime endTime,
        String tblName, DataGridView dgv)
    public void writeToDB(String tblName, String value, DateTime timeStamp,
        int quality)
    public void createNewTable(String tblName)
    public void createNewTrigger(String tblName)
}
```

Fields:

String SqlConnectionStr – contains the connection string that makes it possible to connect to the database server

Constructors:

DBOper() – basic constructor without any argument – prepares the default connection string and assigns the value to the class field

public DBOper(String SrvName, String DbName, String UserName, String Pass, Boolean PersistSecInfo) – constructor that creates the object instance, with the connection string built using constructor's arguments

Methods:

public void changeConStr(String SrvName, String DbName, String UserName, String Pass, Boolean PersistSecInfo) – used to change the elements of the connection string

public void getValuesBetweenTime(DateTime startTime, DateTime endTime, String tblName, DataGridView dgv) – retrieves the data from the database – data is collected from the *tblName* table for the period beginning on *startTime* and ending on *endTime*. Then the data is connected as the data source to the *dgv* DataGridView on the user interface

public void writeToDB(String tblName, String value, DateTime timeStmp, int quality) – writes the data to the database – data are inserted into the *tblName* table, particularly the data consists of *value*, *timeStmp* and *quality*

public void createNewTable(String tblName) – executes the database stored procedure responsible for creating the table for another OPC Variable, the table name to be created is the *tblName*

public void createNewTrigger(String tblName) – executes the database stored procedure responsible for creating the trigger that fills the database write timestamp in the table for the table *tblName*

7.2. XML operations

The application that performs operations on the database and communicates over the network always possesses some set of settings that is necessary for maintaining user-defined configuration as well as other information that should not be lost after closing the application. The same situation can be observed in our project. We need to read/write the settings of OPC server, OPC variables, database server and instead of using the INI files, we decided to do that by means of XML technology. Following functions and procedures perform the operations on the XML settings files.

XML methods connected with the main application form:

public void readWriteStatusXML(List<CdbWrite> list) – reads which monitored variables from the *list* are being written to the database

public String readServerNameXml() – returns the OPC server name

public DbSet readDbSettingsXml() – returns the object of DbSet type, which is a structure containing database settings, with information about the current database settings

public void writeDbSettingsXml(DbSet currentDbSet) – writes the database settings stored in the *currentDbSet* structure to the XML settings file

XML methods connected with the server name form:

public void readFromXml() – reads the server name from the XML file and puts it in the appropriate text field

public void writeToXml() – writes the entered server name to the XML file

XML methods connected with the OPC variables form:

public void readFromXml() – reads information about monitored OPC Variables from XML file and puts them into a ListView

public void writeToXml() – writes information about monitored OPC Variables to XML File

XML methods connected with the database write form:

private void getVariableNames(ListView lv) - reads information about variables from the XML file, and puts them into the *lv* ListView

private void getWritableVariableNames(ListView lv) - Reads information whether the value of monitored variable should be put into database or not and enters appropriate variables into the *lv* ListView

public void writeChangesToXml() - writes all the changes made on the ListView control to the XML file

7.3. OPC read

One of the main parts of our project is cooperation with various OPC Servers. Particularly the OPC2DB application stays in stick connection with the TopServer, which is used as the bridge between our application and the Java OPC UA server. Library provided by the TopServer manufacturer gives the opportunity to call OPC read functions and modify some of their parts for custom purposes.

Every item, for which the read is performed provides information, assembled in the following structure:

```
public struct OpcItemValue<TValue>
{
    public TValue Value {get; set;}
    public DateTime TimeStamp {get; set;}
    public OPCQuality Quality {get; set;}
    public int ClientHandle {get; set;}
    public int intQuality {get; set;}
}
```

```
}  
    public String ItemID { get; set; }  
}
```

The OpcServer class contains following methods performing OPC read, and making read values accessible:

public OpcltemValue<TValue> SynchronousRead<TValue>(OPCItem item) - performs the Synchronous read for the *item* and returns the *OpcltemValue* structure for the mentioned item

void opcGroup_DataChange(int TransactionID, int NumItems, ref Array ClientHandles, ref Array ItemValues, ref Array Qualities, ref Array TimeStamps) – procedure invoking data changed event handler, if any, for each provided tag with the transaction identifier, number of items and arrays for every OPC factor: Value, Quality and Timestamp

7.4. Stored procedures and functions

The assumption of the structure of our project is that OPC servers, OPC2DB application and the database with its management service are working on the same machine. However it is not the only option for these elements' placement. For instance, the database could be put somewhere else. Thus, operations performed directly on the database are designed to be done by means of stored procedures and functions. Client application that requests data from the database needs only to invoke the stored procedure with appropriate parameters.

insertValue (@value NVARCHAR(10), @opctime DATETIME, @quality NVARCHAR(20), @tblname NVARCHAR(25)) – inserts OPC data: value, timestamp and quality into *tblname* table

getValuesBetweenTime(@startTime datetime, @endTime datetime, @tblname NVARCHAR(25)) – gets data from *tblname* table from the period of time starting at *startTime* and ending at *endTime*

getAllValues(@tblname NVARCHAR(25)) – gets the whole contents of *tblname* table

dropTable(@tblname NVARCHAR(25)) – deletes the *tblname* table

createTrigger(@tblName NVARCHAR(25)) – creates the trigger for *tblName* table, responsible for entering the timestamp of database write

createNewTable(@tblname NVARCHAR(25)) – creates new table with the name *tblname* for new OPC variable

insertTime<table_name> - every trigger has the name created according to this pattern, for example the table "NEWVALUE" will have the trigger *insertTimeNEWVALUE* created for it

All above procedures have been written using the Transact-SQL language and their code is enclosed in the Appendix A.

7.5. User interface

The graphic user interface (GUI) made for the OPC2DB application consists of seven forms. Each form has, of course, its own purpose and the functionality of every form has been thoroughly described below. Shape of forms may differ depending on the Windows version, the application has been built, run and tested under Microsoft Windows 7 Professional and figures used in the description are specific for this particular version.

7.5.1. Server name form

Just after starting the application the form for entering the OPC Server (TopServer) instance name. There is the name of instance read from *OPCServer.xml* supplied by default.

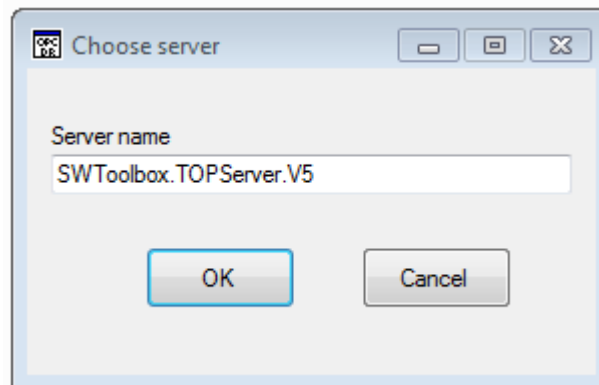


Figure 4 - Server choice form

7.5.2. Main form

When the user selects the instance to connect to, application's main form is loaded. Depending on the state of the OPC read operation its view is a bit different. Moreover, when the user decides to acquire data stored in the database, the content of some elements changes as well.

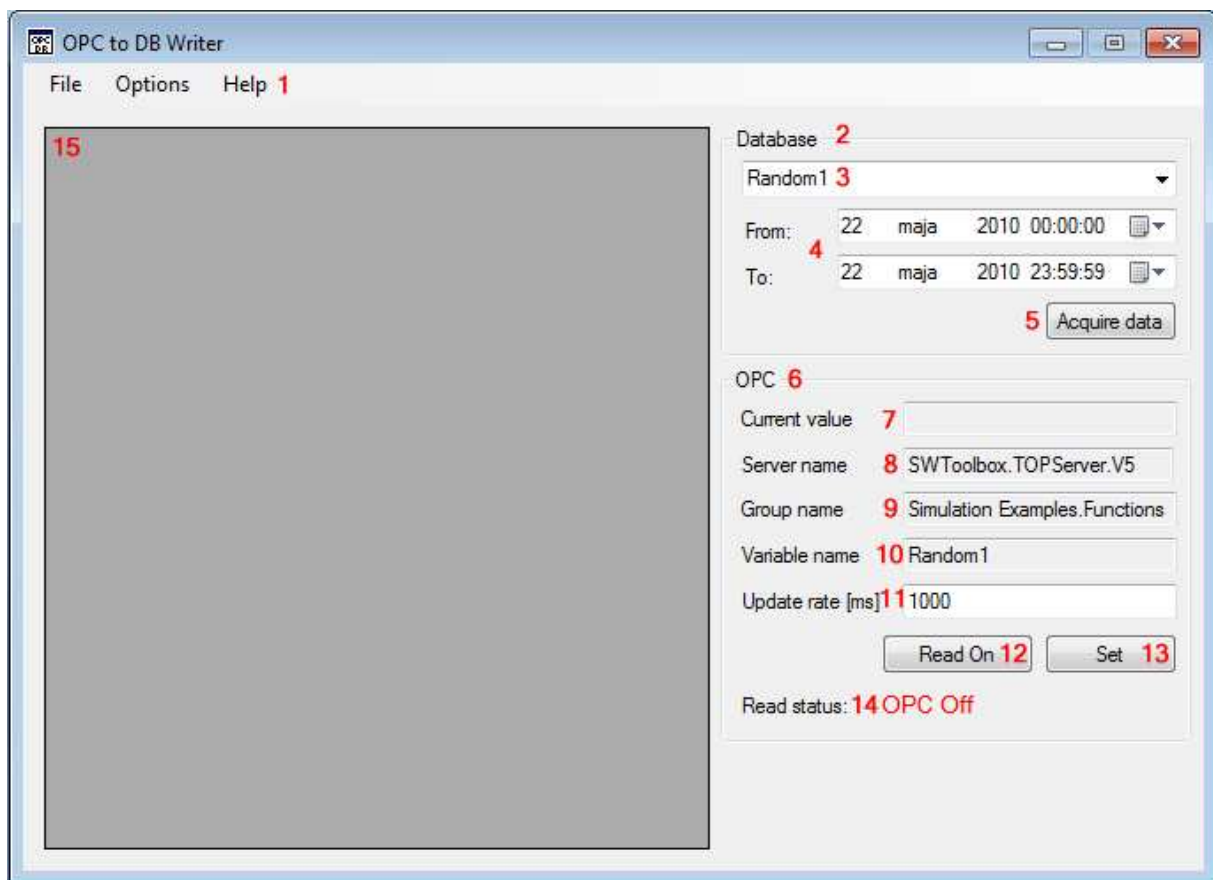


Figure 5 - Main form of OPC2DB application (OPC read disabled, no data fetched from database)

Detailed description of the form's elements:

1. Main menu
2. Database group box
3. Name of the OPC variable monitored by the application
4. Date/Time pickers for specifying the time period for data read from the database
5. Button for executing the database read
6. OPC group box
7. Current value of the variable selected in 3. (When OPC read is enabled)
8. Name of the server
9. Name of the group
10. Name of the variable
11. Update rate in milliseconds
12. Enable/Disable OPC read
13. Apply the entered update rate
14. Status of the OPC read
15. Grid view for the data acquired from the database

On the figure below the same main form is enclosed, but with other options active. The OPC read status is marked as enabled and data from the database is read for the given period of time.

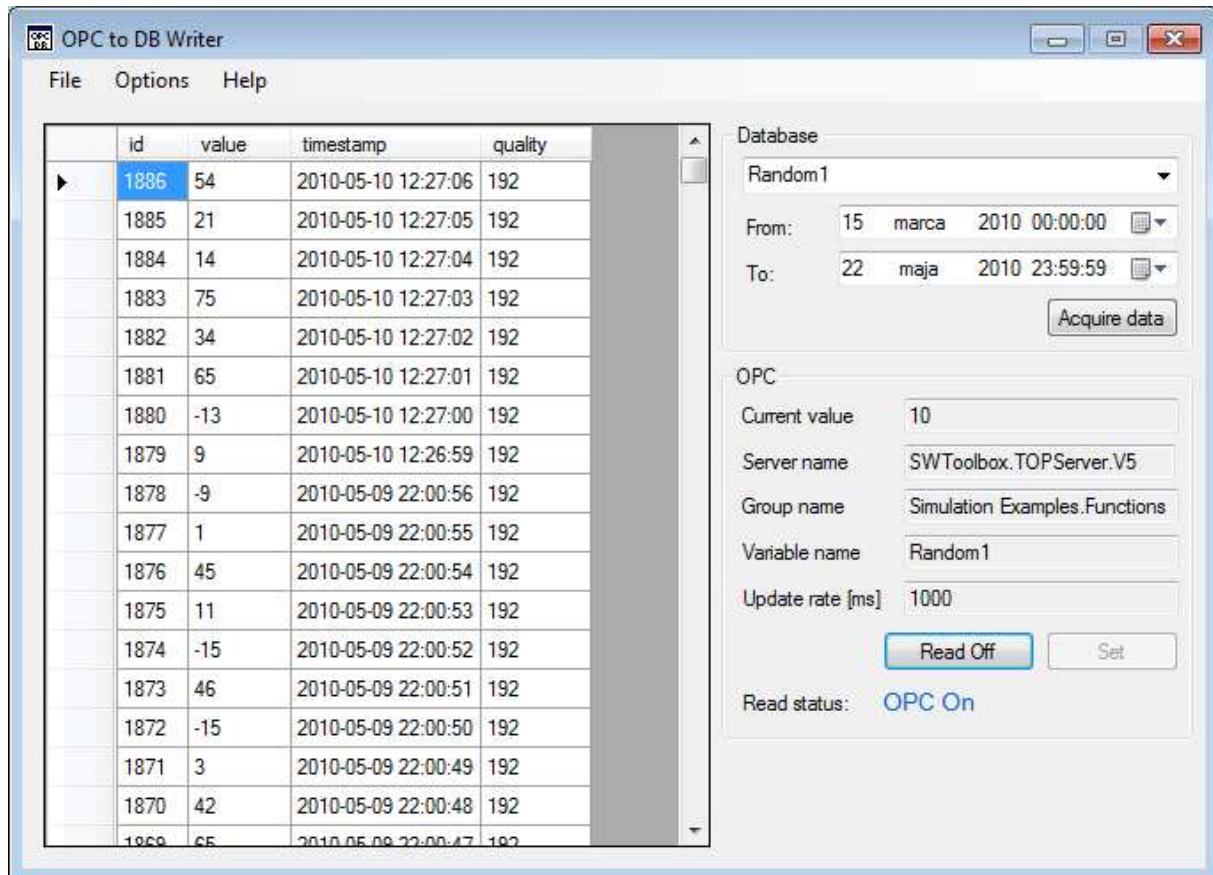


Figure 6 - Main form of OPC2DB application (OPC read enabled, data fetched from database)

7.5.3. Database server settings form

Having selected Options->Database the user is able to change database server settings. This simple form consists of text fields for entering the information required to successfully connect to the database server. User is requested to enter Host name or IP address, name of the database, name of the user that has enough privileges for the operation that could be performed using the OPC2DB application, user's password. Last thing is to select or unselect the checkbox *Persist Security Info*.

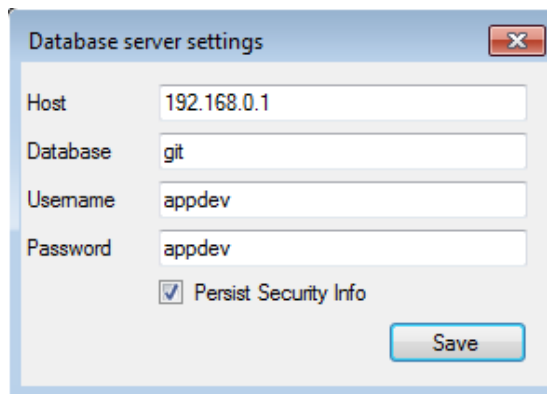


Figure 7 - Database server settings form

7.5.4. Database variables' write form

The form that allows users to select which of the monitored tags' values are written to the database is shown after selecting Options->Write. Form is constructed mainly from two list views. Left one shows the whole list of monitored variables, while the right one is filled only with those, which are currently selected for the database write. The basis for filling this form is the information collected from the *OPCVariables.xml* file, when the application is started.

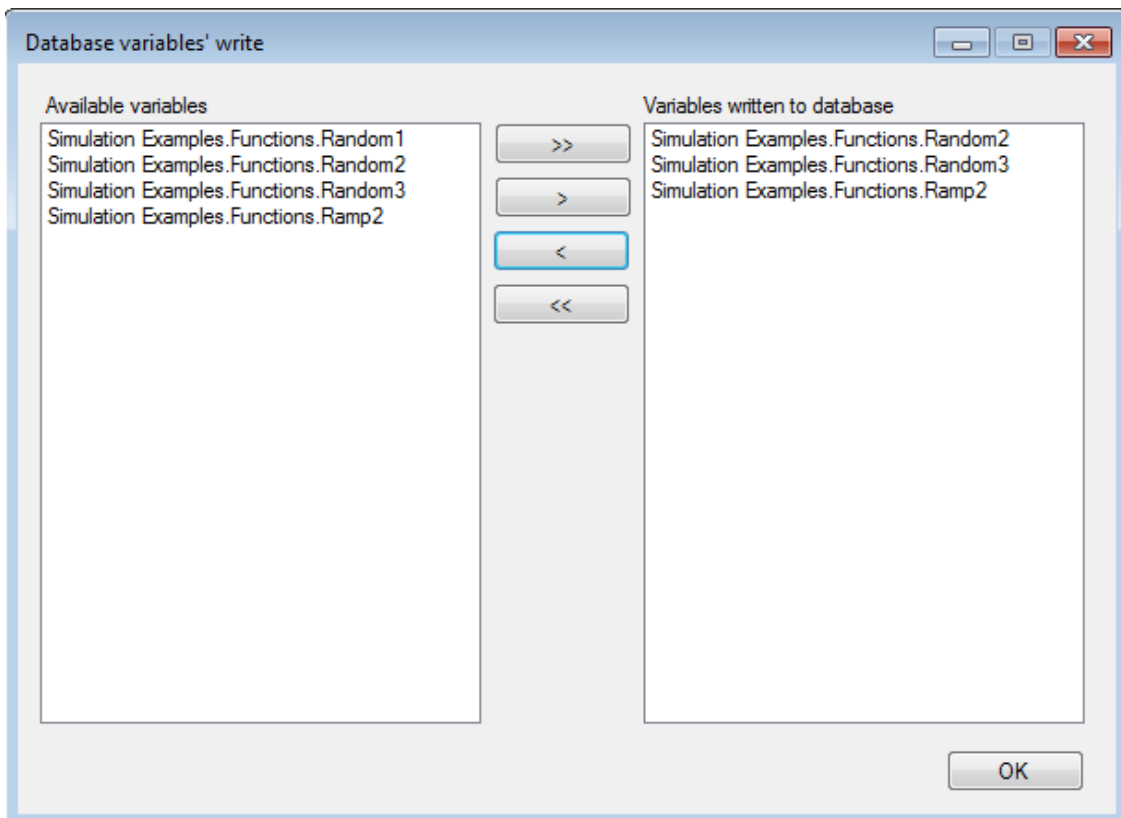
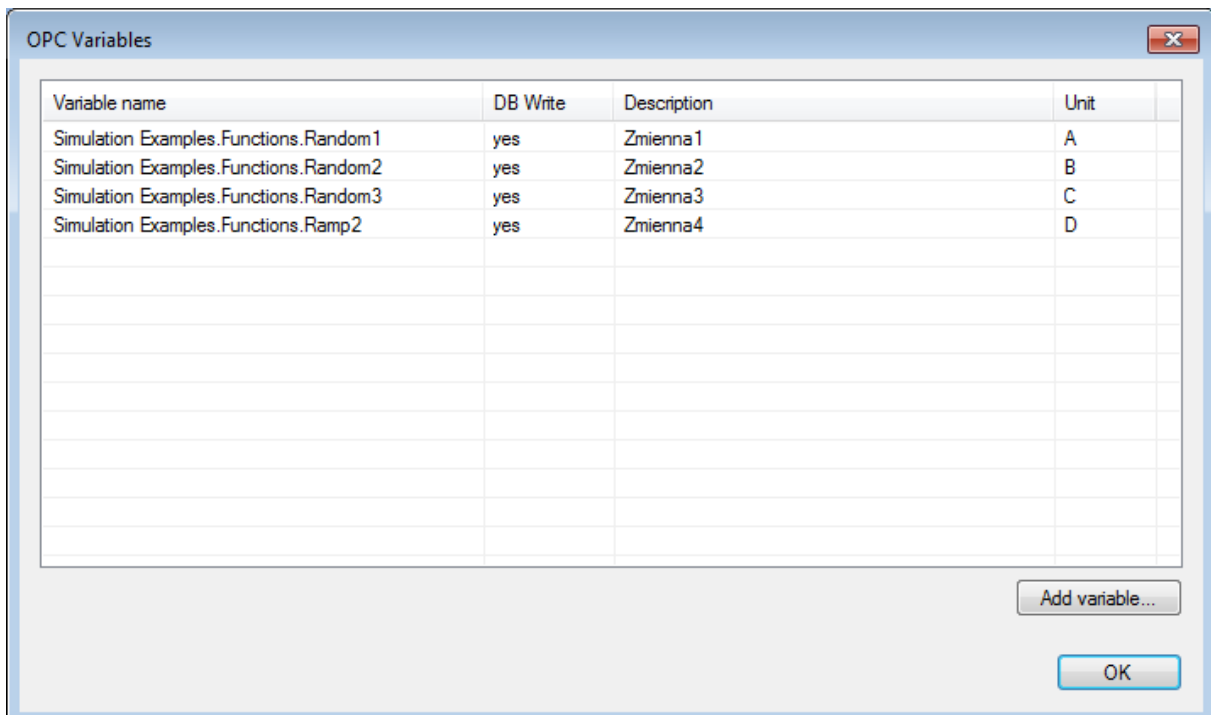


Figure 8 - Database variables' write form

7.5.5. OPC Variables form

As mentioned earlier, information on monitored variables are stored in the *OPCVariables.xml* file, however all changes are supposed to be made on this very form of OPC2DB application. As soon as the form is loaded, user is shown variable names, status of the database write, descriptions and measuring units of every monitored OPC tag.

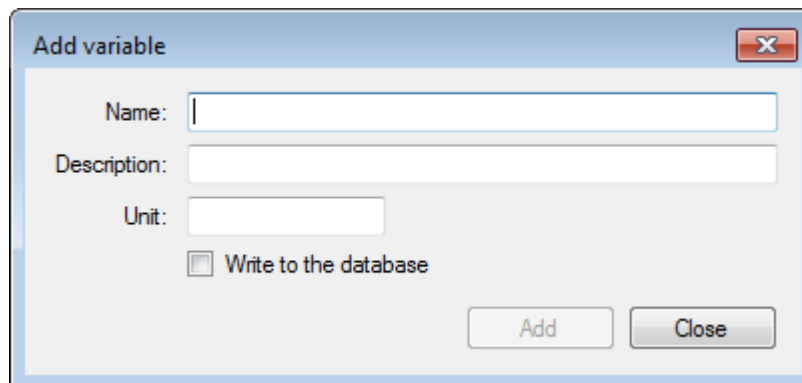


Variable name	DB Write	Description	Unit
Simulation Examples.Functions.Random1	yes	Zmienna1	A
Simulation Examples.Functions.Random2	yes	Zmienna2	B
Simulation Examples.Functions.Random3	yes	Zmienna3	C
Simulation Examples.Functions.Ramp2	yes	Zmienna4	D

Add variable...
OK

Figure 9 - OPC Variables form

The importance of this form can be observed, when the *Add variable...* button is clicked. Apart from filling the information about new variable, its description etc. adding new variable calls appropriate stored procedure in the database, creating new table and proper triggers in the tablespace. Similarly, when the user decides to delete the variable (delete option is available in the right-click context menu) its database table is dropped.



Add variable

Name:

Description:

Unit:

Write to the database

Figure 10 - Add variable form

7.5.6. About form

Information about the project which includes this application can be found in the about form, accessible through Help->About option.

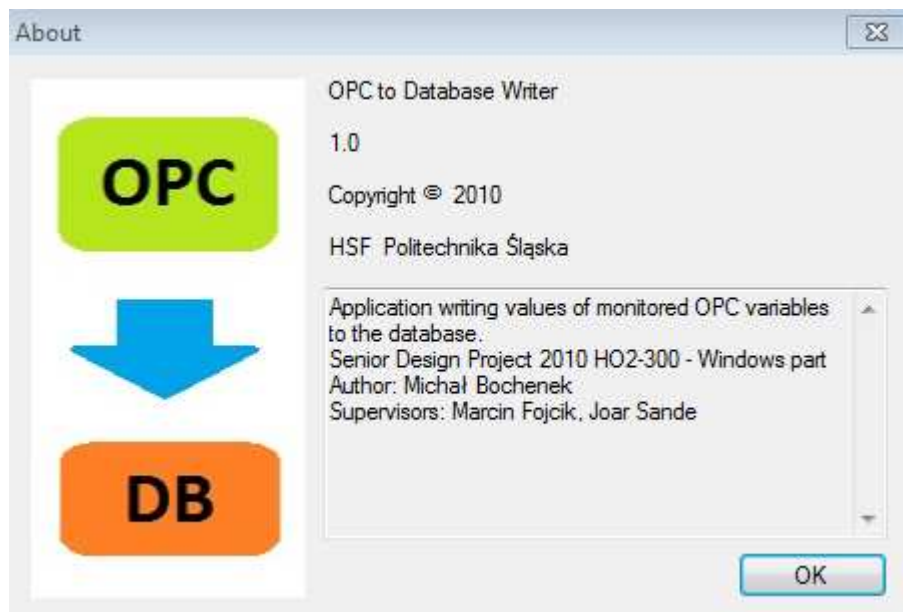


Figure 11 - About form

7.6. Additional libraries

Apart from using the OPC2DB application for local data monitoring and applying the changes to the configuration our project is providing the OPC HDA server for clients who want to use this protocol for acquiring the historical data. As we mentioned earlier in the report the OPC HDA Server SDK has been used as the base for fulfilling this task. However, the provided SDK does not contain any mechanism for reading the data from data storage other than the CSV file. To prevent the changes that would cause any unpredicted errors the best solution we

decided to supply the HDA server with the DLL containing all necessary database connection interfaces. The other reason for such solution was the low level code used in its development that would not be able to cooperate with the C++ SQL library efficiently enough. Functions and procedures included in the DLL are as follows:

int GetNumberOfData(char* tblName) – returns the number of records (rows) in the *tblName* table

void ReadValues(char * tblName, double * resTable) – reads OPC values from the *tblName* table and stores them in the *resTable* array

char ** ReadTimestamps(char * tblName, int dataNum) – returns array of *dataNum* read OPC timestamps from *tblName* table

void ReadQualities(char * tblName, int * resTable) – reads OPC qualities from the *tblName* and stores them in the *resTable* array

7.7. Additional software

OPC HDA server that we included in our project to provide remote access to the historical data already stored in the database using OPC HDA client uses its own XML file with the information about every OPC variable that is offered to the client. In order to keep the HDA server variable list up-to-date an additional console application *updateXML* was required. It has one task – check the variables configured by the OPC2DB application to be monitored and create the XML settings file for the HDA server.

We developed following classes in the application:

```
class varData
{
    String description;
    String unit;
    String varName;
    String dbName;
}
```

Objects of this class store information about every variable – description, unit, name and the related table name in the database.

```
class xmlList
{
    private String DataBaseName;
    private List<String> variables;
    private List<String> attrNames;
```

```
private List<String> attrTypes;  
private List<String> attrValues;  
private List<varData> varsAttrValues;  
  
public List<String> readVarsFromXml()  
private String readDbNameXml()  
private List<varData> getVarAttrValues()  
private void fillAttrNames()  
private void fillAttrTypes()  
private void fillAttrValues(varData currentVarData)  
public void updateHDAXml()  
  
public xmlList()  
}
```

Object of this class is crucial for this application, as its methods perform all the necessary operations for the XML file to be updated.

Its fields contain information about: database name, list of variable names, list of variable attributes' names, list of variable attributes' types, list of variable attributes' values and list of all attributes values' for every variable.

Methods perform following operations:

- Reading the database name
- Reading variable names
- Fetching the attribute values for the variables
- Filling all of the lists with appropriate data
- Creating the new HDA server configuration file

7.8. Further development

We proudly present the fully operational system, however we are aware of undeniable facts that in the future some changes would be made and should be made. The OPC HDA client development is left to the user that would use this kind of software. We enclose, however the sample client in our project, but any custom actions or features needed should be implemented in the future.

Other element of the project that could be changed in the future is the usage of the TopServer. The OPC UA server that is running on the same machine at the moment requires the bridge to the database application and that is the role of the TopServer. However, developers may think of changing the structure of the OPC connection in the project and replace the library provided by TopServer manufacturer with their own set of methods that would make it possible to read the variable values straight from the UA server. Of course,

the changes to the class *OpcServer* containing functions responsible for the communication with the current OPC server will be indispensable.

7.9. Configuring the operating system

Writing the code of the OPC2DB application and additional libraries or applications were the biggest parts of developing the Windows part as for the time spent on creating those elements. However it turned out that it had not been enough for programmes to start working effectively. The problem was lying in the security settings of the Windows system. OPC HDA server was made on the basis of the DCOM model so it required a lot of effort to set the DCOM settings properly for all services enabling data to be accessible through the OPC HDA protocol.

Services that are of utmost importance here is the OPC HDA server itself but also the *OPCEnum* service that browses the machine, searching for all instances of OPC servers running on it, and showing them to the clients, waiting for the connection.

All necessary steps that need to be undertaken to configure the Windows DCOM settings are enclosed in the Appendix C.

7.10. Principle of operation

This point describes how the Windows part of our project really works. It does not include the OPC UA server, as it is the part that should be implemented on the Linux based NPE controller, but at the moment is impossible due to unavailability of the Java Virtual Machine for the ARM processor, compatible to the version for which the OPC stack was built.

When the OPC2DB application is started, the user is asked to specify the name of the OPC server, from which the data will be acquired, name of server from the previous session is entered as default (server information is stored in the *OPCServer.xml* file). Then the list of variables, specified in the *OPCVariables.xml* file is being checked. If the OPC server does not possess any of the variables specified in the file on its list, the user is informed about that fact. The main form is displayed, and after turning the OPC read to the *ON* state the program starts to fulfil its main task, which is writing to the database values read from the OPC server. All variables (tags) specified on the OPC Variables form are entered into the new group subscribed in the TopServer. When the value of any tag in that group changes there is an

event triggered and the OPC2DB application receives the list of new variables' values. Then application checks, which of the tags have been specified to be written in the database (which can be done using *Write* form). All values that are supposed to be written into the database are passed to proper tables and the situation repeats.

When user is entering the new tag to be monitored, a new table in the database is created for that tag, whether the database write option is enabled or not. It is a sort of preparation, as the mentioned option might be selected in future. Information about the status of database write for each tag is also kept in the *OPCVariables.xml* file.

Beside the main functionality, OPC2DB application is also the local viewer of data already stored in the database and their current values. So that, after enabling the OPC read, current value of selected tag is displayed on the main form, and after specifying the time period that is relevant for the user, values, timestamps and qualities are fetched from the database and shown to the user.

Note that settings concerning monitored tags, including all options connected with them are loaded to the memory at the application start, so every change made to the configuration requires application to be restarted.

As for the OPC HDA server, it does not require any attention after configuring the operating system to cooperate with it properly (see Appendix C).

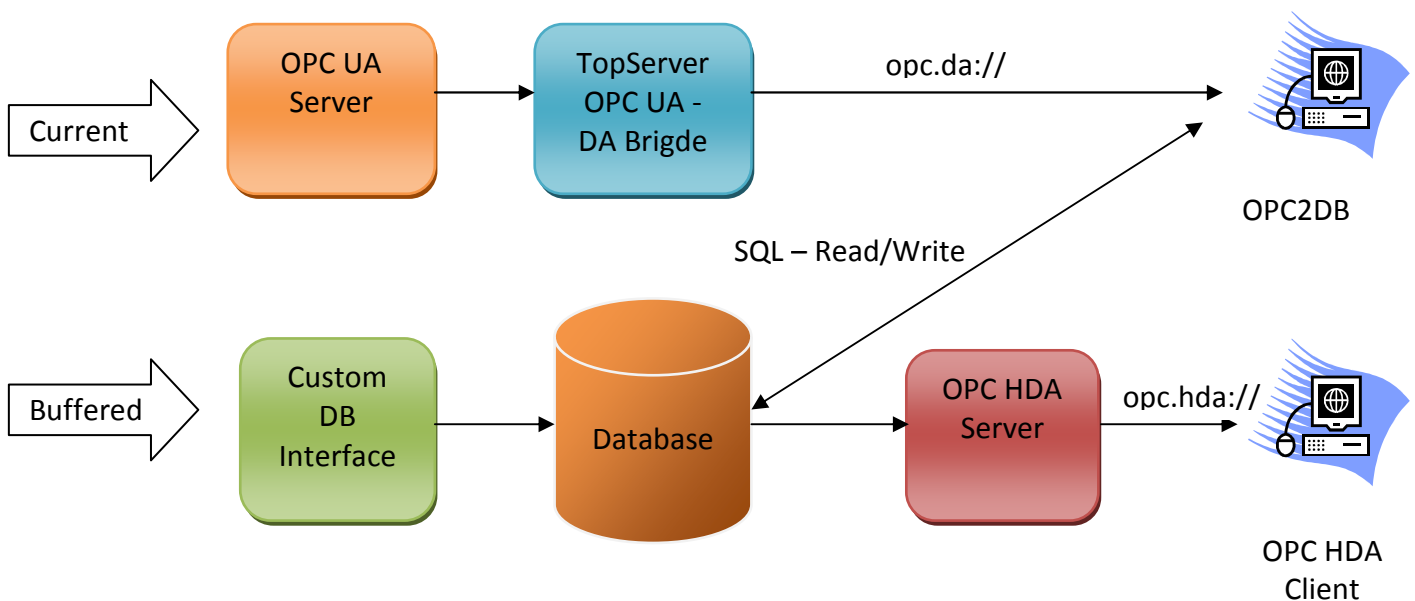


Figure 12 - Windows part's principle of operation

8. Linux part implementation

Linux-based part of the project consists of several sub-parts. First are bash scripts providing access to the Internet, mounting shared folders and setting configuration of NPE controller. Second is Java data gathering application, which is able to use one of existing protocols of operating system and connect to main node of our system. Third part is OPC UA server, which is sharing data gathered from each mediation node. Despite of this part is going to be run on Windows machine, it is considered as one of Linux parts because it was strictly connected with development of software for NPE, is written in portable Java language and can be run on Linux desktop system also.

8.1. NPE scripts

NPE is Linux-based industrial computer, which can be configured almost like desktop computers with Linux system. We prepared some bash scripts[7] that provide access to internet and are loaded into memory with every system start[5]. Below we enclose short reference of most important of them.

- connect.sh

```
#!/bin/sh
touch /tmp/connect_log
echo "Connecting to Telenor EDGE..."
usleep 50000
echo "Logging to /tmp/connect_log started."
cp -f -r /mnt/mtd/ppp/etc >> /tmp/connect_log
usleep 10000
echo "Starting GSM modem..."
gsmon >> /tmp.connect_log
usleep 10000
echo "Applying EDGE connection modem settings..."
modem /dev/ttyS1 230400 "at+ipr=115200" >> /tmp/connect_log
usleep 20000
modem /dev/ttyS1 115200 "at+ifc=0,2" >> /tmp/connect_log
usleep 20000
modem /dev/ttyS1 115200 "at&d0&clvlvs0=0" >> /tmp/connect_log
usleep 20000
echo "Starting PPP daemon..."
pppd call telenor-edge-ttyS1 >> /tmp/connect_log
usleep 20000
route del default
echo "Setting ppp0 as default gateway..."
usleep 10000
route add default gw 10.0.0.1
usleep 20000
```

```
route >> /tmp/connect_log
echo "Done."
```

- rcs.sh

```
#!/bin/sh
#
# Custom System Startup script
# Run once at boot time
#
# Start selected services
/etc/init.d/rcS0
# Put custom actions here
usleep 20000
cp /mnt/mtd/connect /bin
chmod 777 /bin/connect
connect
```

- edge-gprs-connect-chat-telenor.sh

```
#!/etc/ppp/edge-gprs-connect-chat
#
TIMEOUT 5
ECHO ON
ABORT '\nBUSY\r'
ABORT '\nERROR\r'
ABORT '\nNO ANSWER\r'
ABORT '\nNO CARRIER\r'
ABORT '\nNO DIALTONE\r'
ABORT '\nRINGING\r\n\r\nRINGING\r'
SAY "Press CTRL-C to close the connection at any stage!"
TIMEOUT 30
'' '\rAT'
OK 'AT+CFUN=1,1'
#OK 'AT+CPIN=""'
OK 'ATE1\d'
SAY "\nWaiting for logged to GSM network..."
SAY "\ndefining PDP context...\n"
OK '\dAT+CGDCONT=1,"IP","telenor",,"",0,0'
#OK 'AT+CBST=81,0,1;+CHSN=6,0,0,0'
OK 'ATD*99***1#'
TIMEOUT 10
SAY "\nwaiting for connect...\n"
CONNECT ""
SAY "\nConnected."
SAY "\nIf the following ppp negotiations fail,\n"
SAY "try again. Sometimes the waiting time to logged to gsm
network is to short.\n"
```

- telenor-edge-ttyS1.sh

```
#!/etc/ppp/peers/edge-gprs

# Debug info from pppd:
# You can comment this off, if you don't need more info
debug

# Path to modem, you should change this line if your modem is
# connected to /dev/ttySX
# and it isn't linked to /dev/modem. See dmesg after put your modem to
```

```
socket.  
/dev/ttyS1  
  
# Max speed  
230400  
  
# Use hardware flow control  
noctscts  
  
# Don't keep pppd attached to the terminal:  
updetach  
  
# Connection options  
noauth  
user dj  
  
# Path to chat script connect/disconnect  
connect "/usr/sbin/chat -v -f /etc/ppp/edge-gprs-connect-chat-telenor"  
disconnect "/usr/sbin/chat -v -f /etc/ppp/edge-gprs-disconnect-chat"  
  
# IP address configuration  
noipdefault  
usepeerdns  
  
defaultroute
```

8.2. Mediation node application

On NPE computer data gathering program is going to be run. Its main task is to communicate with sensor network, to store and pack the gathered data, to buffer it in case of lack of connection with central node and to send the data when connection is established. For this purpose we designed multi-threaded Java application, which possesses those features. In future this application can be equipped with some artificial intelligence algorithms, which would be responsible for local data analysis.

Below we enclose reference of main functions implemented in this part of our system.

```
/**  
 * Reads and returns new values of OPC variables  
 * @param currentVariables List of variables  
 * @return List of new values of variables  
 */  
public static List getNewData(List currentVariables)  
  
/**  
 * Bufferes given variable in the file  
 * @param variable Variable to be archived  
 */  
private static void archiveVariable(Variable variable)  
  
/**  
 * Gets current configuration of variables  
 * @param filename Configuration file name  
 * @return List of variables
```

```
*/
public static List getVariables(String filename)

/**
 * Gets all nodes with given name from given XML file
 * @param filename Name of file to search in
 * @param nodename Name of node to search for
 * @return Enum with all found nodes
 */
public static Enumeration getNode(String filename, String nodename)

/**
 * Sends give message to server and receives the answer
 * @param message Message to be sent
 * @return Answer returned from server
 */
public String sendReceive(String message)

/**
 * Creates new socket and connects to the server
 * Also creates IO streams using created socket
 * @param host Address of the server
 * @param port Port used for communication on the host system
 * @return True if connected, otherwise false
 */
public boolean connectServer(String host, int port)

/**
 * Performs gzip compression of given string
 * @param str String to be compressed
 * @return Compressed string
 * @throws IOException
 */
public static String compress(String str) throws IOException

/**
 * Performs gzip uncompression of given string
 * @param data String to be uncompressed
 * @return Uncompressed string
 * @throws IOException
 */
public static String uncompress(String data) throws IOException
```

8.3. OPC UA Server application

The OPC UA server application is built using Prosys' Java SDK. It provides all necessary functions creating OPC nodes and address space, allowing to manage OPC nodes and updating OPC variables value, timestamp and quality. We used our own idea of distributed data providing, which minimises traffic via GPRS network by packing sent data. Our OPC UA server is multi-threaded, concurrent application. For every new data provider new thread is invoked. Another threads are responsible for data providing, another for user interface and

another for OPC data sharing. That's why our OPC UA server needs to be run on quite powerful computer.

Reference of most important functions used in this part of our project can be found in the Prosys' OPC UA SDK Javadoc.

9. Organization

Organization of the project is divided into three groups. We can name the ordering institution, supervising group and the developers group.

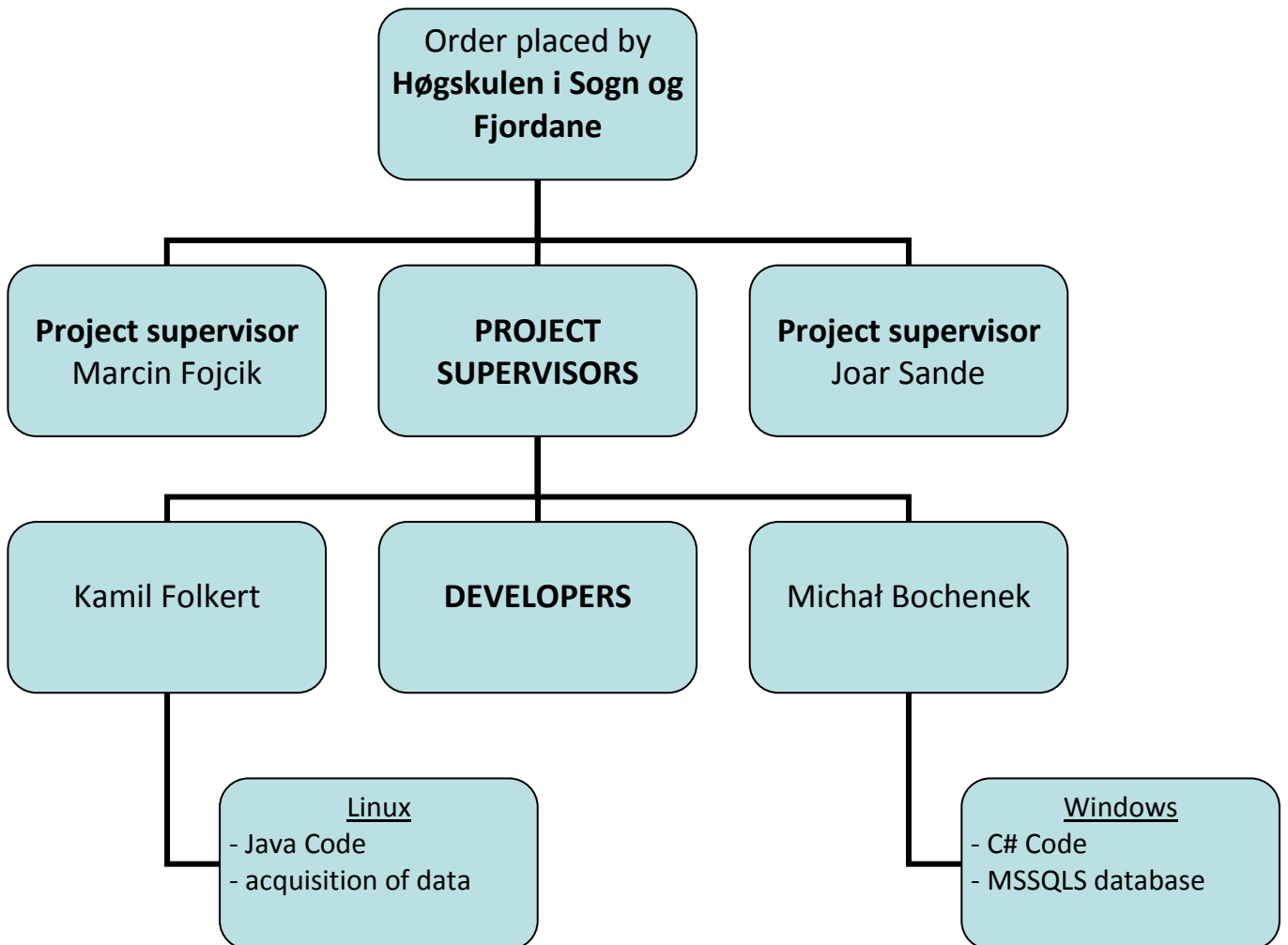


Figure 13 - Project organisation

9.1. Developers

By developers we understand two persons: Kamil Folkert and Michał Bochenek. They are the students of Polish Silesian University of Technology (*pl.* Politechnika Śląska) who came to Høgskulen i Sogn og Fjordane on the STF exchange programme. Their work is conducted within the confines of Senior Design Project on the premises of HSF. As the project is divided into two parts, according to the runtime platform it does not need to have the leader whose task would be the subtask distribution. Kamil Folkert is responsible for the Linux part, and the Windows part is dependent on the work of Michał Bochenek. Additionally each person has to take part in the documentation creation and the website building.

Name:	E-mail:
Kamil Folkert	kamilf@stud.hisf.no
Michał Bochenek	michalb@stud.hisf.no

9.2. Project supervisors

Project possesses two supervising persons, who are: Marcin Fojcik and Joar Sande. They both are employees of Sogn og Fjordane University College. Also, the HSF is the company that placed the order for our project.

Name:	E-mail:	Tel.:
Joar Sande	Joar.Sande@hisf.no	57 72 26 29
Marcin Fojcik	Marcin.Fojcik@hisf.no	57 72 26 70

9.3. Order placement company

Project is being made for the Høgskulen i Sogn og Fjordane that is represented by Marcin Fojcik and Joar Sande. The final result is going to be prepared to be a part of the research programmes made in cooperation with Silesian University of Technology.

10. Project administration

The project has been realized within the confines of school classes in Senior Design Project. The subject code is HO2-300. The subject has a value of 20 ECTS points and took time from 08.03.2010 to 07.06.2010. Each member of the project group spent about 500 hours working on the project.

10.1. Carrying out project according to plan.

Our project can be considered as very innovative, OPC UA still is not very popular and there are not many examples of automation based on this protocol. For that reason our project was developed in sequential order - before every part of work, we had to find documentation of used Software Development Kits, standards and protocols, then we developed the part as stand-alone and in the end we integrated it with previously compiled software. This system seems to be very efficient in such cases when you have to go through lots of documentation and solve various problems with compatibility. During the project development we faced many difficulties, but we managed to carry on due to well-prepared project development plan. In the area of planning Gantt Diagram was very helpful, because thanks to it, we divided time of our work into reasonable periods. As we were supposed, we spent on the project about 500 hours per person.

10.2. Gantt diagram

During the last few decades, when computer science evaluated as an independent science, to improve the quality of intellectual goods developed in its domain, some good manners were invented and introduced into common practice. One of them is defining a project schedule and developing the project according to it. There are many notations used for defining such schedule. The one providing great clearness is the Gantt diagram.

In our schedule we took all the project development phases into consideration and we assigned the time period to every one of them, according to extracted subtasks. We defined time dependencies between project development phases as well. It allowed us to broaden our perspective on the project and to get aware of possible problematic issues. Despite

thorough planning and deep consideration there were, of course, some displacements of time periods assigned to the most complex implementation parts. The Gantt diagram prepared for preliminary project report is an appendix to this document.

10.3. Week schedule

Before we started the work on the project, we created the week schedule. It represented hours which our group should spend on project development. Work on the project proceeded according to the plan prepared at the beginning. That week schedule we enclose in the table below.

Table 2 – Week schedule

Time	Monday	Tuesday	Wednesday	Thursday	Friday
8:30 - 9:30					
9:30 - 10:30					
10:30 - 11:30	Meeting	Project	Project	Project	Project
11:30 - 12:30	Project				
12:30 - 13:30					
13:30 - 14:30					
14:30 - 15:30					
15:30 - 16:30					
16:30 - 17:30					

10.4. Meetings schedule

There were some differences between meetings schedule prepared at the beginning and the actual meetings times. Most of the meetings took place in the right time with two exceptions. The meeting planned for the 26th April and 3rd May were postponed for a week ahead as we did not want to present intended parts of the project not fully operational. Meetings were always called via e-mail to both supervisors. Below we enclose a table showing actual meetings schedule.

Table 3: Meeting schedule

Date	Place	Time	Attending groups
26.02.2010	Linus	12:00 – 13:00	Project group
08.03.2010	Linus	10:30 – 11:30	Project group
15.03.2010	Linus	10:30 – 11:30	Project group
19.04.2010	Linus	10:30 – 11:30	Project group
03.05.2010	Linus	10:30 – 11:30	Project group
10.05.2010	Linus	10:30 – 11:30	Project group

11. Budget of the project

Project was intended for a group of two students which makes about 1000 hours of work. The time we spent on the project was divided into four parts: analysis of problem, implementation, test and document.

To fulfil the given tasks we needed necessary development environment. The Java programme was written using the free of charge NetBeans IDE. The C# programme and the T-SQL procedures required Microsoft tools such as Microsoft Visual Studio and the SQL Management Studio which are the commercial solutions. However, as we are students at the Silesian University of Technology, we are taking part of the MSDN Academic Alliance programme which gave us the opportunity to use mentioned software free of charge for non-commercial purposes. The same rule was applied when using Microsoft Project Professional.

The main cost part focused on the NPE controller. To properly test the system we needed at least three of those controllers. According to the fact that one piece was lent by the Silesian University of Technology we needed to buy another two controllers and the necessary equipment for them, which included: SD memory cards and pre-paid cards, making the transmission over GPRS possible. Overall cost of the system came as the sum of 8400 NOK.

Detailed costs of the project are listed in the table on the next page.

Table 4 - Budget approximation

No.	Requirement	Cost
1.	Two computers - provided by the University College	0 NOK
Linux part		
2.	Programming environment – Eclipse	0 NOK
3.	Power supply – provided by the University College	0 NOK
4.	NPE-9x00-EDGE – Linux Embedded Controller x2 (we have one controller borrowed from Silesian University of Technology, but we need at last two more to test communication, time synchronisation and MSSQL multiple-source data transfer)	2x4000 NOK
5.	GSM/GPRS antenna – provided by the University College	0 NOK
6.	Pre-paid card x2	2x100 NOK
7.	SD memory card x2	2x100 NOK
8.	12V battery x2 – provided by the University College	0 NOK
Windows part		
9.	OPC Server – TopServer Demo Version	0 NOK
10.	Microsoft Visual Studio 2008 Professional	0 NOK
11.	Microsoft SQL Server 2005	0 NOK
12.	Microsoft Project Professional	0 NOK
TOTAL		8 400 NOK

12. General project evaluation

As final result of our project development process we managed to implement complete system providing multiple input and output interfaces, modern data gathering and sharing algorithms. Our system can be used in many various areas of data processing applications. It allows establishing communication channel using local network, Internet cable connection or GPRS/EDGE connection.

The most difficult part of our project was to design and prepare solutions for implementation various input and output interfaces working together. We planned to use OPC UA as data sharing protocol and we managed to do this, but during project development we decided to change the general idea and instead of using multiple OPC UA servers embedded in NPE industrial controller we designed data sharing part of our system as one central OPC UA server with distributed data providing part. We used Java OPC UA SDK which is based on Java 1.6 UA Stack. As soon as JVM for NPE will be deployed by Sun Microsystems or other vendor the OPC server can be embedded in NPE.

We had many problems with OPC HDA server configuration because of complex DCOM security settings in Windows OS. Finally, we managed to find correct sequence and OPC HDA server is fully operational and can be accessed locally or remotely.

OPC Unified Architecture seems to become the most popular protocol for process data sharing. However, many areas of its specification must be improved and updated. There is lack of information about details of implementation process. Most of the books and other materials we found was too general and did not explain more complex situations, like deploying OPC UA server on Unix-based operating systems. We hope that our work would be useful for the next adepts of OPC UA application programming. That's why we provide full documentation of the project, including code documentation for code completion mechanisms for C# and Java IDEs.

Despite the fact that we divided our project development process into two parts, it was very valuable lesson of team work. We had to consult many issues to provide consistent solutions

and many times we tested our project's parts each other out for more efficient errors elimination.

Due to the project development we got great knowledge about various types of OPC protocols, Java programming language and its runtime environments, embedded systems programming and GPRS data transfer providing. We are sure that experience we got with this project is going to be for us a big handicap in the future. We are grateful for anyone who was supporting us during project development, especially our supervisors - Joar Sande and Marcin Fojcik, the exchange programme coordinator - Eli Nummedal, our master thesis promoter - Rafał Cupek, consultants and vendors of outer solutions.

13. List of figures

Figure 1 - Simple relational database

Figure 2 - Relational database terms

Figure 3 - System overview

Figure 4 - Server choice form

Figure 5 - Main form of OPC2DB application (OPC read disabled, no data fetched from database)

Figure 6 - Main form of OPC2DB application (OPC read enabled, data fetched from database)

Figure 7 - Database server settings form

Figure 8 - Database variables' write form

Figure 9 - OPC Variables form

Figure 10 - Add variable form

Figure 11 - About form

Figure 12 - Windows part's principle of operation

Figure 13 - Project organisation

14. List of tables

Table 1: Relational database terms and their SQL equivalents

Table 2: Week schedule

Table 3: Meeting schedule

Table 4: Budget approximation

15. List of appendixes

Appendix A: Gantt diagram

Appendix B: Transact-SQL stored procedures and functions

Appendix C: Configuring DCOM settings for OPC HDA server

Appendix D: Reports from project meetings

16. Reading list

1. OPC Foundation
<http://www.opcfoundation.org>
2. TopServer specification
<http://www.toolboxopc.com/html/specifications.html>
3. Microsoft Developer Network
[http://msdn.microsoft.com/pl-pl/library/kx37x362\(en-us\).aspx](http://msdn.microsoft.com/pl-pl/library/kx37x362(en-us).aspx)
4. Wikipedia (definition for the abbreviations)
<http://en.wikipedia.org/wiki>
5. NPE-9100-EDGE Documentation
6. OPC HDA Documentation
7. GNU manual pages
8. "Database systems" R. Coronel
9. "Professional C#" S. Robinson, O. Cornes, J. Glynn, B. Harvey, C. McQueen,
J. Moemeka, C. Negel, M. Skimer, K. Watson
10. "SQL Server 2000 – Developer's Guide" M. Otey, P. Conte
11. "SQL Server 2000 Programming" R. Dewson