STUDENTWORK

Final report

Efficient data access in OPC UA based systems

25.05.2011

Group:

Paweł Czembor

Przemysław Zieja

Senior Design Project HO2-300 05/2011

HØGSKULEN i SOGN OG FJORDANE

TITLE: Final report	REPORT NUMBER: 6/2011	DATE: 25.05.2011
PROJECT TITLE: Efficient data access in OPC UA based systems	AVALIABLE: http://prosjekt.hisf.no/~11opcua/	NUMBER OF PAGES: 53
AUTHORS: Paweł Czembor Przemysław Zieja	SUPERVISORS: Marcin Fojcik Joar Sande	
EMPLOYER: Sogn of Fjordane University College		

SUMMARY:

Our project has been developed as a Senior Design Project. The order for the project came from HSF. Realisation of the project took place on the Sogn og Fjordane University College in Førde. The main goal was to create efficient ways of accessing historical data stored in database.

SUBJECTS:

HO2-300, Senior Design Project – Final report.

Efficient data access in OPC UA based systems.

1. Foreword

This rapport is final document collecting all information about project we made as the Senior Design Project. Projects title is Efficient data access in OPC UA based systems. Company that placed the order is Høgskulen i Sogn og Fjordane University College , represented by Joar Sande and Marcin Fojcik . Our main goal was to create efficient ways of accessing historical data stored in database.

After three and a half month of intensive work we can say, that project was finished with success. Despite a lot of problems, mainly in Java part, we managed to achieve very satisfactory results. Java part required especially lot of work, because application needed to be designed and created from the scratch.

In order to easily understand some aspects of our work, reader should have some basic knowledge about databases and object oriented programming.

We are really glad that we had an opportunity to work on the project in Førde , as members of international students exchange program. We would like to thank the Sogn og Fjordane University College for giving us this possibility and for supporting us. Time spent on project gave us new experience and new knowledge about OPC UA architecture.

2. Table of contents

3. Abbreviations and symbols

OLE Object-Linking and Embedding - a technology developed by Microsoft that allows embedding and linking to documents and other objects.

OPC OLE for Process Control – communication standard developed by OPC Foundation.

OPC HDA Standards for communicating stored data developed by OPC Foundation.

OPC UA OPC Unified Architecture – the most recent OPC specification.

HA Historical Access – part of OPC UA standard specification.

COM Component Object Model – a standard, developed by Microsoft Corporation, of creating programming interfaces.

DCOM Distributed COM - Extension of COM standard.

XML Extensible Markup Language set of rules for encoding documents in machine-readable form. It is defined in the XML 1.0 Specification produced by the W3C organization.

C++ statically typed, free-form, multi paradigm, compiled, general purpose programming language developed by Bjarne Stroustrup.

C# One of the programming languages, designed for the Common Language Infrastructure, developed by Microsoft.

.Net Software framework for Microsoft Windows operating systems developed by Microsoft. It includes a large library, and it supports several programming languages.

SQL Structured Query Language - database computer language designed for managing data in relational database management systems.

MSSQL Microsoft SQL Server – relational database engine.

MySQL Free, cross-platform relational database engine created by Sun, currently developed by Oracle

PostgreSQL Also called Postgres Is another free and cross-platform database system.

Java Technology developed by Sun Microsystems for creating machine independent software.

DBMS Database Management System – set of applications used to manage data stored in specific database.

SCADA Supervisory Control and Data Acquisition - refers to industrial control systems: computer systems that monitor and control industrial, infrastructure, or facility-based processes.

ORM Object Relational Mapping - programming technique for converting data between incompatible type systems in object-oriented programming languages.

SDK Software Development Kit - a set of development tools that allows for the creation of applications for a certain software package.

HSF Sogn og Fjordane University College

HO2-300 Code of the Senior Design Project subject at the HSF

MSDN Microsoft Developer Network - Portion of Microsoft responsible for managing the firm's relationship with developers and testers.

T-SQL Transact-SQL - Microsoft's and Sybase's proprietary extension to SQL. T-SQL expands on the SQL standard to include procedural programming, local variables, date processing, mathematics, etc.

JRE Java Runtime Environment required to run applications written in Java

Hibernate ORM library attached to Java application in order to support communication with databases.

4. Summary

These days computers are present in almost every aspect of our lives. They control bank transactions, telephone conversations, even street lights. In many cases computer systems are responsible for processing huge amounts of information. One of areas, there efficiency and reliability are critical is industry.

Because production process very often involves cooperation of machine and human, industrial computer systems need to be very efficient. Fact that production processes become more and more complex they produce a lot of information. Controlling whole production line is extremely demanding task for computer system because all produced data need to be processed and if something goes wrong decision about action that need to be taken, have to be made in mater of seconds or even milliseconds. This time constraints are required in order to provide safety.

Economy and providing products of high quality are also very important aspect of companies functioning. Only way to improve technological process is analysis of data in order to find places, where improvements can be made. This creates another important work, that need to be done by computer systems. Collecting very large amounts of data for future analysis.

During our work we concentrated our efforts on creating solutions, that would be able to access big amount of information stored inside database in the most efficient way. As a final result we have two different solutions for accessing historical data in OPC UA[4] based systems.

5. Introduction

During the semester we worked on our Senior Design Project. It was supervised by Marcin Fojcik and Joar Sande who are representing the Sogn og Fjordane University College in Førde. Our project was continuation of work made by our colleagues (Michał Bochenek and Kamil Folkert) last year [11]. We divided our project, titled "Efficient data access in OPC UA based systems", into two parts.

As a result of his work Michał came to conclusion, that OPC Historical Data Access protocol based on existing code, provided by OPC Foundation is not efficient enough to cope with supplying large amounts of data in reasonable period of time [11]. Paweł was working in cooperation with Michał as consultant in order to improve performance of existing OPC HDA C code. His main task was creation of more efficient HDA solution.

On the other hand historical data are carrying a lot of information about industrial processes and its costs. Because of that it would be great to have a possibility to make fast and easy integration of OPC UA[4] platform with corporate infrastructure, in order to get more control over economy of production process. It was the starting point for work on Java based component. This part was performed by Przemysław in cooperation with Kamil Folkert who already gained some experience in developing data acquisition systems using Java platform and OPC standard[1].

Before main project phase, which consisted mainly of development and testing, we have planed our work and divided whole work into two parallel phases: C/C++ development and Java development.
First part was based on Microsoft Windows platform with .net framework[2] installed.

Java code was created under Linux platform and it is be able to use many different database engines, what was one of goals that we wanted to achieve.

6. Main project objectives

6.1. Java and Linux part

Main goal of Java part was creating application, that would be able to work properly with many different DBMSs. To achieve this hibernate library[8] was used. Because such solution have not been tested yet, final tests had to answer, if this approach is efficient enough to use it in industrial systems.

6.2. Windows and C#/C++ part

Main objective of my work was to enable efficient data access through OLE for Process Control HDA protocol to data collected on OPC HDA servers. Such solution could be used in most of nowadays industrial systems based on OPC standards and various operating systems.

The tools and language was chosen because of already existing OPC HDA Sample code written in C/C++ and C# language and destined for Microsoft Visual Studio environment.

There already exists efficient way to access those data by setting up the OPC server and MSSQL server to cooperate and then use standard SQL client application to retrieve data from Historical server. One main advantage of that solution is speed of dedicated SQL client solution. But casual SQL client using simple stored procedures to retrieve data from MSSQL server do not follow OPC HDA protocol at all.

For OPC dedicated clients the difference in amount of time needed to access and process required information is significantly larger than solution discussed in

previous paragraph. Especially when the client application asks server for above thousand of values - where one thousand of values in Process Control Systems is rather small amount of data. In practical environment amount of data is significantly larger than ones described above. In some solutions, for example in geological measurements, number of historical data stored on HA servers can be reduced with no significant influence on whole system, but in some other cases when measurements are need to be done more frequently – like gas / oil refinery, the majority of data stored during one process can appear critical in later analysis or for following processes.

6.3. Possible solutions

Currently presented solution used in OPC Foundation SDK as is described in "Data acquisition system with database using OPC" project [11] appeared to be very ineffective approach to solving the problem of fast data access through OPC HDA protocol.

Even with some improvements that has been made before, the efficiency of presented solution leaves customers in insufficiency of effective OPC HDA solution[11]. There was few ways of dealing with that problem. One was to create completely new library that provides brand new solution of implementing HDA protocol. Second was to modify library in order to improve its performance. The third solution was to work out partially new library modeled with use of existing one.

7. Windows and C#/C++ part implementation

To achieve goal of this part of project I used Microsoft's Visual Studio 2010, Microsoft's SQL Server 2008 and Sample OPC HDA Solution from OPC Foundation. I have designed system with database dedicated to work under Microsoft's Windows – family operating system.

7.1. System architecture

The whole system is based on Microsoft environment and mostly on Microsoft's technologies. Essential parts of systems are:

- SQL Database which should contains historical data, more information about database architecture is placed further, in paragraph **8.3.**
- OPC HDA Server, that can provide historical data for clients.
- OPC HDA Clients, it could be any client application that follows the OPC HDA standard.
- Data Acquisition System that should provide historical data to databases, but it is not a part of this year project. This system was in fact last year project build by Michał Bochenek and Kamil Folkert[11].
- The network connection between every system part mentioned above.

During my work I usually take tests and measurements on local-host only because the main goal was to measure speed and efficiency of data processing and retrieving data from database, which takes place on OPC HDA Server side. For that measurements I needed to try sustain similar physical conditions of hardware and software layer. The simplest way to achieve this was working on local-host.

In the picture below the sample system architecture is presented.

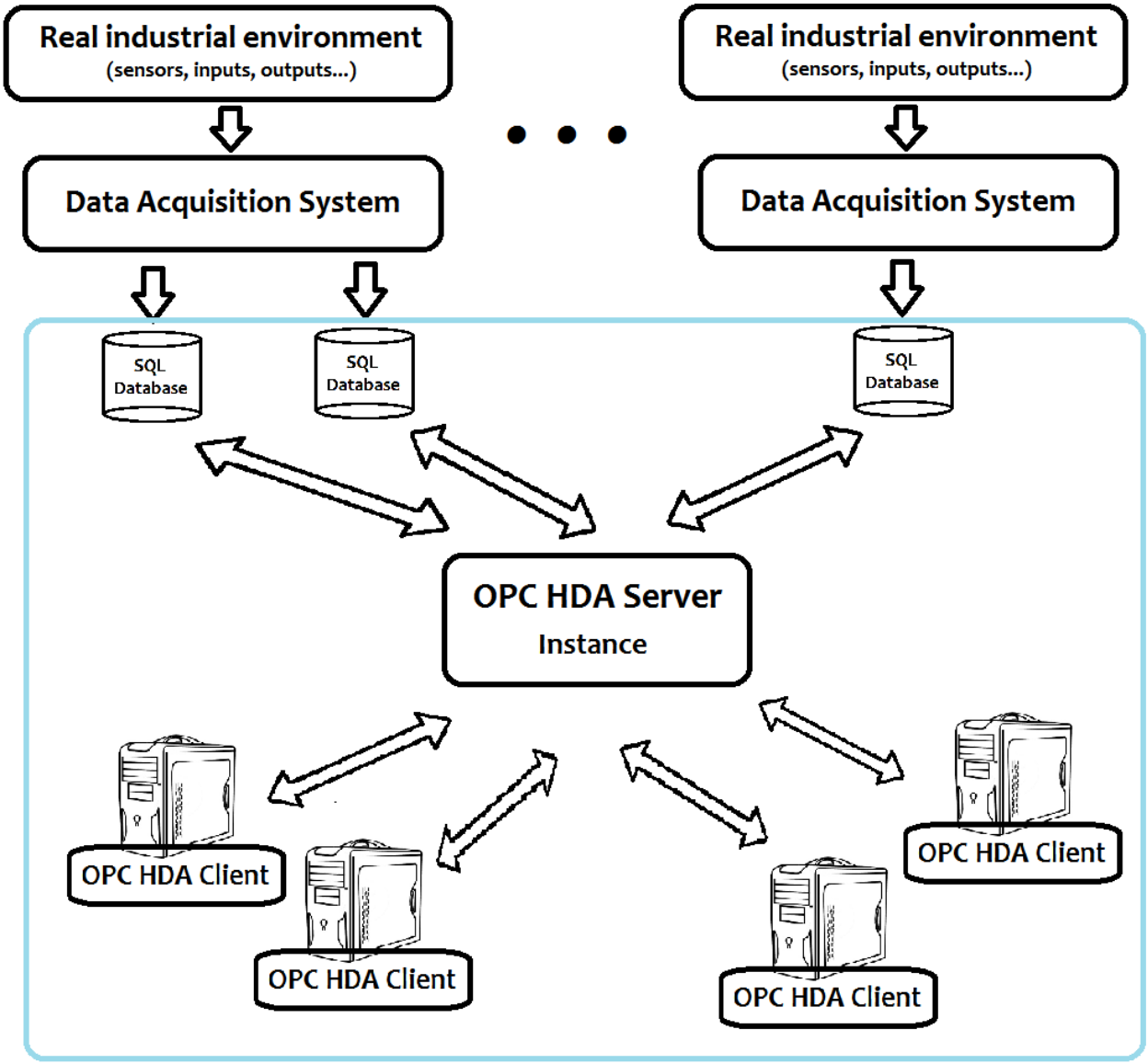

Figure 1 : Windows and C/C++/C# part - System structure

The pale-blue color frame contains the elements of system that I worked on.

7.2. Environment preparation

In order to run the OPC samples run successfully on Windows Vista and Windows7 some special preparations of systems were needed described below. Except installing Historical Data Access sample code there was need to install OPC Core Components as well. Also to enable cooperate with SQL Server some configuration work was required described below.

7.2.1. Operating system configuration

In order to run successfully Sample OPC HDA Server and Client and allow them to communicate with each other usually some changes in system COM/DCOM configuration are needed. Positive surprise was that under Windows 7 after installing whole SDK, the server and client were ready to cooperate for local work and simulations without applying any changes into default COM/DCOM configuration. However network and remote connections required some minor fixes. In second case - Windows Vista needed to apply more changes in COM/DCOM configuration.

The detail instructions can be found in internet or on last-year project website titled "Data acquisition system with database using OPC"[11] made by Michał Bochenek and Kamil Folkert. Since Windows XP the interface and settings have not change significantly so I think there is no need to write down whole new tutorial.

7.2.2. SQL database configuration

To allow remotely connect with database engine all databases which are part of the system need to have enabled SQL Authentication mode, and created special login and SQL User account which possess db-owner role and rights to read and write into specific database (the one which stores historical data from industrial

object). Also to fully cooperate with created solution there are some additional requirements about database structure which are described in chapter **8.3.**

7.2.3. MS VS2010 Solution configuration

While sample solution code was originally designed to work under Visual Studio 2005 there was no problems exporting the solution to VS 2008 format. However while converting to the most recent format – for VS 2010 some errors occurred. Those errors can not be solved automatically by conversion tool, because some reference paths in "OPC Sample Utility Classes", "OPC HDA Sample Server Classes" and "OPC HDA Sample Server" projects properties required manual correction. The developer environment pointed quite accurate where problems occurred so I think there is also no need to write down detailed tutorial for solving this problems.

7.3. Basic SQL database project and preparation

For proper work each SQL database that stores historical data need to have enabled SQL Authentication mode and created created special login and SQL User account which possess db-owner role and rights to read and write into database.

While creating the database project I tried to follow schema presented in OPC sample solution, to keep full compatibility with OPC standard[1] and protocol.

7.3.1. Database schema

The database consist on simple schema. There is required one "Configuration" table and one table for each variable that is archived in database. In that case names of variable cannot repeat.

Below are placed the design and short description of Configuration table and table that stores historical data of specific variable.

- Configuration table – need to be named "Configuration"

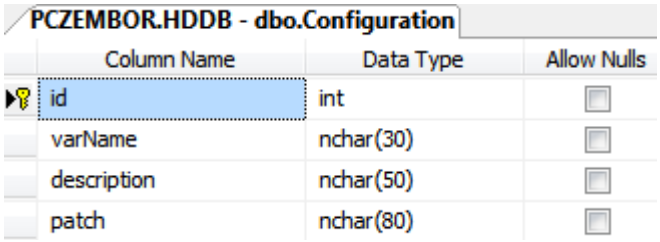

Figure 2 : Windows and C/C++/C# part - Configuration table design

id – identity primary key for table

varName – should contain the name of variable which will be also a table name in database for archiving data.

description – is a short description that can be placed to identify variable.

path – describes the location, and path in explorer tree visible in client applications, also determines item ID inside the OPC HDA Server address space.

- Other tables – tables used to archiving historical data for variables

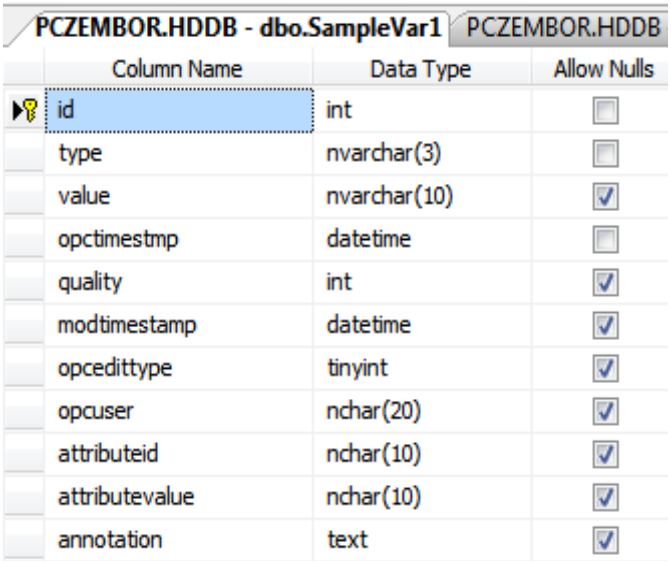

Figure 3 : Windows and C/C++/C# part - Data table design

id – identity primary key for table

type – contains identifier that specifies type of entry value/attribute/modified value/annotation

opctimestamp – value time stamp of OPC server

quality – quality of stored value

opctimestamp – time stamp of value modification

opcedittype – enumeration that identifies one OPC edit type for modified value insert/replace/insertreplace/delete

opcuser – user name that created entry in table

attributeid – OPC HDA Server attribute id

attributevalue – OPC HDA Server attribute value

annotation – unrestricted length text of annotation

7.3.2. Stored procedures

To maintain system consistency I have created also two essential stored procedures that allows to create and drop tables from database. While adding new tables into database there is need to add single entry into Configuration table and while dropping table from database there is need to delete this entry from Configuration table.

- createNewTable procedure

Table 1 : Windows and C/C++/C# part – createNewTable stored procedure

```
ALTER PROCEDURE  [dbo].[createNewTable]
      @tblname NVARCHAR(30) = NULL,
      @vardesc NVARCHAR(50) = NULL,
      @varpath NVARCHAR(80) = NULL
AS
BEGIN
      SET NOCOUNT ON;
      IF (OBJECT_ID (@tblname, N'U') IS NULL)
```

```
            BEGIN
                  EXECUTE ('create table '+ @tblname +
                        '(id int PRIMARY KEY NOT NULL IDENTITY,
                        type NVARCHAR(3) NOT NULL,
                        value NVARCHAR(10) NULL,
                        opctimestmp DATETIME NOT NULL,
                        quality INT NULL,
                        modtimestamp DATETIME NULL,
                        opceditttype TINYINT NULL,
                        opcuser NCHAR(20) NULL,
                        attributeid NCHAR(10) NULL,
                        attributevalue NCHAR(10) NULL,
                        annotation TEXT NULL
                        )'
                  );
                  INSERT INTO Configuration
                     (varName, description, patch)
                       VALUES
                     (@tblname, @vardesc, @varpath)
            END;
END
```

- dropTable procedure

Table 2 : Windows and C/C++/C# part – dropTable stored procedure

```
ALTER PROCEDURE  [dbo].[dropTable]
      @varname NVARCHAR(30) = NULL
AS
BEGIN
      SET NOCOUNT ON;
      EXECUTE ('DROP TABLE '+ @varname);
      DELETE FROM Configuration
        WHERE varName = @varname;
END
```

7.4. Old-sample library analysis

The code that was destined to rewrite were stored in few different modules of server application. Some tests with use of datasets contains 1k, 2k, 4k, 8k, 16k were taken. After measuring time needed to process 8k and 16k values I came to conclusion that tests for 32k values is useless.

Unexpected result of those tests was that the time is not even approximately

linear-dependent on values count.

In addition doing some simple calculations and supposing that HDA server is running on single machine where multiple clients can connect, processing and sending large amount of data can be overloading even for multiprocessor system.

7.4.1. Data processing based on string parsing

Right then the data processing was based on copying all retrieved values into string in memory. The length of this string is one of main factors that determine time needed to process data. After that the string was parsed and divided again into smaller parts to retrieve values that required still conversion to the exact types like integer or double. Main cause of this kind of processing is fact the originally sample data were stored in csv files which did not contain any information about variable/values types.

7.4.2. Main modules to rebuild

Except adding necessary functions to retrieve data from database, some major changes in existing code were required. The most important part of data processing were placed in COpcHdaItem and COpcHdaItemValue classes. My modifications were not based only on rewriting and adding new code but also were based on removing pretty large amount of unnecessary and inefficient code blocks.

7.5. Rebuilding data access and data processing layers

The changes can be divided into three categories: adding completely new code – like creating new projects and methods, changing existing code blocks – including writing new lines in existing classes or removing code block that became unnecessary, and specific changes in server configuration part (this changes do not have any significant influence on efficiency) which changes the

way of use and structure of existing xml configuration files.

Except separate project containing database connection and configuration functions I have created test project to create sample data for database and sample SQL Client project to make measurements at the end of development.

7.5.1. New project and methods

The main new project which I added to solution is named DataBaseCon. It contains three essential functions for whole project. I decided to use standard pointer-arrays to minimize overhead of structures and additional classes.

- `void dbCon::UpdateConfig(char* filePath)`

It contains code responsible for read new configuration xml file and update the xml responsible for reading and creating address space of OPC HDA Server.

- `int dbCon::GetNumberOfData(char* tblname, char* confPath)`

Function used to retrieve information about amount of data requested. First parameter contains unique item Id passed by OPC HDA Server instance and the second parameter contains path to configuation file which is needed to create connection to the specific database.

- ```
  void dbCon::ReadItemValues(char* tblname, char* confPath,
  char** itemTypes, double* itemValues, long long *itemTimestamps, int
  *itemQualities, long long *itemModTimestamps,
  short *itemOpcEditTypes, char **itemOpcUsers,
  unsigned long *itemAttributeIds, char **itemAttributeValues,
  char **itemAnnotations, int itemCount)
  ```

Function used to retrieve all data about item requested. First parameter contains unique item Id passed by OPC HDA Server instance and the second parameter contains path to configuration file which is needed to create connection to the specific database. Another parameters are lists of values to read. The last parameter is item number required to read.
  ```

I also created project named "SQL Data Creator" used for testing and creating sample data for database, and windows form project "SQL Client" used to take measurements at the end to compare efficiency.

7.5.2. Changes in configuration

I have decided also to change configuration of OPC HDA Server to create more complex solution that can connect to multiple databases. Now the configuration file is named "SQLConfig.xml". It contains only information about databases that OPC HDA Server should connect to, instead containing all parameters of each variable accessible by OPC HDA Server. The structure of configuration file is simple so adding another server to browse boils down to add few new lines to xml file. Sample content of configuration file is presented in the frame below.

Table 3 : Windows and C/C++/C# part – Sample configuration xml file

```
<?xml version="1.0" encoding="utf-8" ?>
<Servers>
  <BrowseElement ElementName="Test on PCZEMBOR">
      <host>PCZEMBOR</host>
      <dbname>HDDB</dbname>
      <persist>True</persist>
      <user>appdev</user>
      <pass>appdev</pass>
  </BrowseElement>
</Servers>
```

7.5.3. Changes in existing code

In order to build working solution also some major changes in existing code were needed. The following functions in following modules were modified or added.

In "OPC HDA Sample Server Classes" project, in "CopcHdaItem.cpp" file following functions were modified:

- `void COpcHdaItem::Init()`
- `bool COpcHdaItem::LoadDataFromFile(const COpcString& cFileName)`
- `void COpcHdaItem::LoadData()`

In "OPC HDA Sample Server Classes" project, in "CopcHdaItemValue.cpp" file following functions were modified:

- `bool COpcHdaItemValue::Parse(COpcHdaItemValue& cValue, double dblVal_c, int qual_c, long long timestamp_c)`
- `bool COpcHdaModifiedValue::Parse(COpcHdaModifiedValue& cValue, double dblValue_c, long long llTimestamp_c, int dwQuality_c, long long llModificationTime_c, short eEditType_c, char * cUser_c)`
- `bool CopcHdaAttributeValue::Parse(COpcHdaAttributeValue& cAttribute, long long llTimestamp_c, unsigned long dwAttributeID_c, char * cValue_c)`
- `bool COpcHdaAnnotation::Parse(COpcHdaAnnotation& cValue, long long llTimestamp_c, char * cAnnotation_c, long long llCreationTime_c, char * cUser_c)`

In "OPC HDA Sample Server Classes" project, in "COpcHdaHistorian.cpp" file following functions were modified:

- `bool COpcHdaHistorian::Start()`
- `bool COpcHdaHistorian::Load(const COpcString& cFileName)`

In "OPC HDA Sample Server" project, following files were modified:

- SQLConfig.xml
- OpcHdaServer.config.xml

Also some minor changes in different modules and places were applied – like change of predefined variable values or enumerates, but this changes do not have impact on functionality or efficiency.

7.6. Library after rebuilding

Originally I planned to try different approaches to optimize efficiency of this solution. At first I have tried to optimize existing methods and follow existing solution but after first stage of improvements efficiency was still terrible when

dealing with large amounts of data. Because of that I jumped this stage and decided to write new solution, that was already presented above.

New implementation of data processing is based on SQL queries and specific-type lists that allow to determine what kind of data/variable is currently read.

Knowing what data is processed and heaving values in separate structures allows to improve greatly efficiency of processing data right before the data is send to the client.

First measurements were taken. Those results indicates that now time needed for client to receive large amount of data has been shortened by about 100 times.

Reaching that level of efficiency points that the further optimizations will improve efficiency insignificantly comparing to results that have been already achieved.

7.7. Building SQL Client and taking measurements

As I mentioned before I have created simple Windows Form application to retrieve data from database with use of stored procedures. Unfortunately I do not have access to any free or trial version of OPC standard[1] based products. Existing solutions that can be compared with created system are expensive and they are inaccessible in any trial or demo versions.

Because of this fact the only reasonable comparison seems to be simple SQL Client that do not follows OPC HDA Protocol but is probably most efficient existing solution on the market.

However still the SQL queries are the heart of my solution so beating SQL Client in this competition should be physically impossible. SQL Client in this case can be rational factor of efficiency.

7.7.1. SQL Client application

The SQL Client application was placed in separate project. It contains simple interface that allow to automatic save of measurements in text files. Of course the times measured do not contain time needed to display data on screen because input and output operations are the most time-consuming operations in nowadays computing and they do not have any influence on speed of data receiving in physical way.

7.7.2. Measurements and comparison

My measurements were taken in three points. One was before rebuilding library, second was at the end of project when the library and system was improved, the third was after creating SQL Client and taking measurements of "clear" SQL based application.

Fast SQL client can allow to measure overhead created by OPC HDA protocol. There is no chance that my solution will be faster than regular SQL client because it's core is still based on simple SQL queries, but I expect that data processing overhead after acquiring data from database wont be very significant.

As a factor I accepted time needed to retrieve and process specific amount of data. The amounts of data for measurements were 1, 2, 4, 8, 16 and 32 thousands.

I placed the end calculations on charts below.

Results before rebuilding library:

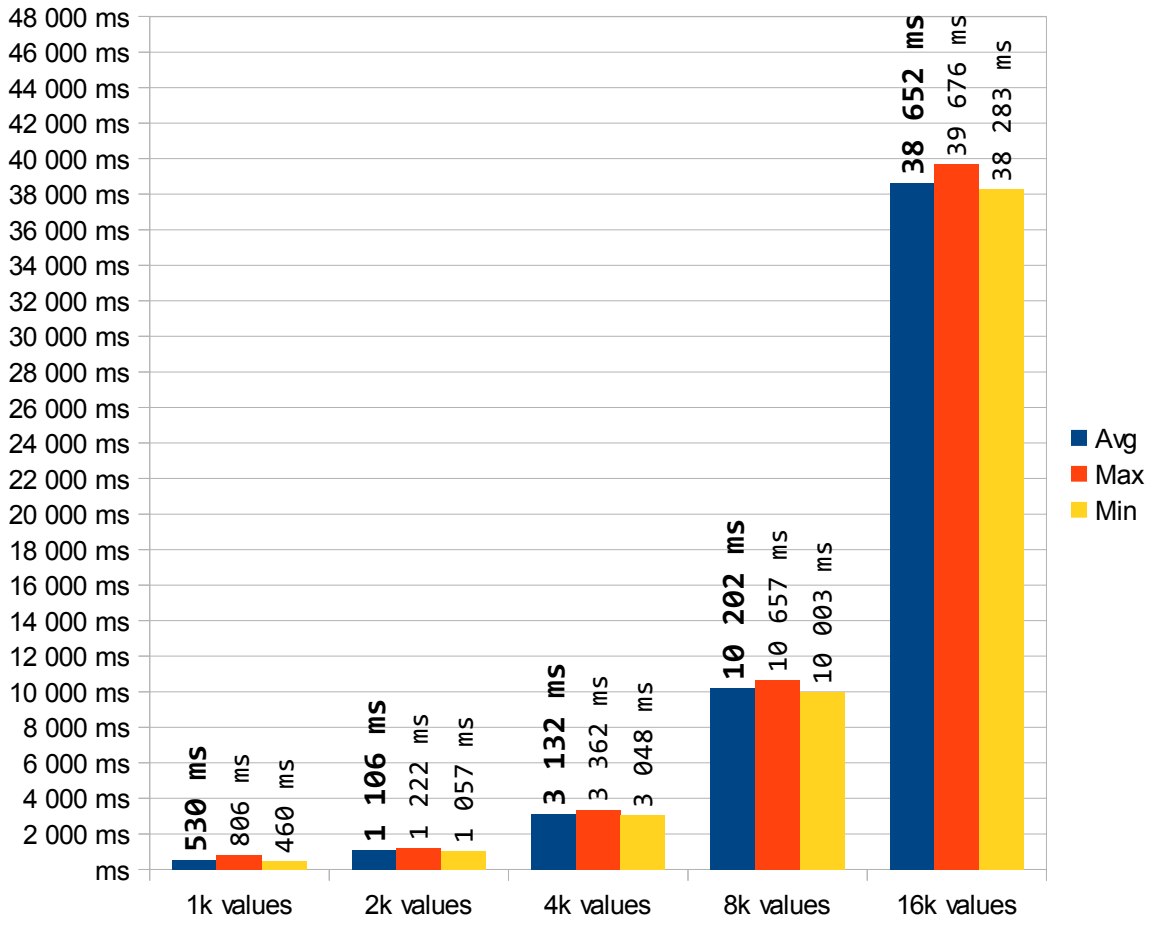

Figure 4 : Windows and C/C++/C# part - Chart 1

The measurements were not taken for 32 thousands of values because of too large amount of time needed to do that. As it is presented the conclusion was that: for above 4 thousands of values old system can be useless especially when more that few clients need to receive data from server. Moreover charts proves that the amount of time needed to retrieve data from server is not linear dependent on amount of data.

Results after rebuilding library:

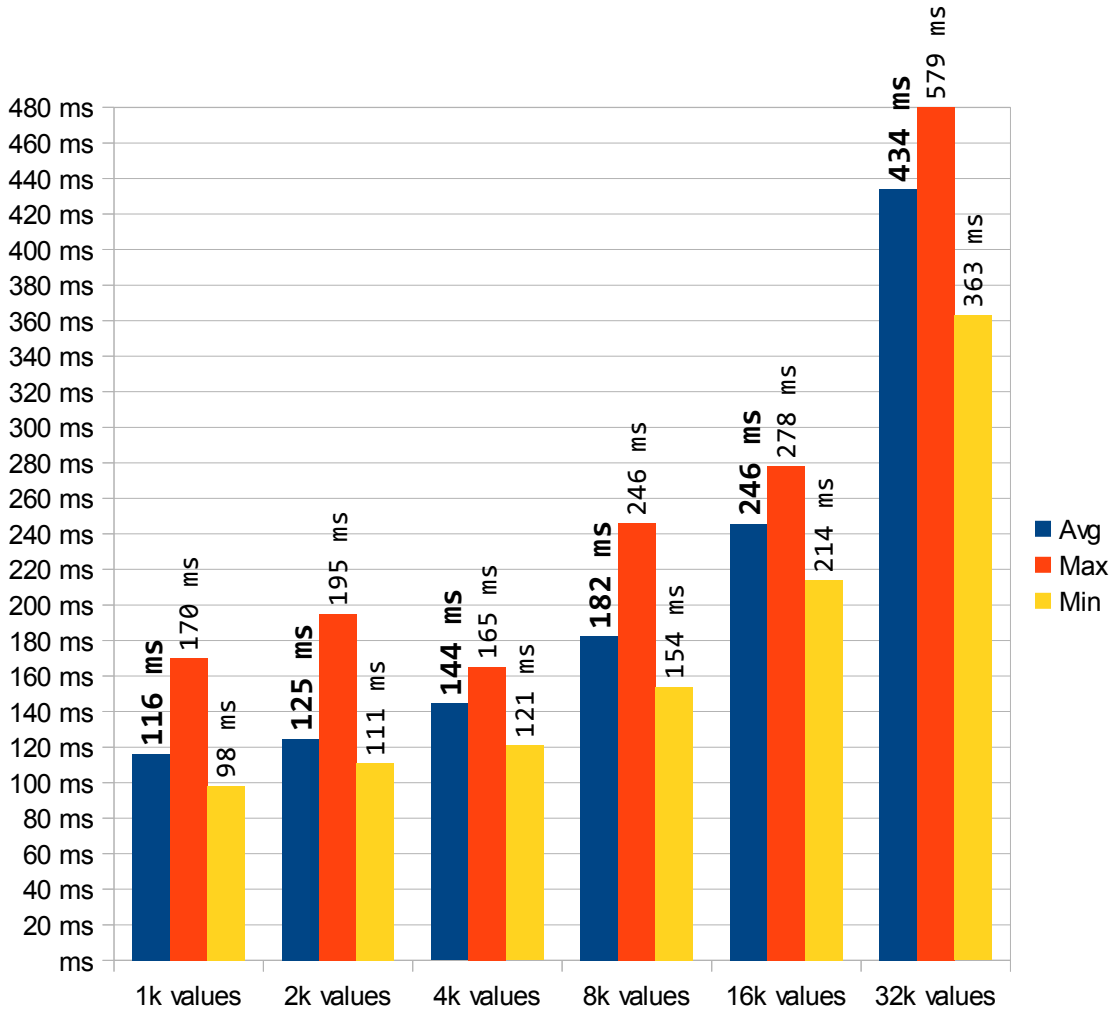

Figure 5 : Windows and C/C++/C# part - Chart 2

As it is presented the efficiency have raise significantly. The speed up allowed to take measurements also for 32 thousands of values. On this chart can be noticed that amount of time needed to retrieve data is in different relation with amount of data requested. To investigate further this dependance I have taken some calculations which rule out constant amount of time needed to establish connection with database and send SQL command.

Chart below is presented in logarithmic scale to present that after library rebuilding the dependency between amount of time and amount of data is linear.

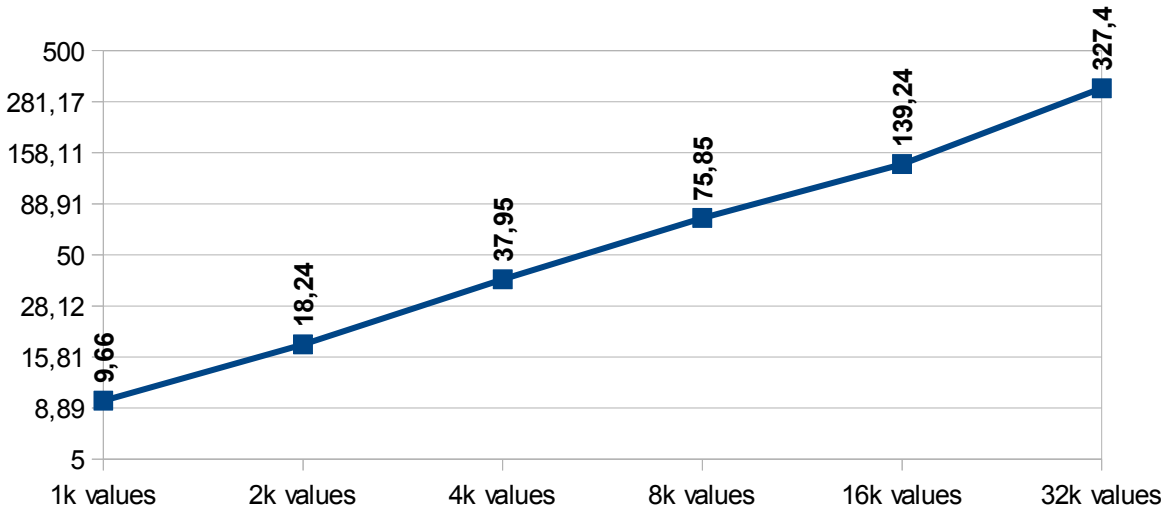

Figure 6 : Windows and C/C++/C# part - Chart 3

Results of SQL Client measurements:

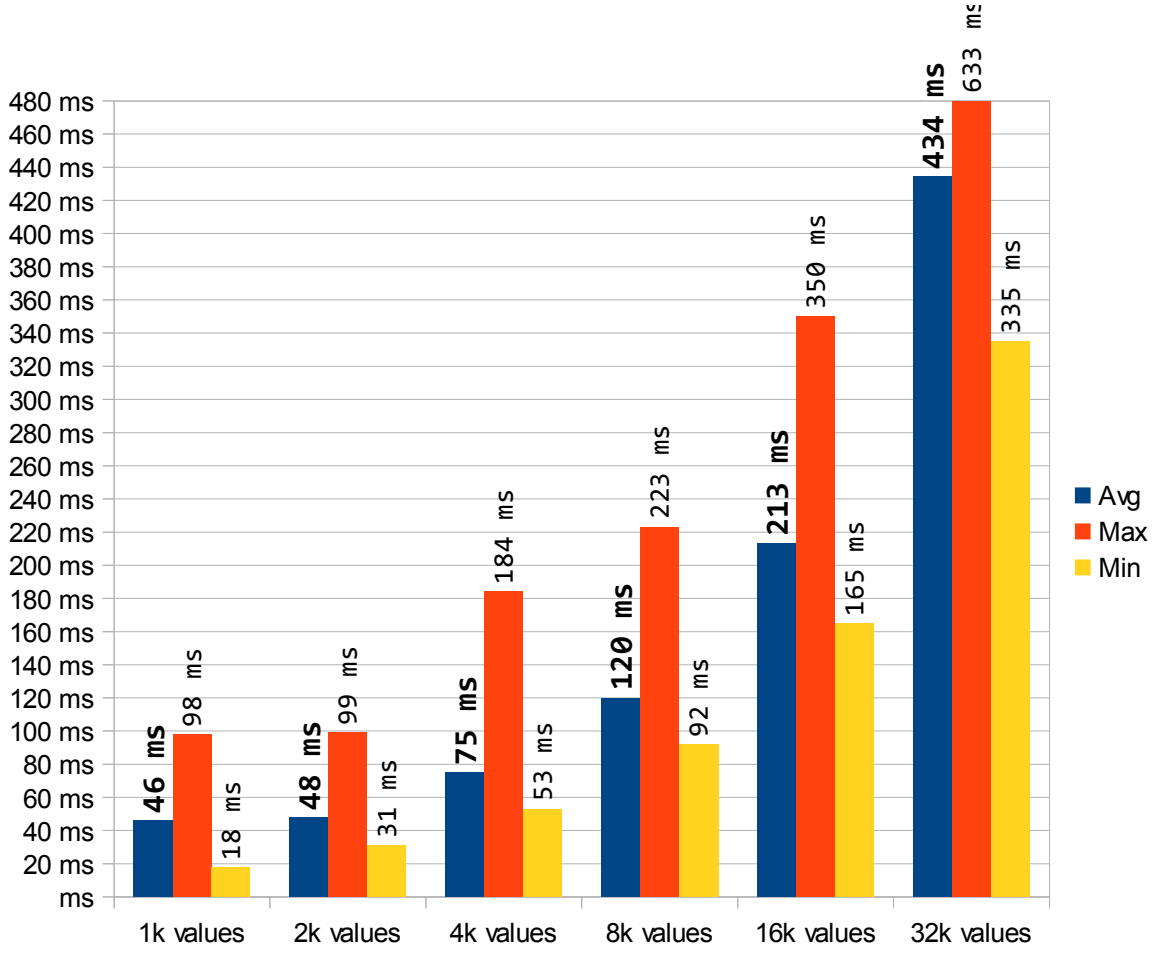

Figure 7 : Windows and C/C++/C# part - Chart 4

As it is presented the efficiency is even better. As I predicted it was not physically possible to beat SQL Client. Comparing to previous results the efficiency of modified system seems to be satisfactory.

7.8. Summary and conclusions

Windows and C++/C# part can be considered as finished successfully. The taken measurements points that the rebuilding existing library while having impact on efficient data access and data processing, can enable efficient and fast data access to historical data stored in SQL servers while still preserving OPC HDA Protocol.

7.8.1. Efficiency of created system

Analyzing the data collected during development of project, we can say that main goal was achieved. Efficiency reached can be compared with standard SQL database based systems what means there should not be any problem while using similar system in real environment. Also it proves that overhead introduced by OPC HDA Protocol is not so significant if it is implemented in proper way. Calculating speed up factor after library rebuild indicates that even when cooperating with small amounts of data new library is almost 5 times faster and when there is need to request larger amount of data like 16 thousands of values the speed up factor jumps to above 150 what is in my opinion more then enough to prove that created system can meet requirements of modern systems which need this kind of data storage.

7.8.2. What else can be done

Still this system can be developed further. Maybe not with efficiency impact, but there is still some features left to implement. Of course the minimal functionality specified by OPC HDA standard was implemented however still some of system aspects like for example authorization of single system users can be resolved.

8. Java and Linux part

While C# part was oriented mainly on finding and eliminating bottlenecks in already existing solutions, in Java part I took completely other way to find solution. My main goal was to create OPC UA[4] historical data server that would be able to store data in database using hibernate library[8] and then check, if this solution is efficient enough to be used in industrial systems. Hibernate was created as a ORM tool for Java language. Because it was natively designed and created for cooperation with Java, I also used this language. Huge advantage of

Java is fact that it is supported on most of operating systems. Because of conception of language, application written in proper way, can be run on every operating system with JRE available. Prototype of OPC UA[4] historical data server I created can cooperate with most of databases on the market. In order to make easier to understand how application works, first I have to say a little bit about the environment, it is created to operate in.

8.1. System structure

Nowadays industry and production is very important to the economy. Industrial processes are getting more complex and control over them is far more difficult than few years ago. That is why computers are involved in this process. They can collect data extremely fast, process them and take appropriate actions. But still we can not say with 100% certainty that there will not be any problems with technological process. In case of any problem with final product, manufacturer need to have possibility of looking on history of process. This is one of the reasons why the critical information about process are collected.

One of the biggest problems in every system that requires data exchange is diversity of products, communication protocols and software tools. It is very common that producers are using their own standards, so their hardware can cooperate only with other devices produced by them. As result the designer need to choose one of wide range of possible solutions and decide which devices or protocol is the best choice. Of course "the best choice" means the best solution in moment of designing the system and if there will be any better components in future it is very likely, that replacing devices in already existing design will be very difficult, inefficient or even impossible. Also in case of expanding functionality of system there can be no such possibility at all or they can be very limited. The same situation is in case of databases which also are

present in considered systems. Although there is SQL standards that regulate most aspects of storing data in relational databases there are some differences in implementation of SQL between DBMSs. This leads to situation in which choosing one particular DBMS in application design stage narrows future possibilities of changing database engine.

OPC standard[1] solves the problem of cooperation between devices from different producers and both exchanging data on hardware level and between hardware and software layers, as well. It defines set of interfaces, services and communicates that need to be implemented in order to make use of simplicity and freedom in connecting any possible device or software that supports OPC.

Nowadays there is tendency to write program basing on object-oriented paradigm. It means that everything is treated as object witch properties and operations that can be executed on it. Because idea of storing data in databases it may be necessary to split very complex object on program side into many tables on database side, what very likely will create problems in implementation. It also requires programmers to build in database queries into source code. In context of slightly different SQL dialect used in miscellaneous DBMSs, embedding SQL code permanently into application practically leaves no opportunity for changing database used, without complex and error-prone process of rewriting application code. In order to cope with all this problems connected with storing data, in 2001 Gavin King started Hibernate project. In general its job is to deal with mapping application data stored in objects into database structure with use of ORM (Object-Relational Mapping). After embedding hibernate[5] classes into Java application, developer can easily configure it through XML files so potential changes of database structure or DBMS will not cause any problems.

If we look at mentioned problems from perspective of company or any project

that requires exchanging and storing data we will conclude, that we can not afford for limitations that comes with specific devices, data exchange protocol or storage engine. Since we have OPC standard[1] that solves our problem on hardware level and hibernate that reduces complications on software level, natural thing is trying to connect them and create solution that will be deprived of mentioned drawbacks.

During last months I have been working on implementation of OPC UA[4] historical data server. At the very beginning I have made some assumptions:

- user should be able to store data in many places (distributed storage)
- particular nodes of network should be independent
- application should be able to collect data from many different sources at the same time

Figure 8 shows the simplest structure of the system. And so, we have two instances of created application (rounded boxes), OPC UA[4] client and database. "Cloud" symbolizes network connection (connection between program instances is also made via network, but I decided to connect them directly to make the idea easier to understand). Numbers near the arrows symbolize next steps of reading data by the client application.

8.1.1. Distributed storage

In real world systems should be as flexible as it is possible. During design and exploitation process we do not want to deal with many limitations. Because of that I have decided to write application in a way that allow user to decide where he want to store data. It is also helpful in case of failure of main storage node. If for example connection with database would be lost operator can easily switch to another database.

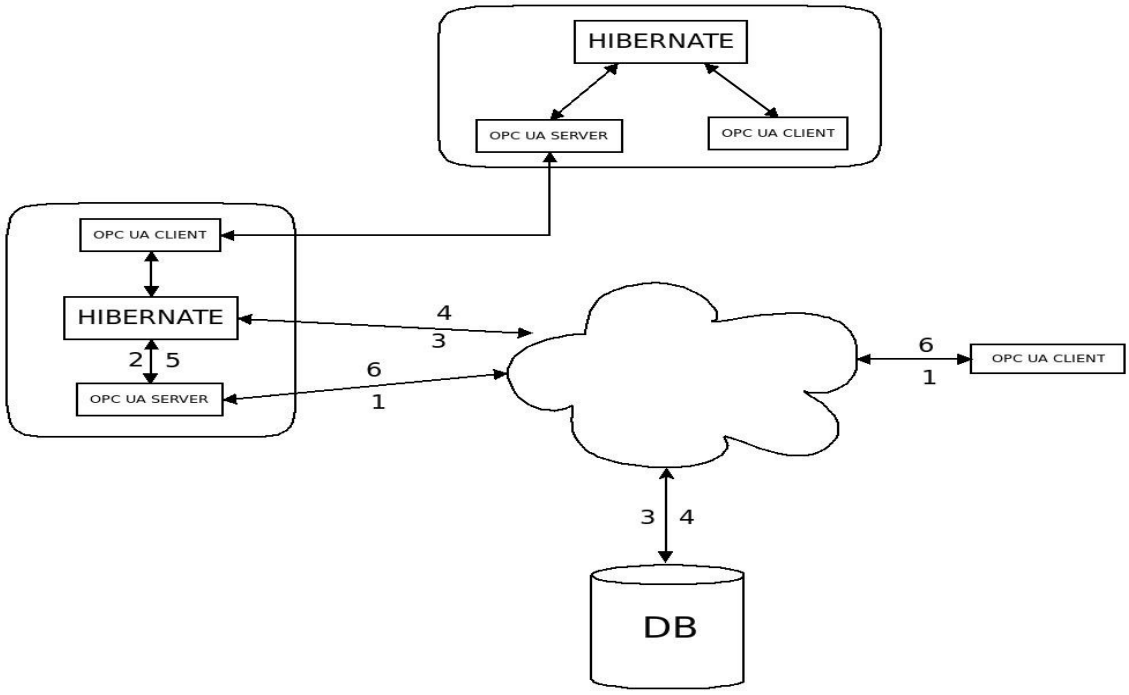

Figure 8 : Java part - Simple system structure

8.1.2. Nodes independence

Program is design in a way that makes it possible to divide functionalities between many units. We can create fully functional system with nodes dedicated to deal with particular tasks, what makes the maintenance cheaper and simpler. Such approach also creates possibility to utilize hardware resources in a more efficient way. For example, we can have one server station with database running

on it. Its only task would be storing data. On another station we can run our application, which will take care of managing incoming connections and providing clients with data they requested.

8.2. Used tools

During development I used Java language with some additional tools and libraries. Basic and the most important of all was Prosys OPC UA Java SDK in version 1.1.2-1941. This software development kit warps OPC UA Java stack and takes care of may aspects of OPC standard[1]. It makes development process a lot simpler and faster. Because of that I did not have to think about communication, session management or other aspects of the functioning of the application and I could concentrate on implementation of historical data management. All configuration information are stored in XML files so in order to read them I used JDom[9] and dom4j libraries. One of crucial functionalities that had to be implemented was ability to cooperate with many database engines. I managed to achieve this goal by using hibernate library to separate functional part of code from implementation of interaction with specific database. As a result for application it is irrelevant what kind of DBMS is used to store data. Development platform I used was Ubuntu Linux 10.04 with Eclipse Helios IDE. At the beginning of project I decided to use two different free database platforms, MySQL 5.1.41 and PostgreSQL 8.4[10]. References to all used tools can be found in reading list .

In OPC UA[4] all information are modeled as an objects of some structure. Description of this structure are stored on the server side in XML file. Unfortunately I do not have access to specification of this XML file and because of that I could only store basic types of data. In application I have implemented support for four data types: integer values, floating point values, double precision

and boolean values.

8.3. Structure of databases

I used two different databases to separate collected data from information required by application. If user want to store all these information in one database he need to create database with tables from both structures described below.

8.3.1. Main database structure

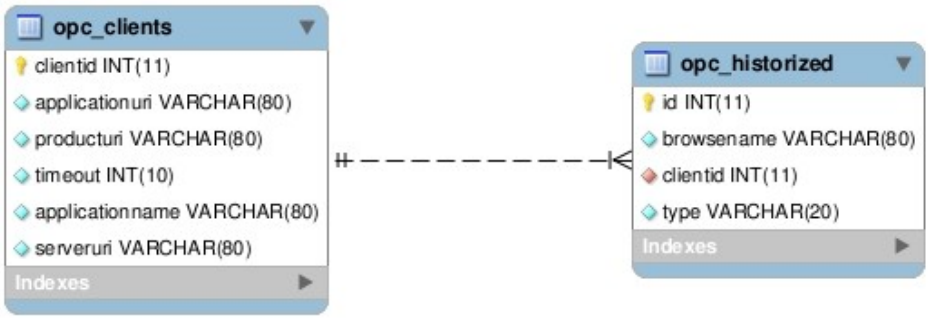

Figure 9 : Java part – Main database structure

Figure 9 shows tables of main database. Connection with this database is required for application. If it is not reachable application will not work. In opc_clients table application stores data about configured clients. When user creates new connection with remote server, he has to put information that are used to initialize client responsible for handling these connection.

clientId – is locally assigned identification number

applicationuri – is application URI address (more details about URI can be found in OPC UA Specification[1], part 3)

producturi – is product URI

timeout – defines the default communication timeout that is used for each synchronous service call

applicationname – name of client

serveruri – is an address of destination server in URI format

Also more information can be found in documentation of used SDK and application documentation.

In order to inform clients about values which history is stored on the server, application stores those data in second table.

id – id of stored element

browsename – string used to identify stored node

clientid – id of client which gathered history of element

type – type of data

OPC standard[1] defines term "address space". If we have two servers, each of them has its own address space with complicated structure of nodes. It is very likely, that on both servers we would have node with the same browse name as node on the other server. In the figure 4 we can see exemplary system with one history data server and two serves from which data are collected.

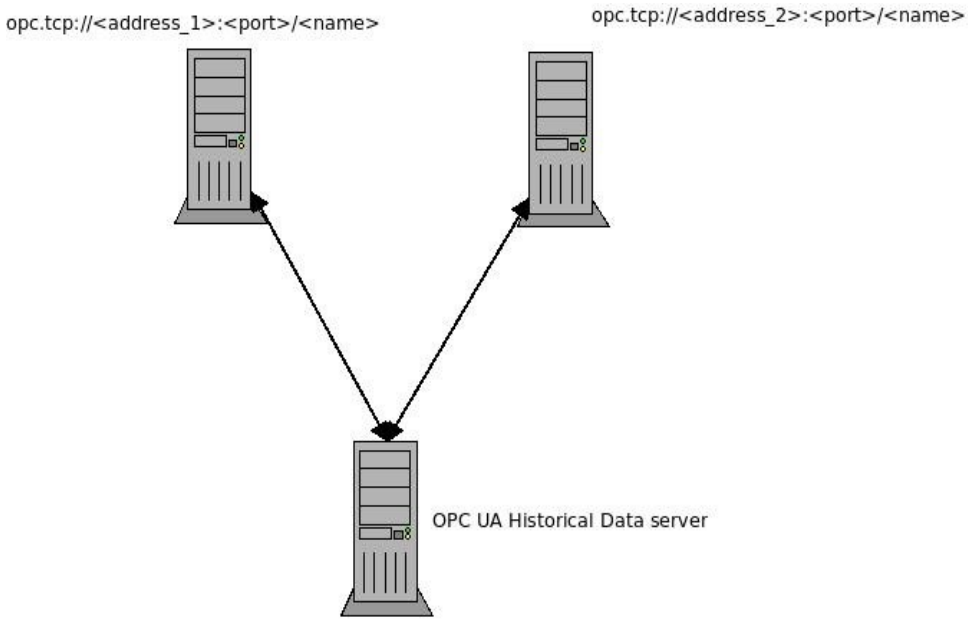

Figure 10 : Java part – Address space issue

Each of servers has different address (address_1 and address_2). Because our network can be quite big, we cannot have two servers with identical network address, this address points directly to only one point in network. OPC standard[1] tells us that browse name is unique among address space, so it corresponds to only one node. I used those two rules to create unique identifier of the node.

Pair <servers URI> and <nodes browse name> gives precise information about source of data.

8.3.2. Historical data database structure

Because of reasons described earlier structure of this database presented on figure below is rather simple. It has only four tables, one for each supported data type.

In general structure of tables is the same. Only difference is type of stored data.

value – is actual value of variable

timestamp – information about time, when value was read

staruscode – statuctode of value gives information about quality of stored value

browsename – browse name of variable

modified – tels us whether value was modified

clientid – id of client that stored this value

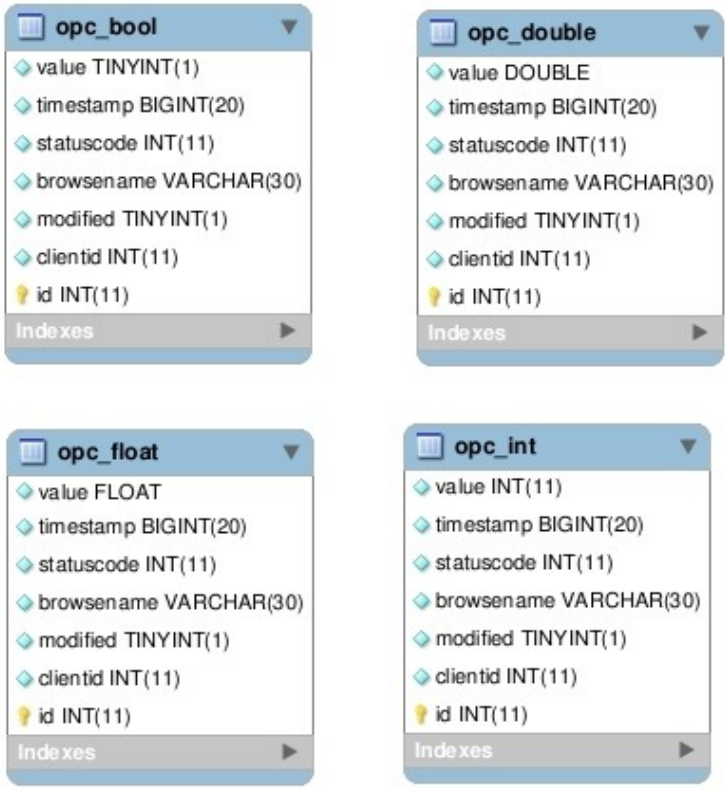

Figure 11 : Java part – Data database structure

8.4. Program structure

Figure 12 placed below show more details about program structure. As we can see application consists basically of OPC UA server and main database. In the picture "Data DB",in which collected data are stored, is separated from "Main DB" but there they can be easily connected in to one database.

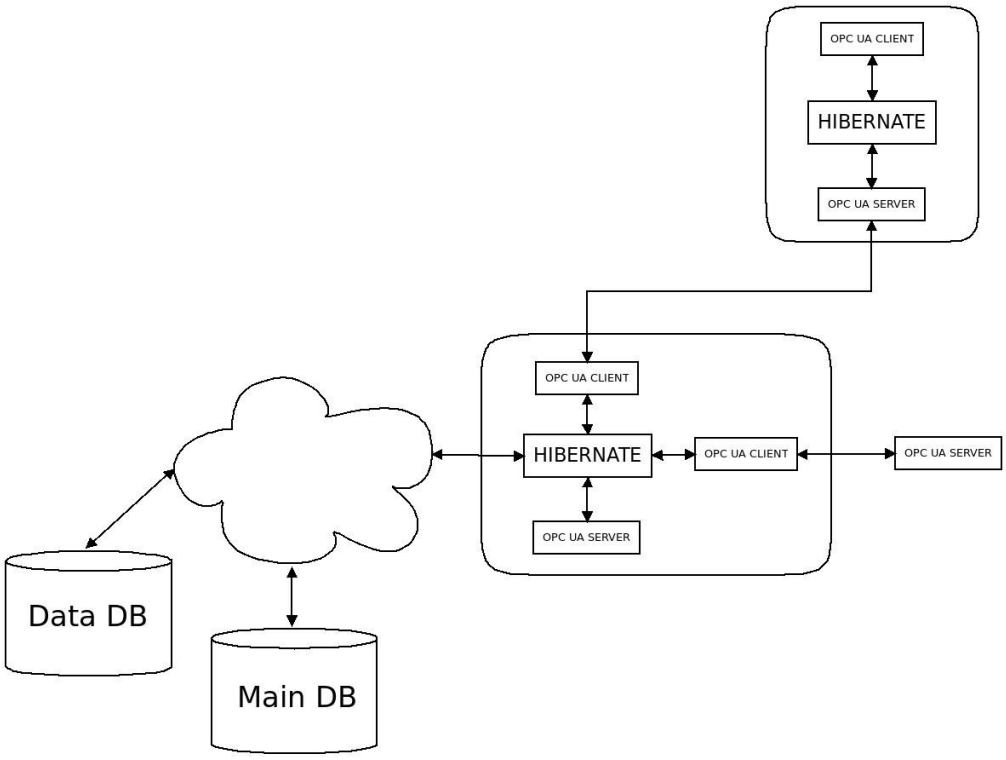

Figure 12 : Java part – Program structure

8.4.1. Hibernate mapping

In OPC UA[4] standard information are modeled as an objects. Because specification was created for industrial systems where every value is associated with measurement or device this approach is natural. On the other hand most popular databases on the market are using relational model to store information. Because of different way of describing reality we need additional tool to map objects to database structure. Hibernate library is exactly this kind of tool. Using XML files to store detailed information about correlation between object and

database structure, hibernate[5] generates SQL queries to manipulate the data. Another extremely important and useful functionality of hibernate is ability to work with 23 different DBMSs. It gives great flexibility during design and exploitation of system. Administrator can simply change database engine, edit configuration files of application and that's it. Because database is separated from functional part of application with hibernate, there is no need to modify source code when different DBMS is used.

Application is able to work with many storage nodes at the same time. To make it possible I wrote DbMapper class that is responsible for mapping process. It extends Thread class in order to run independently from the rest of program. During startup there is created instance of that class for each configured database, so every connection is supported by its own thread.

8.4.2. OPC UA server

Created program is responsible for collecting data from many sources, saving them into database and then, when any client want to read them, application should provide it with requested data.

OPC UA server build in application is responsible for all functionalities that OPC UA server should have. More detailed information about functionalities of server are contained in OPC Specification.

Developer of SDK made it very easy to implement methods that are responsible for certain actions. In order to react properly to clients data request I had to implement method:

```
public void onReadHistory(ServiceContext serviceContext, NodeId nodeId,
            UaNode node, NumericRange indexRange, byte[] continuationPoint,
            HistoryReadDetails details, HistoryData historyData,
            DiagnosticInfo diagnosticInfo) throws StatusException
```

in `HistoryManagerListener` interface. When server receives request from client it creates appropriate query in HQL (Hibernate Query Language is slightly modified SQL). As a result of possibility of storing data in many places it's possible, that history of one value can be stored in many databases. To assure that client will get all the requested data no mater where they are, HQL query is executed by all mapping threads. When execution of query is finished by all threads results are combined and send back to client.

8.4.3. OPC UA client

In order to test the solution we need some data. In normal environment we would have a lot of possible data sources, like PLC's for example. In laboratory case data were generated by simulation build in program. The data can also be collected from other instances of application or independent OPC UA servers. Off course data streams are continuous and we want to store data from many sources at the same time. To accomplish this for each remote data source, application creates an instance of OPC client that takes care about gathering data and storing them through hibernate layer into database.

8.5. Testing and results

The most important part of project was testing process. At this point I have to emphasize that program I wrote is merely a prototype. It was created in order to test solution and find out if it is possible to use it in industrial systems where efficiency in processing huge amounts of data is required. Fact that used SDK has a lot of shortcomings in documentation led to situation, where only basic functionalities are implemented. In order to make it fully functional historical data server a lot of work is required.

8.5.1. Testing methodology

Tests were performed on one machine with remote database server emulated by virtual machine with Debian 6.0 guest operating system and PostgreSQL 8.4 running on it. MySQL database was running on host system. Software used for virtualization was Oracle VirtualBox 4.0.8. Application was running locally using JRE in version 1.6.

After starting server I connected to it from local client and made a request for history of one value. Data were requested 60 times: first read required retrieving data from databases. Response to next five request was made using data set created earlier. After six request database read was forced and cycle was repeated. After ten cycles of six requests application was closed and cache of databases cleared. Next, another query could be made. I tested 12 different queries, with result sets of size between 5367 and 72000 elements. Average time of reading 1000 elements from databases by the server varies between 42 ms and 114 ms. It is good result if we take into account fact, that hibernate creates one object for each row retrieved from database, what creates overhead.

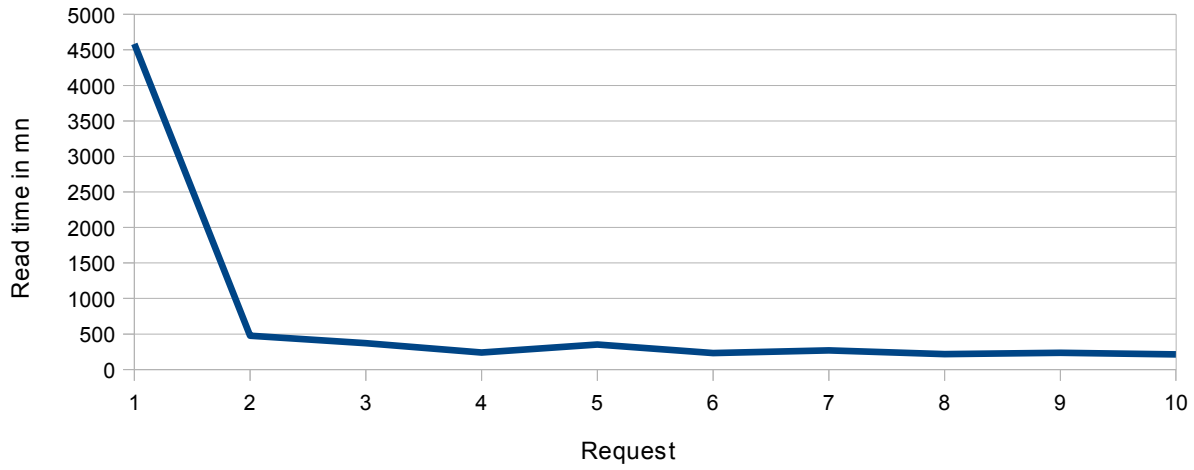

Figure 13 : Java part – Chart 1

Further analysis of results showed, that each time we read from database information, that was earlier read, time we have to wait for response from database is shorter. This effect is result of caching mechanisms implemented in DBMSs and it means, that running database on dedicated server with a lot of RAM memory may result in achieving even better performance. Exemplary dependency is shown on figure 13.

8.6. Summary and conclusions of Java part

After four months of work my main goals were achieved. Realization and successful ending of this project shows, that it is possible to create efficient and flexible solution of accessing historical data using hibernate library. Additional layer in the form of hibernate creates very small overhead but makes final product extremely flexible and adjustable. If this project will be continued additional performance could be gained by tuning database engine, running DBMS on dedicated machine in order to make use of caching mechanisms and using hibernates caching mechanisms. During test I have noticed, that hanging packet size in history read window has influence on efficiency. Probably it has connection with process of copying in `getPart` method inside Server class. In future this dependency could be analyzed. Also code can be improved in many places.

9. Development plan

9.1. Gantt diagram

Gantt diagram is added as appendix to this preliminary report. It contains plan of our work and task we planned to execute.

9.2. Week schedule

Below is presented week schedule. It includes daily project work time and every week meetings which will be taking place in Linus.

Table 4 : Week schedule

Hours	Monday	Tuesday	Wednesday	Thursday	Friday
8:30 – 9:30					
9:30 – 10:30		PROJECT	PROJECT	PROJECT	PROJECT
10:30 – 11:30	MEETING				
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					

Summarizing – the Java part has consume about 500 hours of work and Windows and C/C++/C# part consume about 460 hours of work.

Because the essence of our work was software development we were able to continue our work remotely - even when we were away from university.

9.3. Milestones and meetings

Below it is presented summary of milestones, that our project contains.

Table 5 : Milestones

Milestone	Date
Preliminary report	1.03.2011
Completion of environment for Windows part	14.03.2011
Completion of Data Provider in Java part	1.04.2011
Completion of 1st stage of modifications in Windows and C#/C++ part	6.04.2011
Completion of 2nd stage of modifications in Windows and C#/C++ part	21.04.2011
Completion of Server Side in Java Part	27.04.2011
Completion of OPC client application in Windows and C#/C++ part	3.05.2011
Completion of Analysis and measurement in Windows and C#/C++ part	13.05.2011
Final report due	25.05.2011
Oral presentation	27.05.2011
Web page done	6.06.2011

Table 6 : Meetings

Meeting no.:	Date	Place	Time
Meeting 1	Fri 25.02.2011	Linus	10:30 – 11:15
Meeting 2	Mon 14.03.2011	Linus	10:30 – 11:15
Meeting 3	Mon 4.04.2011	Linus	10:30 – 11:15
Meeting 4	Mon 2.05.2011	Linus	10:30 – 11:15
Meeting 5	Mon 19.05.2011	Linus	10:30 – 11:15

To be perfectly honest we came across difficulties that we did not expected before beginning of the project. We managed to meet two times during the semester. It was caused by very slow progress of work and because of character of our work, that was writing source code, except this code there was nothing spectacular to show. Nevertheless we managed to finish work on time with very satisfactory results.

10. Budget

Our work will be focused mostly on software solutions, so the most important will be tools, applications and development environment that will allow us to complete our goals. Some of used and pointed below software is not freeware or open source products, however as students of Silesian University of Technology, we are taking part in MSDN Academic Alliance program which give us access to free Microsoft software for academic use. Because of that we should be able to use only free of charge software for our work.

Table 7 : Budget of project

Number	Requirement	Cost
COMMON RESOURCESS		
1	Additional screen provided by HSF	0 NOK
2	Desktop computer provided by HSF	0 NOK
3	Additional network cables and router provided by HSF	0 NOK
4	Microsoft Project Professional 2010	0 NOK
WINDOWS AND C# / C++ PART		
6	Microsoft Visual Studio 2008/2010 Professional	0 NOK
7	Microsoft SQL Server 2008 Express Edition	0 NOK
8	OPC HDA Sample code	0 NOK

JAVA AND LINUX PART		
9	Eclipce	0 NOK
10	Prosys OPC SDK (60 days evaluation)	0 NOK
11	MySQL / PostgeSQL	0 NOK
Total cost :		**0 NOK**

11. Organization

Presentation of project structure is placed below.

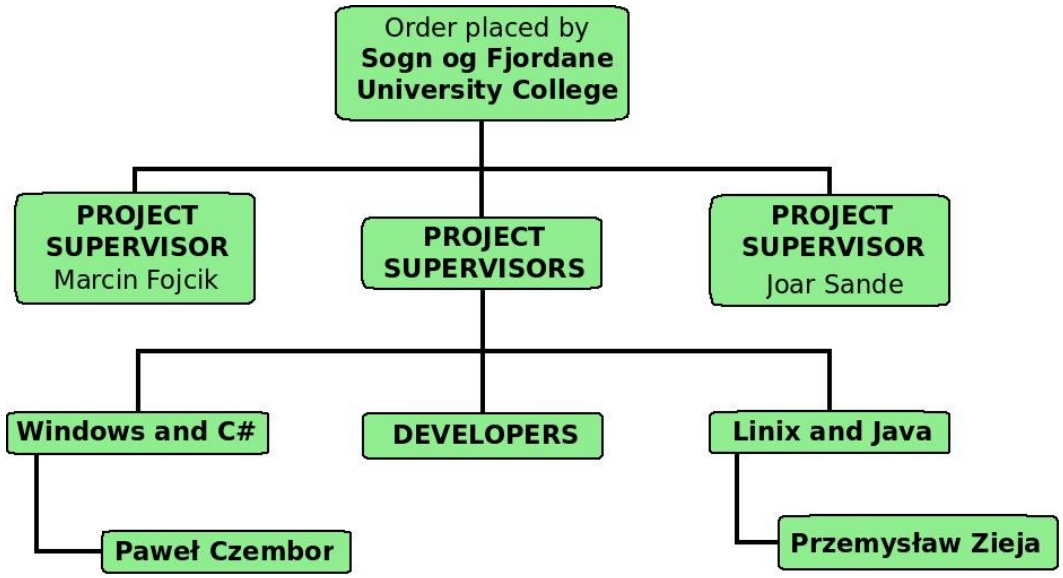

Figure 13 : Project organization structure

11.1. Ordering company

Project has been ordered by Sogn og Fjordane Univeristy College that is represented by Marcin Fojcik and Joar Sande. The finalized Senior Design Project will be presented as part of partnership program between HSF and Silesian University of Technology in Gliwice.

11.2. Supervisors

Project supervisors 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. Technical support supervisor is Marcin Fojcik and management and organisation supervisor is Joar Sande.

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

11.3. Developers

The project developers are: Paweł Czembor and Przemysław Zieja. They are authors of this document as well . Both Paweł and Przemysław are students from Poland who has came to HSF as participants of STF international student exchange program.

Because of way that project has been divided there was no need to specify who was the project leader. Clearly shared tasks made Paweł responsible for whole Windows and C#/C++ development part while Przemysław's responsibility was Java and Linux development part.

Name:	E-mail address:
Paweł Czembor	pawelc@stud.hisf.no
Przemysław Zieja	przemysz@stud.hisf.no

12. Reading list

1. OPC Foundation
http://www.opcfoundation.org
2. Microsoft Developer Network
http://msdn.microsoft.com/pl-pl/library/kx37x362(en-us).aspx
3. Wikipedia (definition for the abbreviations)
http://en.wikipedia.org/wiki
4. "OPC Unified Architecture";
W. Mahnke, S.H. Leitner, M. Damm
5. "Beginning Hibernate";
Dave Minter and Jeff Linwood
6. Java 6 platform documentation
http://download.oracle.com/javase/6/docs/
7. "Thinking in C++";
Bruce Eckel
8. Hibernate community documentation
http://docs.jboss.org/hibernate/core/3.6/quickstart/en-US/html/
9. JDOM v1.1.1 API Specification
http://www.jdom.org/docs/apidocs/index.html
10. PostgreSQL 8.4 documentation
http://www.postgresql.org/docs/
11. Senior Design Project -titled :
"Data acquisition system with database using OPC"
http://prosjekt.hisf.no/~10opcdp/
Also available in HSF library

13. List of tables

14. List of figures

15. List of appendixes

1. Gantt diagram for Senior Design Project titled :

   "Efficient data access in OPC UA based systems"

2. Preliminary Report for Senior Design Project titled :

   "Efficient data access in OPC UA based systems"

15.1. Additional Java libraries

Hibernate 3.6.1
http://sourceforge.net/projects/hibernate/files/hibernate3/3.6.1.Final/

Jcalendar 1.3.3

http://www.toedter.com/en/jcalendar/index.html

JDOM 1.1.1

http://www.jdom.org/

Prosys OPC UA Java SDK Client-Server Evaluation 1.1.2-1941

http://www.prosysopc.com/opc-downloads.php

Important!

Because of license restrictions I am not allowed to attach Prosys SDK to project. In order to obtain this libraries ant run application please contact with developer of this SDK.