EasyIT

KPRO10
Institute of computer science, NTNU

Hans Olaf Borch
Thorvald Johannessen
Truls Jørgensen
Jan Ove Skogheim Olsen
Kristoffer Stenersen
Harald Søvik
Øystein Ulseth

kpro10@idi.ntnu.no

19th November 2004

Contents

List of Figures

List of Tables

Bibliography

[1] M. S. AB. http://www.mysql.com.

[2] G. D. Access and C. Framework. http://sourceforge.net/projects/gdac/.

[3] G. D. Access and C. Framework. http://sourceforge.net/projects/opcxml/.

[4] B. Anda. http://www.idi.ntnu.no/emner/tdt4290/docs/faglig/uc-projectestimatemethodv2-test.xls.

[5] B. Anda and R. Conradi. http://www.idi.ntnu.no/emner/tdt4290/docs/faglig/usecase-estimering.doc.

[6] ComponentArt. http://www.componentart.com/.

[7] M. Corporation. http://www.microsoft.com.

[8] O. Corporation. http://www.oracle.com.

[9] W. O. C. Dictionary. http://www.webopedia.com/.

[10] T. O. Foundation. http://www.opcfoundation.org/.

[11] D. from Agile Integration. http://www.agile.co.za/software/daqbench.htm.

[12] O. S. from Eldridge Engineering Inc. http://www.opcsystems.com/.

[13] M. from MathWorks. http://www.mathworks.com/.

[14] P. A. C. E. from OsiSoft. http://www.osisoft.com/.

[15] IEEE. Recommended practice for software design descriptions.

[16] IEEE. Recommended practice for software requirements specifications.

[17] IEEE. Standard glossary of software engineering terminology.

[18] N. Instruments. http://www.ni.com/.

[19] C. Labs. http://www.canarylabs.com/.

[20] Matrikon. http://www.matrikon.com/.

[21] I. Objects. http://www.integ-objects.com/.

[22] C. One). www.componentone.com.

[23] Skype. http://www.skype.com.

[24] I. Systems. http://www.idi.ntnu.no/emner/tdt4175.

[25] V. V. (vetle@stud.ntnu.no) and K. S. (skogstro@stud.ntnu.no). http://www.idi.ntnu.no/emner/tdt4145/programvare/ermod/ermod.jnlp.

Part I

Project Directive

Chapter 1
Prelude

The organization of the project takes place in the project directive. The purpose of this directive is to regulate the administrative part and the accomplishment of the project. The guidance is however expected to change throughout the lifetime of the project. The project directive will also change when these changes take place.

Chapter 2
Project mandate

2.1 The name of the project

The name of the project is EasyIT.

2.2 Customer

The customer is:
ABB Corporate Research Center.
Bergerveien 12
1375 Billingstad
Norway

Customer representative:
Name: Karl-Petter Lindegaard
Phone: 66843395
Fax: 66843060
Email: karl-petter.lindegaard@no.abb.com

2.3 Interested parties

Interested parties are the customer (see 2.2), the EasyIT project group and the groups advisors. Please see appendix A for a detailed list of the interested parties.

2.4 Background

A couple of years ago, the easy access to data the industry now witnesses wasn't feasible. The reason for this change is the rapidly changing world of information technology. The industry now wants tools for processing this data.
Examples of tools are:

1. Tools to quantitatively identify and characterize the business. These tools aim to optimize the production, and thereby increase the result.
2. Tools to identify damaged or partly damaged equipment. These tools aim to reduce the down time of the production as a whole, and with that increase the production and result.
3. Tools to generate reports for documenting the business (for instance, environmental requirements stated by the authorities)

These tools could be used for automation of processes (generation of predefined applications and reports). However, qualified personnel could also use these kinds of tools for looking back in history and searching for new ways to view the processes.
A big challenge is to present an alternative that is

- easy to configure
- easy to expand with new applications for analysis of data
- user friendly when it comes to accessing data
- low cost

Earlier, the market for these systems was reserved for the big vendors and their proprietary communication protocols. Now, several smaller vendors have made their way into the market, because they use new and open standards for accessing the system data at a much lower cost compared to the bigger vendors (i.e. ABB).

2.5 Effect goals - high level requirements

By effect goals it is meant the effect that is to be achieved in the project. The main goal of this project is to make a system that helps ABB to become even more competitive as a vendor for the industry.
The system should be

- easy to configure
- user friendly when it comes to accessing new data, and exploring historical data
- easy to expand with new applications for analysis of data
- low cost - use open standards for communication with the underlying system.

2.6 Result goals - deliveries

By result goals it is meant those deliveries that the client demands to make the project a success. These are:

1. an infrastructure[1] for collection and suitable storage for large volumes of measure- and process data
2. a framework[2] for development of small, scheduled applications that operate on already collected data, thereafter writing back the result to the infrastructure in pt 1.
3. a GUI gateway that presents both raw values and application results, logged by pt 1. The client wants a web application for this purpose.

Of these three points, the most significant is the first one. Without this one, there won't be foundation for the rest.

2.7 The scope

The different stages of the project are listed in the project plan (please see chapter 3). These stages will be put together to form the report. The members of the group must learn what is needed to make the project a success.

This includes

- Documentation tools
- Programming platform

[1] A set of ready-to-use applications
[2] a skeleton to build applications upon

- Version control systems

The end product - the report, source code and documentation will be delivered to the client. The following documents will be included in the final report:

- Project directive
- Pre study
- Requirements specification
- Software Design Description
- Test Document
- Implementation Description
- Evaluation

2.8 External conditions

This subsection describes the external conditions. This means elements that can constrain the project team's work:

- The client is physically located in Oslo, and is limited to very few face-to-face meetings, perhaps only three:

    1. At startup
    2. In the middle
    3. At the final presentation

    We are planning to use various forms of net meeting programs as a substitute.

- Microsoft products are used for most development tasks in ABB. In order to incorporate our solution, the client wants us to use Microsoft products in the development.
- A computer where necessary software can be installed. Necessary software includes an OPC server and perhaps a SourceSafe server. This computer will be provided by Department of Computer and Information Science (IDI).

2.9 Resources

The project team consists of 7 persons who are expected to work 310 hours each during the project. That gives 2170 hours totally for the entire team.

2.10 Schedule

This part presents the project's duration and important dates.

2.10.1 Duration

The project starts 2004.08.24 and ends 2004.11.18

2.10.2 Important dates

2004.08.24 Start of the project

2004.08.27 Group dynamics part 1, "Six thinking hats"'

2004.09.06 Group dynamics part 2, "Team building" , Luftkrigsskolen

2004.10.28 Pre delivery of the Preliminary Study and the Requirements Specification delivered to examiner.

2004.11.08 Order time for copying the final report.

2004.11.15-17 Copying of the final report

2004.11.16-18 Presentation rooms available for rehearsal

2004.11.18 Presentation and demonstration

Chapter 3
Project plan

3.1 Phases

We have chosen to divide the project into the following phases:

- Preliminary planning (project directive) - see 3.1.1
- Pre study - see 3.1.2
- Requirement specification and create test plan - see 3.1.3
- Design and test plan construction - see 3.1.4
- Implementation and testing - see 3.1.5
- Project evaluation - see 3.1.6
- Presentation and demonstration - see 3.1.7

There are also two continuous tasks spanning the project lifetime, which are always executed in parallel to one or more phases:

- Project management - see 3.1.8
- Lectures and self study - see 3.1.9

The use of the term "responsible" in the sections below are used in the sense that the group member listed is intended to perform the activity described.

3.1.1 Planning

In this phase the first draft of these documents was sketched and the group was organized.

Activity	**Responsible**
Organizing the group	Everyone
Responsible for the project directive	Harald
Starting up the pre study	Jan Ove
Templates and standards	Truls and Øystein
Time	Kristoffer
Creating activity plan	Kristoffer and Truls
Plan quality assurance	Thorvald

3.1.2 Pre study

In this phase we'll begin the study of our problem. A description of today's solutions will be presented, as well as tomorrow's potential solutions. The operational and business requirements are laid down, and form the foundation for alternative solutions. Criteria for these solutions will be set and considered before we reach a conclusion upon our selected solution.

Given the amount of material in the technological pre study, we'll split the workload of this part of the pre study in three parts of the project. The parts are made up by looking at the result goals: (Please see section 2.6.)

- The data collector side of the project.
- The application side of the project
- The presentation side of the project

Activity	Responsible
Describe today's situation and solution	Jan Ove, Hans Olaf, Øystein
Describe desired situation and solution	Jan Ove, Hans Olaf, Øystein
Outline business demands	Harald
Outline evaluation criteria	Truls
Market research	Thorvald
Describe alternative solutions	Kristoffer
Evaluate solutions	Harald
Choice of solution	Jan Ove
Finishing of the pre study report	Harald

3.1.3 Requirement specification

The project's requirements will here be specified, thoroughly examined, systemized and prioritized. Our customer and our pre study give the basis for this phase. An outline of the software architecture will be made, as well as data models, perhaps some prototypes of screen shots and choice of tools for development.

Activity	Responsible
Responsible for the requirements document	Thorvald
Write requirements introduction	Kristoffer
Write overall description	Hans Olaf
Write non-functional requirements	Harald
Write functional requirements	Jan Ove
Document requirements	Truls
Rules for acknowledgement testing	Thorvald

3.1.4 Software Design Description and test plan construction

In this phase we prepare the realizable solution, and we'll estimate the amount of time needed for implementation. All technological issues are now to be fixed in more detail. We should now be able to know that the implementation is possible and feasible.

Activity	Responsible
Document introduction	Kristoffer
DI responsible	Jan Ove
APP responsible	Øystein
GUI responsible	Hans Olaf
DFD responsible	Harald
Class diagrams	Kristoffer
UML responsible	Truls
Estimation responsible	Truls
Sequence diagrams	Thorvald
Document composition / responsible	Jan Ove
Detailed test plans	Thorvald

3.1.5 Implementation and testing

Here we begin to implement our solution, and produce documents for programming standards, commenting, special algorithms etc. The test documentation is also produced in this phase.

Activity	Responsible
DI responsible	Jan Ove
Document introduction	Harald
APP responsible	Øystein
GUI responsible	Hans Olaf
Produce test data	Thorvald
Execution of tests	Thorvald
Documenting code	Everybody
Implementation document	Kristoffer

3.1.6 Project evaluation

Here we evaluate the process and the result, our customer and our problem, the subject and discuss further work on the project.

Activity	Responsible
Write evaluation of process and result	Harald
Evaluate customer and task in general	Hans Olaf
Evaluate course Kundestyrt Project	Truls
Further work	Øystein

3.1.7 Presentation and demonstration

Here we prepare for presentation and demonstration of our project.

Activity	Responsible
Prepare demonstration	Øystein
Prepare presentation	Kristoffer

3.1.8 Project management

This is the administrative part of the project. Meetings are the main part of this task. All of the group members are allocated to specific roles - tasks they are responsible for. See section 4.2 for details.

3.1.9 Lectures and self study

The staff of TDT4290 Kundestyrt Project arranges series of lectures relevant to the project, but also courses in team building and group dynamics. Hours spent on these activities will be registered here.

3.2 Milestones

The information in this section can also be found in context in appendix B.1 - the Gantt chart.

Milestone	**Date**
Finished project directive (planning phase)	Sep 06. 2004
Finished pre studies	Sep 17. 2004
Send requirements specification for approval to client	Sep 24. 2004
Finished requirements specification	Sep 27. 2004
Finished design of overall design	Oct 05. 2004
Finished design documents	Oct 15. 2004
Pre deliverance of pre study and requirements specification	Oct 28. 2004
Finished implementation and test documentation	Nov 04. 2004
Finished project evaluation	Nov 11. 2004
Presentation	Nov 18. 2004

3.3 Effort in each phase

The information in this section can also be read in context in appendix B.1 - the Gantt chart.

Phase	**Hours estimated**	**Hours spent**
Project management	217	
Lectures and self study	217	
Planning	245	
Pre study	301	
Requirement specification	264	
Design	340	
Programming and documentation	340	
Project evaluation	112	
Presentation and demonstration	130	
Sum	*2170*	

CHAPTER 4 ORGANIZATION

4.1 ORGANIZATION CHART

The organization of this project consists of

- the project group
- customer contact
- advisor group

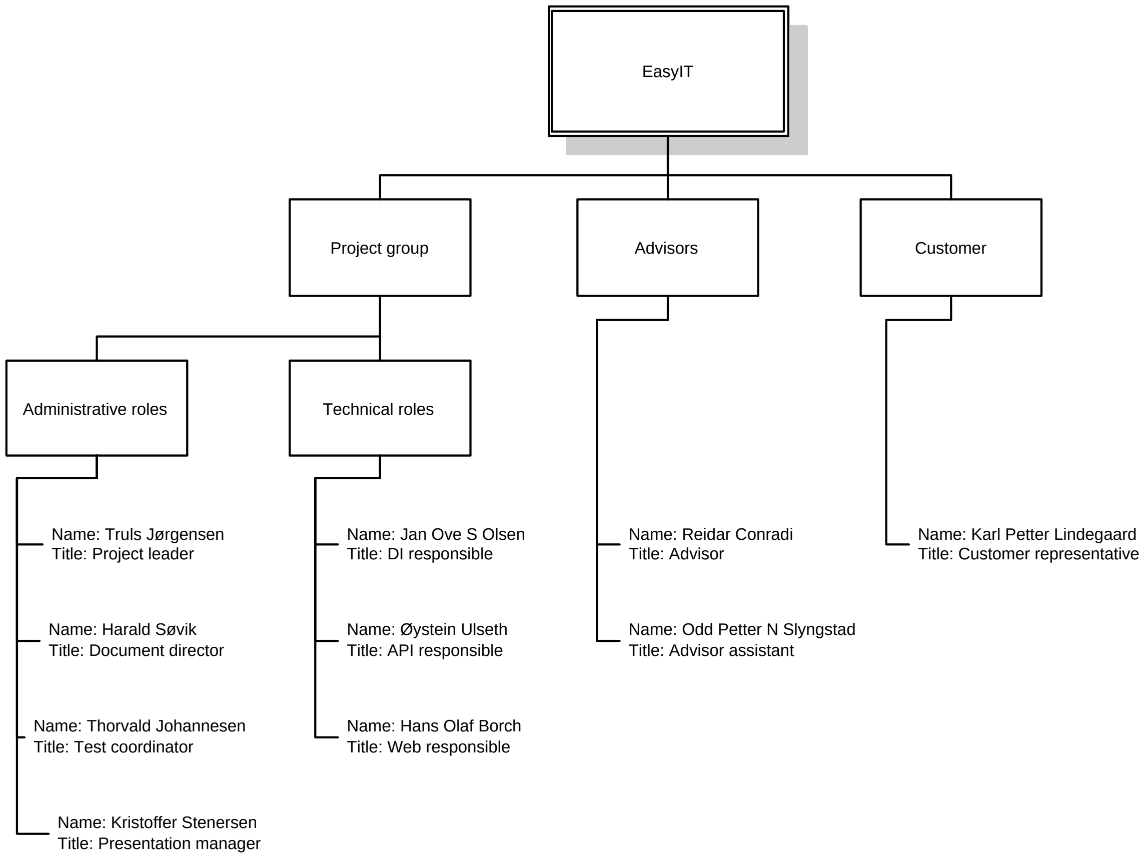


Figure 4.1: Organization structure after 2 weeks

4.2 Formal roles

These roles are assigned to last throughout the project, and are different from the roles in 3.1 in the sense that they designate responsibility, but not necessarily the actual work.

4.2.1 Project leader

His function is to reduce the overhead of administrative tasks like distributing work, follow-up on workers, coordination of tasks. In case of disagreement between members of the group, low morale or poor work completion, he has the responsibility of putting matters straight.

The project leader is Truls.

4.2.2 Document director

His function is to reduce the overhead of administering documents and source, to arrange structures and formats for storage, making sure the repository is in working order and completing documents when their content are available. When reports and drafts should be distributed, it is his responsibility that they will be so on time.

The document director is Harald.

4.2.3 Test coordinator

The test coordinator shall supervise the testing of modules and the product, making sure everything is in order to satisfy the functional requirements.

The test coordinator is Thorvald.

4.2.4 Presentation manager

The presentation manager is there to coordinate every part of the demonstration of the product. Making sure slides are being made, demonstration equipment is in working order.

The presentation manager is Kristoffer.

4.2.5 Customer contact

The customer contact will coordinate all communication with the customer, to reduce his overhead, and avoid duplicate questions or redundant information flow.

The customer contact is Hans Olaf.

4.2.6 Subsystem director 1: DI

The director of the Data Interchanger module is supervising the construction of the module, reporting to the project leader if there is to much or too little labour available. He is also working together with the test coordinator to make sure his module is compatible with other modules and is satisfying the functional requirements.

The subsystem director 1 is Jan Ove.

4.2.7 Subsystem director 2: APP

The director of the application module is supervising the construction of the module, reporting to the project leader if there are too much or too little labour available. He is also working together with the test coordinator to make sure his module is compatible with other modules and is satisfying the functional requirements.

The subsystem director 2 is Øystein.

4.2.8 Subsystem director 3: GUI

The director of the GUI module is supervising the construction of the module, reporting to the project leader if there is too much or too little labour available. He is also working together with the test coordinator to make sure his module is compatible with other modules and is satisfying the functional requirements.

The subsystem director 3 is Hans Olaf.

CHAPTER 5 TEMPLATES AND STANDARDS

During the project, we will produce a number of documents. We have developed templates that will be applied to these documents because we want our documentation to be as structured as possible. The group has also agreed on using standards regarding naming of directories and files. This chapter defines the templates and standards and corresponds with chapter 8 - Quality Assurance regarding the time to respond for each document type.

5.1 TEMPLATES

5.1.1 Phase documents

This part, the project directive, forms the standard for the appearance of all the phase documents. We have planned the use of a internal log that contains updates in these phase documents.

5.1.2 Notification of meetings

We have two different types, for two different purposes:

- Notifying the customer of a meeting.
  This template is attached in D.2
- Notifying the advisor of a meeting
  This template is attached in D.1

5.1.3 Status report

This template will be applied to all status reports. The status report is a weekly summary document written by the project coordinator and delivered to the advisor before the weekly advisor meeting. This template is attached in D.5

5.1.4 Minutes of meetings

This template is meant to be applied to all minutes written after meetings with the supervisor and the customer. However, minutes of internal meetings do not need to be have this template applied - it will depend on whether we consider the meeting important or not. This templates are attached in D.3 and D.4.

5.2 Standards

The group is using CVS for file sharing and version control on documents. The main directories are named by the phases of the project: Project directive, preliminary study and so on. Each file is a chapter, and chapters are named like this: `Chapter_X_NAME.tex`. However, we are probably going to use another tool for file sharing when it comes to programming code; Source Safe. The reason for this is that we probably will develop in Visual Studio .Net, and Source Safe provides version control for .Net code.

Chapter 6
Documentation and versioning

This project will be using both CVS [1]and SourceSafe [2]for versioning files.

As this project is both educational and professional, there is loads of documentation that will be produced by a number of people. It is therefore a necessity to be able to gather documents and keep them consistent while under development. Because of the considerable volume of text, LaTeX will be used for markup, which also allows us to use CVS for versioning due to its plaintext-nature.

6.1 Textual documents

6.1.1 LaTeX - what and why

LaTeX is a compiler that translates your written document code into your favorite document format. You can write in any editor capable of saving plain text, and is therefore very platform-flexible. Your code consists of HTML-like tags, but (unlike HTML) LaTeX is very strict regarding correct formatting. While you are writing, you are supposed to write, and not try to make your document look nice. Layout can be managed in a separate part of the document, possibly even in a separate file (which in turn can be common for all of your documents).

LaTeX has a lot of labelling and numbering functions built-in, and i.e. headers and references are managed automatically. Content can be separated into different files, and later included in the complete document. This eases the overhead of cooperating on the same report.

6.1.2 CVS - what and why

Concurrent Version System called CVS controls your files by storing them in a “library” (repository). Your CVS-client will do a “checkout” of your project, making a local copy. This copy can be edited without interference from other editors whenever you like. When you are done editing your document and you have had your document checked for correctness, consistency and completeness, you ask the CVS client to update the repository.

This is where CVS has its greatest strength: If someone else has changed the repository while you were away editing, CVS can either merge the differences or let you resolve conflicting sections. (CVS also provides methods for setting locks to files, but since setting locks to plain text documents will block the merging capability of CVS, we’ll not be using this feature.)

In addition there are functions for adding and removing files in the central storage without haveing to bother with FTP or overwriting changes you were unaware of. You are also able to lock files if you feel like they should stay the way they are, or ask CVS to notify you when someone changes a file.

[1]CVS is a version control system. Using it, you can record the history of sources files and documents. http://www.gnu.org/software/cvs/

[2] Visual SourceSafe provides individual developers and small development teams with tools to make safe alterations to existing code and track changes across users, projects, and time. http://msdn.microsoft.com/vstudio/previous/ssafe/

That is the concurrent part. The versioning part is rather simple. Every time someone commits a change to a file, the change is saved to the file, making you able to "roll back" to an earlier version of your choice. This works well with LaTeX, since both text and code is in plaintext.

6.1.3 Alternatives to CVS

- Bitkeeper - expensive
- Subversion - not available at student servers
- SCCS - only available under UNIX source code license

Anyway, CVS covers of all our needs, is available for use with hardly any configuration and is installed on student servers. CVS is available under GNU General Public License[3].

6.2 HANDLING OF PROGRAM CODE

6.2.1 Microsoft Visual SourceSafe

The actual program code development will, because of customer request, be done in the programming language C#. This leads us to use Microsoft Visual Studio as development environment. This in turn will enable us to use integrated solutions for versioning, source control and code documentation. SourceSafe integrates in the development environment, and thereby reduces code management overhead, in addition to offering functions to reduce actual development time, i.e. reuse of code.

6.2.2 Alternatives to SourceSafe

What's described in the previous section is very similar to CVS and other versioning systems. However, MVSS is the only Microsoft-native system. Using Visual Studio with a "alien" versioning system would provide another layer with possibility of errors. There are versioning systems which integrate with the Explorer interface of Windows, but this adds file management overhead. No other system offers the system of dynamic update of reused code. Our only concern with economics in this project is hours spent. In addition, both Visual Studio and SourceSafe is available "for free" through Microsoft Development Network Academic Alliance[4]. This is therefore the cheapest solution.

[3]http://www.gnu.org/copyleft/gpl.html
[4]http://www.msdnaa.net

CHAPTER 7

PROJECT MONITORING

7.1 MEETINGS

Our meetings can be split up into three categories:

7.1.1 Internal meetings

We hold our internal meetings every Monday at 10 am. The agenda is to review the last week's activities, check our status compared with our plan, discuss and coordinate further activities. There is an informal ambience, but is led by the project leader. The report of this meeting is the basis of the status report for the main meeting at Wednesdays.

7.1.2 Advisor meetings

Our meetings with our advisor and the advisor assistant are held Wednesdays 10 am to 11 am in room ITV-242. These meetings are of a more formal matter, and our project leader lead them according to the meeting agenda. The group can receive comments on their productivity and work, get help to approach eventual problems and get general guidance. Project status and progress are reported to the project advisor and the advisor assistant who again give feedback on these matters.

7.1.3 Customer meetings

These meetings are seldom person to person, because of our customer's distant location. We will mainly be in touch with him by email, but also by net meetings / video conference etc. The group's customer contact will be responsible for these meetings.

7.2 INTERNAL INFORMATIVE FUNCTION

7.2.1 Time keeping

Every group member keeps their hour list updated in our shared group folder. These lists are Excel sheets, and there is also one sheet for giving a total of all the reported hours, and compare these numbers to our schedule. Group members have to fill in the last week's hours by the internal meetings at Mondays. A printout will be brought to the main meetings.

7.2.2 Activities and milestones

Our activities and milestones will mainly be discussed Mondays, in our internal meetings. Greater variances from the decided schedule will be clarified and dealt with.

7.3 Status reports

Status reports will be generated as a result of the internal meetings, and delivered to our project advisor before 12 am Tuesdays (the day before the main meetings), along with the other documents he'll receive. See 8.7.3 for the list of documents. Our advisor assistant will receive an email copy of these documents before 12 am. See 5.1.3 for a description of the status document.

7.4 Risk management - TRECQ

TRECQ is an evaluation of the project's status in view of

- Time. Progress and milestones compared to our timetable?
- Risk. Elements of risk, consequences and response? See C.1 for the risk table.
- Extent. Is our task growing or decreasing?
- Cost and time keeping. Are we following the budget?
- Quality. Do we have to reduce the quality of our product?

These points are addressed at project meetings, but they are also commented in the status report.

Chapter 8
Quality Assurance

This document describes quality guidelines for all routine tasks within the project. By setting these standards, we hope to both gain efficiency and secure the quality in our work. We have established guidelines for communication, response times, creation and approval of all documents and tasks considering all meetings.

8.1 Quality assurance and measurement in EasyIT

The system we have been assigned to design, will take the form of a prototype, and by this serve the purpose of a pre-study for later development within the ABB research group. Our system will never be used directly by an end user in a realistic environment. This implies that an error in the performance of our system will not have major consequences for either the user or the rest of the control-system, and will of course affect our quality assurance guidelines. This does not mean that we have a lower demand for quality assurance, but it will affect which areas we choose to emphasize in the design, implementation and test phases. Our specific product will affect the standards by which we, and the customer measure quality.

8.2 Customer related quality criteria

In collaboration with our customer, we have identified goals that forms a base of high-level objectives for the project. These are classified as quality criteria factors. Some of the criteria are directly and objectively measurable, while others are subjective and indirectly measurable. Regardless of measurability, the customer will evaluate the success factor in the project by these criteria.

Usability

- It should take a minimum of time to develop a new application that fits into the derived framework. This includes easy access to collected data, scheduling of applications, and web access for non-developers / users.
- The application development process must be easy and should not require extensive knowledge of the underlying system.
- Reconfiguring the data collector should be easy.
- The web portal should have a uniform layout, a recognizable theme displaying web pages in a similar way.
- All text presented should be in a common language for all users.
- The final product should be easy to install. "Wizard" installation is preferred.

Reliability / Efficiency

- The system should be able to perform without difficulties, while treating 500 tags on a 1 second rate, 500 on a 10 second rate and 500 on a one minute rate.
- The system should be able to provide historical data logging up to 6 months.
- The web portal shall be able to serve several web browsers accessing the data in parallel.

Portability

- It should be easy to change the layer communicating with the underlying data sources.

Documentation

- Documentation should reflect the choices made in implementation. The possible implementation solutions available and those finally chosen should be documented well, describing the reason for our choices. Use of standard notation and diagrams are preferred. Source code should be included in the final deliverance.

8.3 Communication

Besides weekly internal meetings, the group will communicate through email (kpro10@idi.ntnu.no) and via the discussion page at the group's web-portal (http://www.stud.ntnu.no/groups/kpro10). The web-portal will serve the purpose of longer discussions not suitable for email. At the web portal the group members also share a calendar, in which each member of the group inserts marks when not available for work. Also included in the calendar are important milestones of the project, lectures etc.

8.4 Response times

8.4.1 With customer

- Approval or comments to the report from customer meetings is sent back to the author within 24hrs.
- Approval or comments on phase documents is sent back to the author within 48hrs.
- Answers to simple questions within 24hrs.
- When bigger questions occur, customer will respond within 24hrs that the question is received. Customer will then give an estimate on how long it will take to gather the requested information.

8.4.2 Internal

Approval or comments on phase documents is received during the next period of weekly internal group meeting.

8.5 Routines for producing quality the first time

There will be no groups smaller than two persons for all tasks involving production of a final document. This document can be either code or a phase document. By this we aim to minimize the effect of errors made by one person.

8.6 Routines for approval of phase documents

All phase documents are stored on the CVS server, since all these documents are considered dynamic until delivery. The author will notify the rest of the group when a new document is ready for approval. Group members will give their comments through the already defined communication channels.

8.7 Reports and meeting notices

8.7.1 Customer

- Notice to customer will be sent 48hrs before the meeting, if not the notice will include the agenda and other relevant documents. All customer meetings will be scheduled at least one week in advance.
- Reports from customer meetings will be ready at 12:00 the day after the meeting. The author will distribute the report internally by email. When the report has the group's approval, email containing the report will be sent to customer. The customer will receive the report within 48 hours after the meeting.

8.7.2 Internal

- Notice for internal meetings is available at the calendar on the kpro10 web portal.
- Reports from internal meetings will be placed at the groups file server, where those not able to attend can read later.

8.7.3 Advisor

Notice of meeting with advisor will be delivered by hand by the project manager at the project guide's office by 12:00 the day before the meeting. The content of the delivery will be:

- Notice of advisor meeting
- Summary of last meeting
- Status report and all other documents of relevance.
- Phase documents to be considered (i.e. Project Directive, Pre study, Requirements Specification and so on.

The attachments of the notice are stated in the notice body. See 5.1.2 for the templates of these documents.

Appendix A
Interested parties

The appendix includes contact information of interested parties

A.1 Client representative

Karl-Petter Lindegaard Researcher, PhD
Corporate Research Center, ABB
Tlf: 66843395
Email: karl-petter.lindegaard@no.abb.com

A.2 Advisors

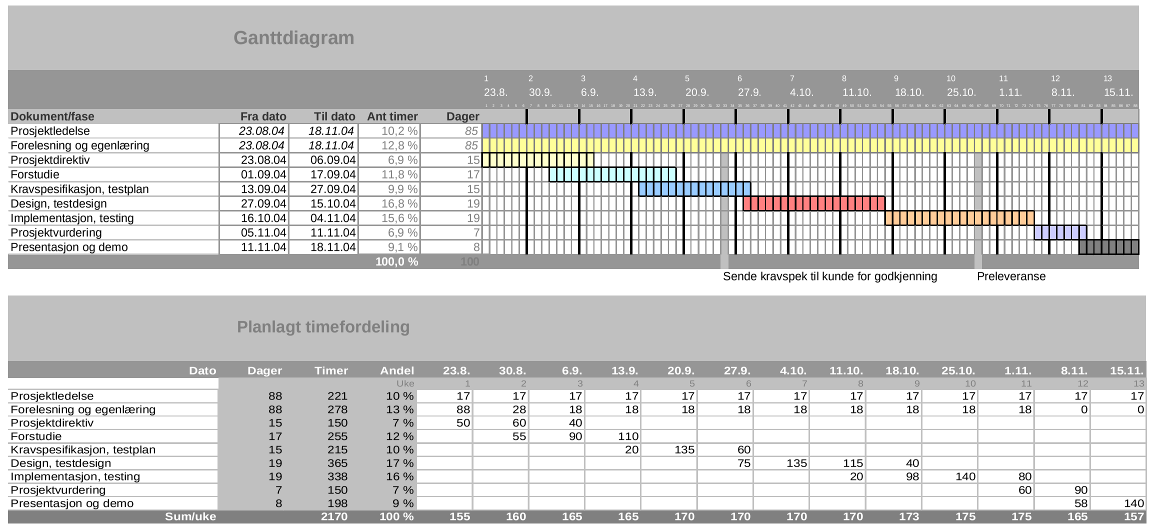
Reidar Conradi
Professor
Department of Computer and Information Science, NTNU
Email: conradi@idi.ntnu.no

Odd Petter N. Slyngstad
Advisor assistant
Department of Computer and Information Science, NTNU
Email: oslyngst@idi.ntnu.no

A.3 Project Team

Jan Ove Skogheim Olsen
Email: janovesk@stud.ntnu.no
Hans Olaf Borch
Email: borch@stud.ntnu.no
Harald Søvik
Email: harals@stud.ntnu.no
Truls Jørgensen
Email: trulsjor@stud.ntnu.no
Thorvald Johannessen
Email: thorvaj@stud.ntnu.no
Øystein Ulseth
Email: oysteiul@stud.ntnu.no
Kristoffer Stenersen
Email: kristost@stud.ntnu.no

APPENDIX B
GANTT CHART AND STATUS DIAGRAMS


Ganttdiagram

Dokument/fase	Fra dato	Til dato	Ant timer	Dager
Prosjektledelse	*23.08.04*	*18.11.04*	10,2 %	*85*
Forelesning og egenlæring	*23.08.04*	*18.11.04*	12,8 %	*85*
Prosjektdirektiv	23.08.04	06.09.04	6,9 %	15
Forstudie	01.09.04	17.09.04	11,8 %	17
Kravspesifikasjon, testplan	13.09.04	27.09.04	9,9 %	15
Design, testdesign	27.09.04	15.10.04	16,8 %	19
Implementasjon, testing	16.10.04	04.11.04	15,6 %	19
Prosjektvurdering	05.11.04	11.11.04	6,9 %	7
Presentasjon og demo	11.11.04	18.11.04	9,1 %	8
			100,0 %	100

Planlagt timefordeling

Dato	Dager	Timer	Andel	23.8.	30.8.	6.9.	13.9.	20.9.	27.9.	4.10.	11.10.	18.10.	25.10.	1.11.	8.11.	15.11.
			Uke	1	2	3	4	5	6	7	8	9	10	11	12	13
Prosjektledelse	88	221	10 %	17	17	17	17	17	17	17	17	17	17	17	17	17
Forelesning og egenlæring	88	278	13 %	88	28	18	18	18	18	18	18	18	18	18	0	0
Prosjektdirektiv	15	150	7 %	50	60	40										
Forstudie	17	255	12 %		55	90	110									
Kravspesifikasjon, testplan	15	215	10 %				20	135	60							
Design, testdesign	19	365	17 %						75	135	115	40				
Implementasjon, testing	19	338	16 %								20	98	140	80		
Prosjektvurdering	7	150	7 %											60	90	
Presentasjon og demo	8	198	9 %												58	140
Sum/uke		2170	100 %	155	160	165	165	170	170	170	170	173	175	175	165	157

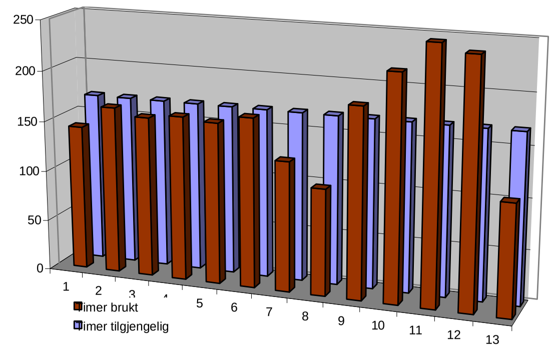
Figure B.1: Our Gantt chart (top) and planned distribution of hours for each phase

Statusoversikt

Tilgjengelig antall timer	2170
Timer tilgjengelig / uke	167

Rød tekst er kopi fra sheet "Gant, planlagt".

Dokument/fase	ıtatt fordeling	Antall timer	Brukt hittil	Gjenstår
Prosjektledelse	10,2 %	221	230	-9
Forelesning og egenlæring	12,8 %	278	237	41
Prosjektdirektiv	6,9 %	150	160	-10
Forstudie	11,8 %	255	267	-12
Kravspesifikasjon, testplan	9,9 %	215	286	-71
Design, testdesign	16,8 %	365	302	63
Implementasjon, testing	15,6 %	338	349	-11
Prosjektvurdering	6,9 %	150	256	-106
Presentasjon og demo	9,1 %	198	110	88
Sum	100,0 %	2170	2197	-27



Totalsum for alle på gruppa, ukevis

Dato	23.aug.04	30.aug.04	06.sep.04	13.sep.04	20.sep.04	27.sep.04	04.okt.04	11.okt.04	18.okt.04	25.okt.04	01.nov.04	08.nov.04	15.nov.04
Uke	1	2	3	4	5	6	7	8	9	10	11	12	13
Prosjektledelse	39	26	26	37	21	17	12	14	9	15	14	0	0
Forelesning og egenlæring	74	38	51	23	18	16	7	8	2	0	0	0	0
Prosjektdirektiv	25	72	23	13	8	5	4	0	0	8	2	0	0
Forstudie	5	29	58	86	51	19	6	2	3	8	0	0	0
Kravspesifikasjon, testplan	0	0	0	3	59	107	51	15	9	37	5	0	0
Design, testdesign	0	0	0	0	2	3	48	55	116	57	21	0	0
Implementasjon, testing	0	0	0	0	0	0	0	11	48	96	158	36	0
Prosjektvurdering	0	0	0	0	0	0	0	0	0	0	50	206	0
Presentasjon og demo	0	0	0	0	0	0	0	0	0	0	0	0	110
Sum/uke	143	165	158	162	159	167	128	105	187	221	250	242	110

Akkumulert timer brukt så langt	143	308	466	628	787	954	1082	1187	1374	1595	1845	2087	2197
Planlagt timer brukt så langt, akk	155	315	480	645	815	985	1155	1325	1498	1673	1848	2013	2170
Avvik	*-12*	*-7*	*-14*	*-17*	*-28*	*-31*	*-73*	*-138*	*-124*	*-78*	*-3*	*74*	*27*

Figure B.2: The generated status diagram - generated 19th November 2004

APPENDIX C RISK DIAGRAM

This section includes the risk diagram, and complements the TRECQ section of 7.4

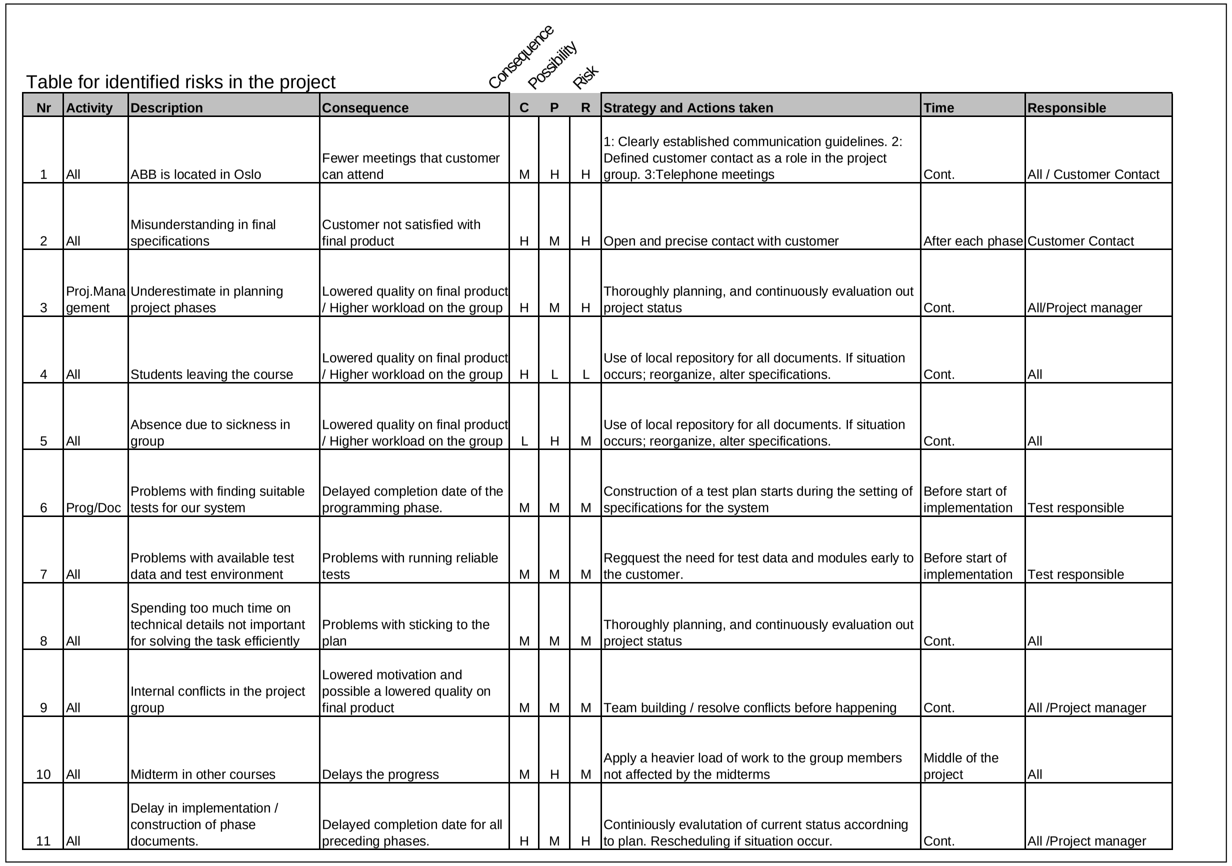


Table for identified risks in the project

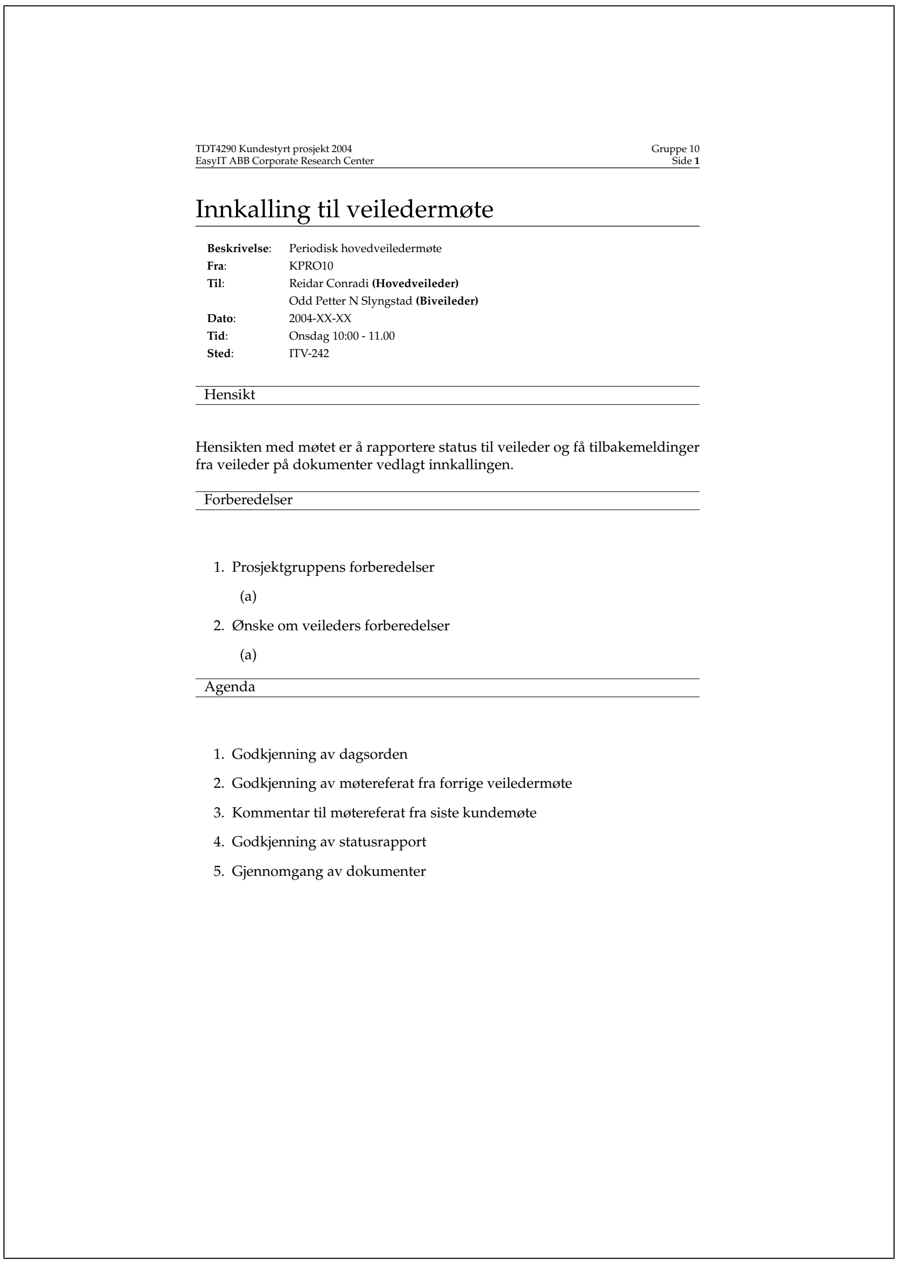
Consequence
Possibility
Risk

Nr	Activity	Description	Consequence	C	P	R	Strategy and Actions taken	Time	Responsible
1	All	ABB is located in Oslo	Fewer meetings that customer can attend	M	H	H	1: Clearly established communication guidelines. 2: Defined customer contact as a role in the project group. 3:Telephone meetings	Cont.	All / Customer Contact
2	All	Misunderstanding in final specifications	Customer not satisfied with final product	H	M	H	Open and precise contact with customer	After each phase	Customer Contact
3	Proj.Management	Underestimate in planning project phases	Lowered quality on final product / Higher workload on the group	H	M	H	Thoroughly planning, and continuously evaluation out project status	Cont.	All/Project manager
4	All	Students leaving the course	Lowered quality on final product / Higher workload on the group	H	L	L	Use of local repository for all documents. If situation occurs; reorganize, alter specifications.	Cont.	All
5	All	Absence due to sickness in group	Lowered quality on final product / Higher workload on the group	L	H	M	Use of local repository for all documents. If situation occurs; reorganize, alter specifications.	Cont.	All
6	Prog/Doc	Problems with finding suitable tests for our system	Delayed completion date of the programming phase.	M	M	M	Construction of a test plan starts during the setting of specifications for the system	Before start of implementation	Test responsible
7	All	Problems with available test data and test environment	Problems with running reliable tests	M	M	M	Regquest the need for test data and modules early to the customer.	Before start of implementation	Test responsible
8	All	Spending too much time on technical details not important for solving the task efficiently	Problems with sticking to the plan	M	M	M	Thoroughly planning, and continuously evaluation out project status	Cont.	All
9	All	Internal conflicts in the project group	Lowered motivation and possible a lowered quality on final product	M	M	M	Team building / resolve conflicts before happening	Cont.	All /Project manager
10	All	Midterm in other courses	Delays the progress	M	H	M	Apply a heavier load of work to the group members not affected by the midterms	Middle of the project	All
11	All	Delay in implementation / construction of phase documents.	Delayed completion date for all preceding phases.	H	M	H	Continiously evalutation of current status accordning to plan. Rescheduling if situation occur.	Cont.	All /Project manager

Figure C.1: Our risk diagram

APPENDIX D
TEMPLATES

This section includes templates used for communication with our customer and supervisor.



TDT4290 Kundestyrt prosjekt 2004 — Gruppe 10
EasyIT ABB Corporate Research Center — Side **1**

Innkalling til veiledermøte

Beskrivelse:	Periodisk hovedveiledermøte
Fra:	KPRO10
Til:	Reidar Conradi **(Hovedveileder)**
	Odd Petter N Slyngstad **(Biveileder)**
Dato:	2004-XX-XX
Tid:	Onsdag 10:00 - 11.00
Sted:	ITV-242

Hensikt

Hensikten med møtet er å rapportere status til veileder og få tilbakemeldinger fra veileder på dokumenter vedlagt innkallingen.

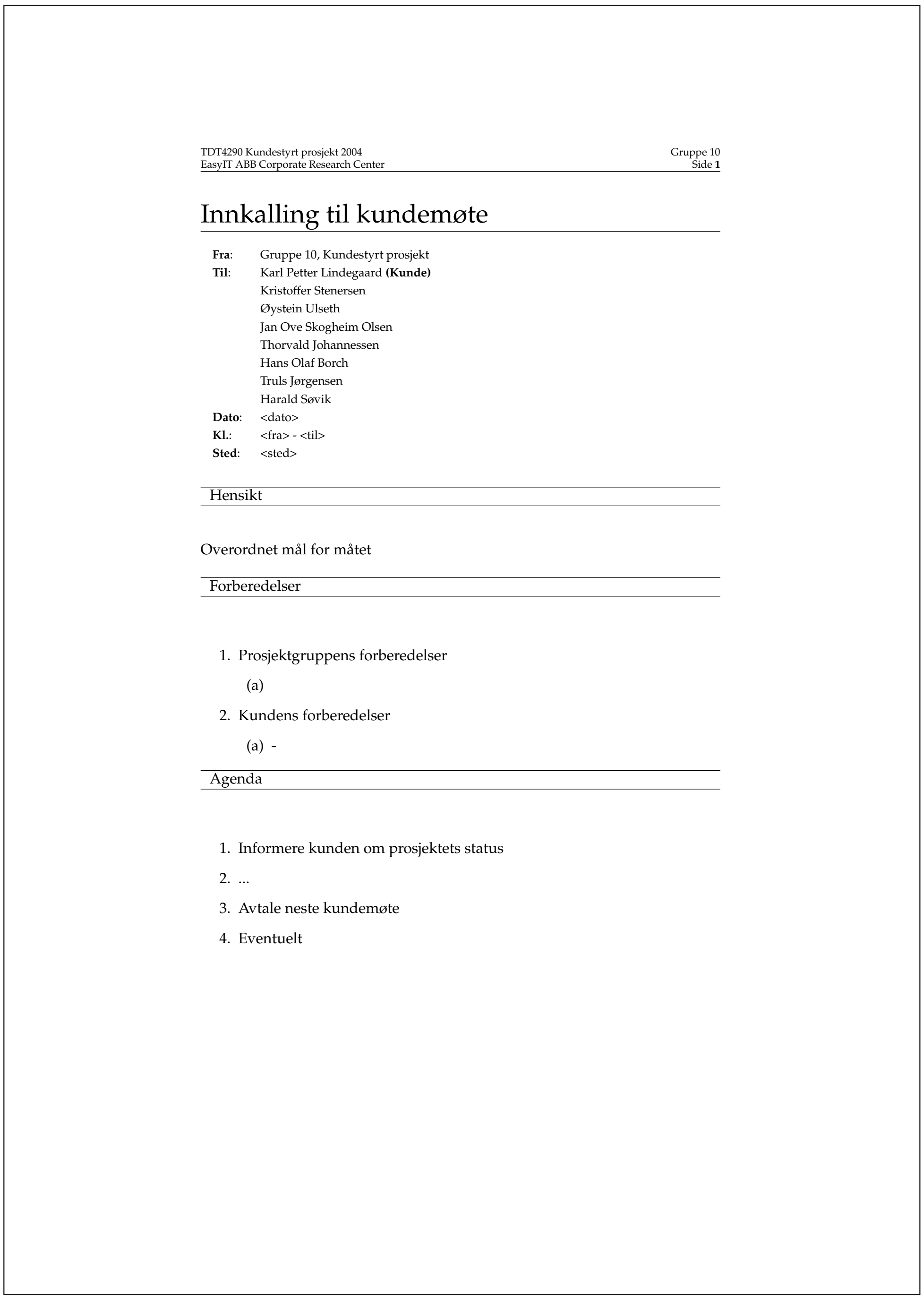
Forberedelser

1. Prosjektgruppens forberedelser
   (a)
2. Ønske om veileders forberedelser
   (a)

Agenda

1. Godkjenning av dagsorden
2. Godkjenning av møtereferat fra forrige veiledermøte
3. Kommentar til møtereferat fra siste kundemøte
4. Godkjenning av statusrapport
5. Gjennomgang av dokumenter

Figure D.1: The template for notice of meetings with the advisors



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **1**

Innkalling til kundemøte

Fra:	Gruppe 10, Kundestyrt prosjekt
Til:	Karl Petter Lindegaard **(Kunde)**
	Kristoffer Stenersen
	Øystein Ulseth
	Jan Ove Skogheim Olsen
	Thorvald Johannessen
	Hans Olaf Borch
	Truls Jørgensen
	Harald Søvik
Dato:	<dato>
Kl.:	<fra> - <til>
Sted:	<sted>

Hensikt

Overordnet mål for måtet

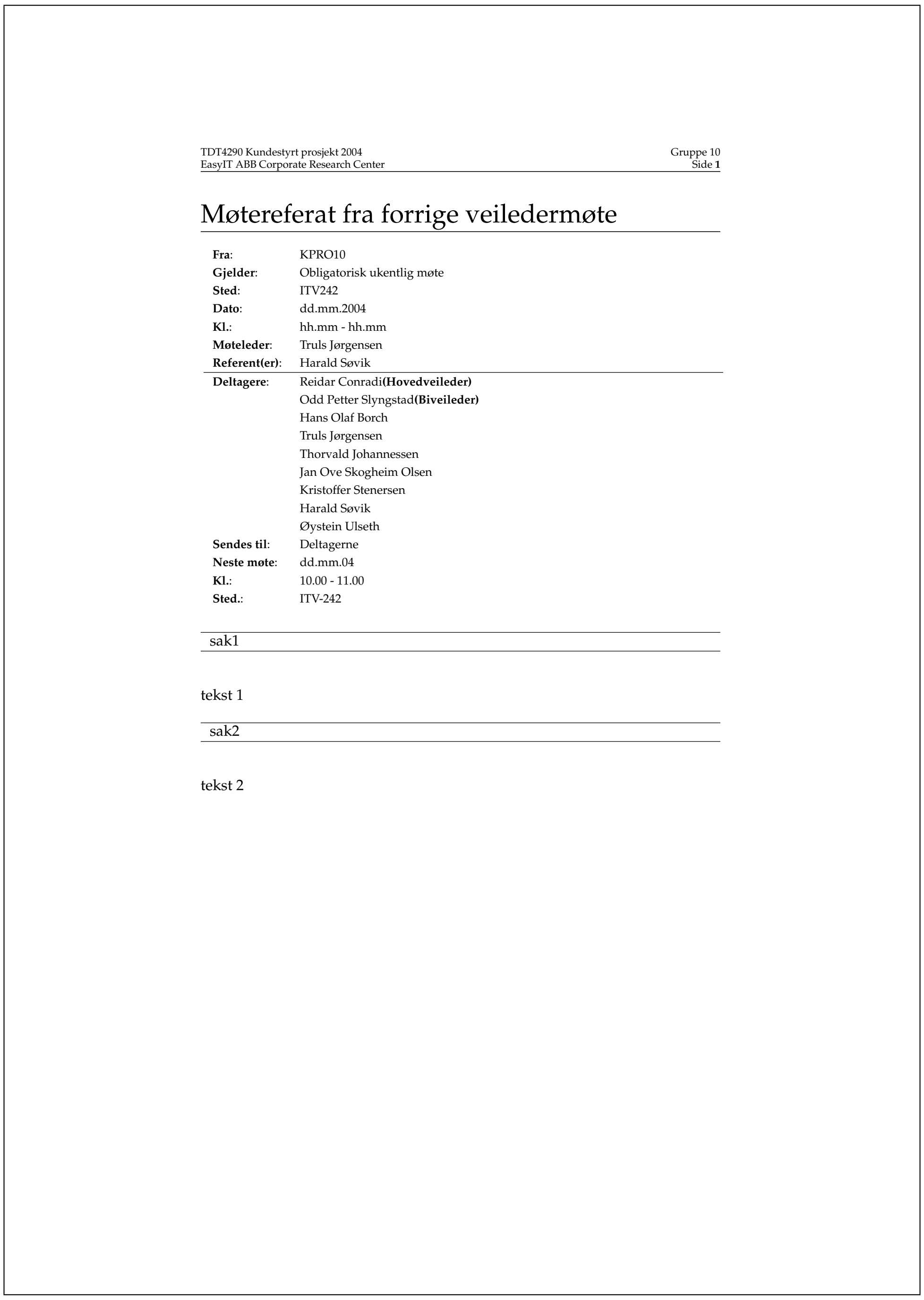
Forberedelser

1. Prosjektgruppens forberedelser
   (a)
2. Kundens forberedelser
   (a) -

Agenda

1. Informere kunden om prosjektets status
2. ...
3. Avtale neste kundemøte
4. Eventuelt

Figure D.2: The template for notice of meetings with the customer


TDT4290 Kundestyrt prosjekt 2004 — Gruppe 10
EasyIT ABB Corporate Research Center — Side **1**

Møtereferat fra forrige veiledermøte

Fra:	KPRO10
Gjelder:	Obligatorisk ukentlig møte
Sted:	ITV242
Dato:	dd.mm.2004
Kl.:	hh.mm - hh.mm
Møteleder:	Truls Jørgensen
Referent(er):	Harald Søvik
Deltagere:	Reidar Conradi**(Hovedveileder)**
	Odd Petter Slyngstad**(Biveileder)**
	Hans Olaf Borch
	Truls Jørgensen
	Thorvald Johannessen
	Jan Ove Skogheim Olsen
	Kristoffer Stenersen
	Harald Søvik
	Øystein Ulseth
Sendes til:	Deltagerne
Neste møte:	dd.mm.04
Kl.:	10.00 - 11.00
Sted.:	ITV-242

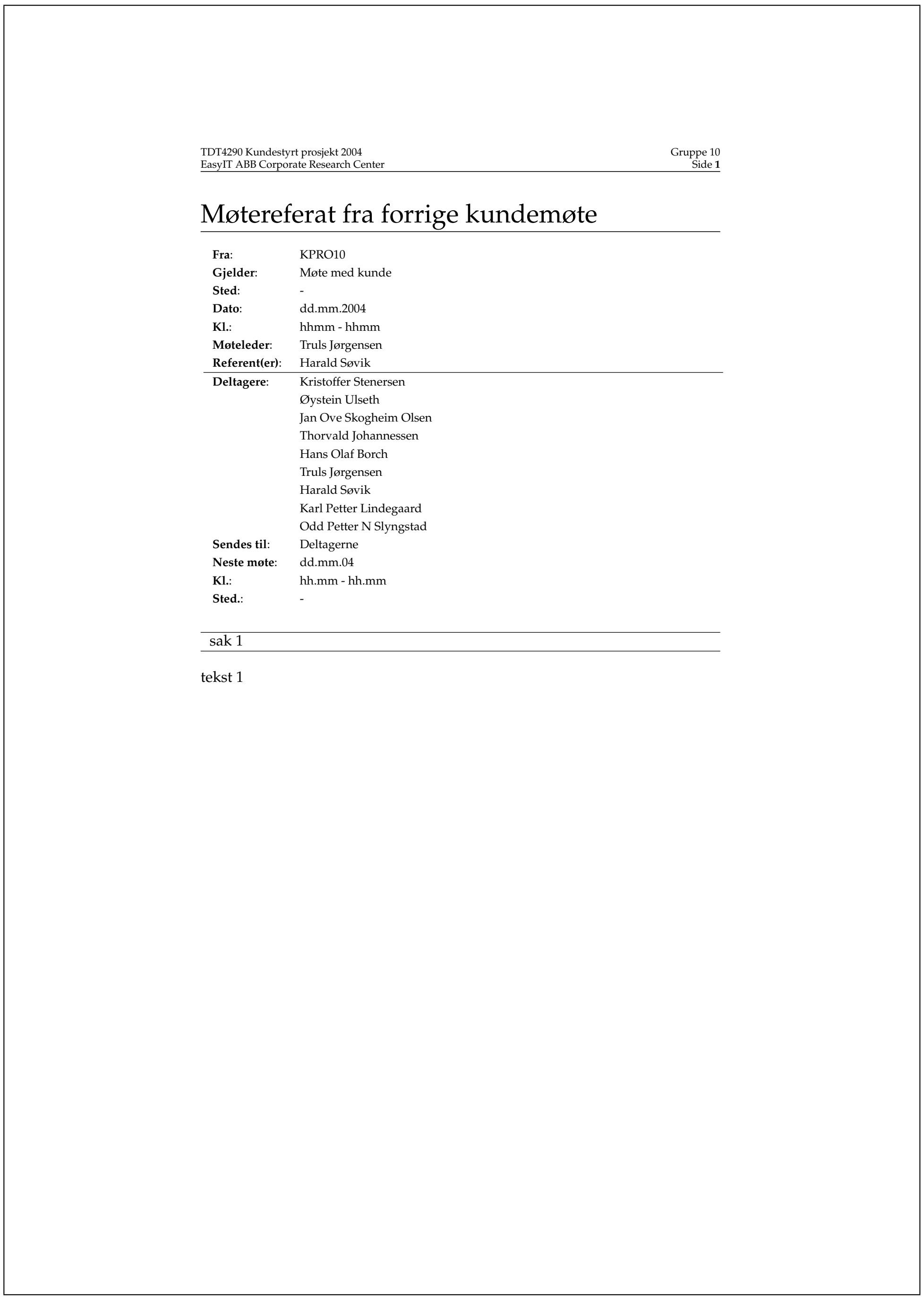
sak1

tekst 1

sak2

tekst 2

Figure D.3: The template for minutes from the meeting with the advisors



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **1**

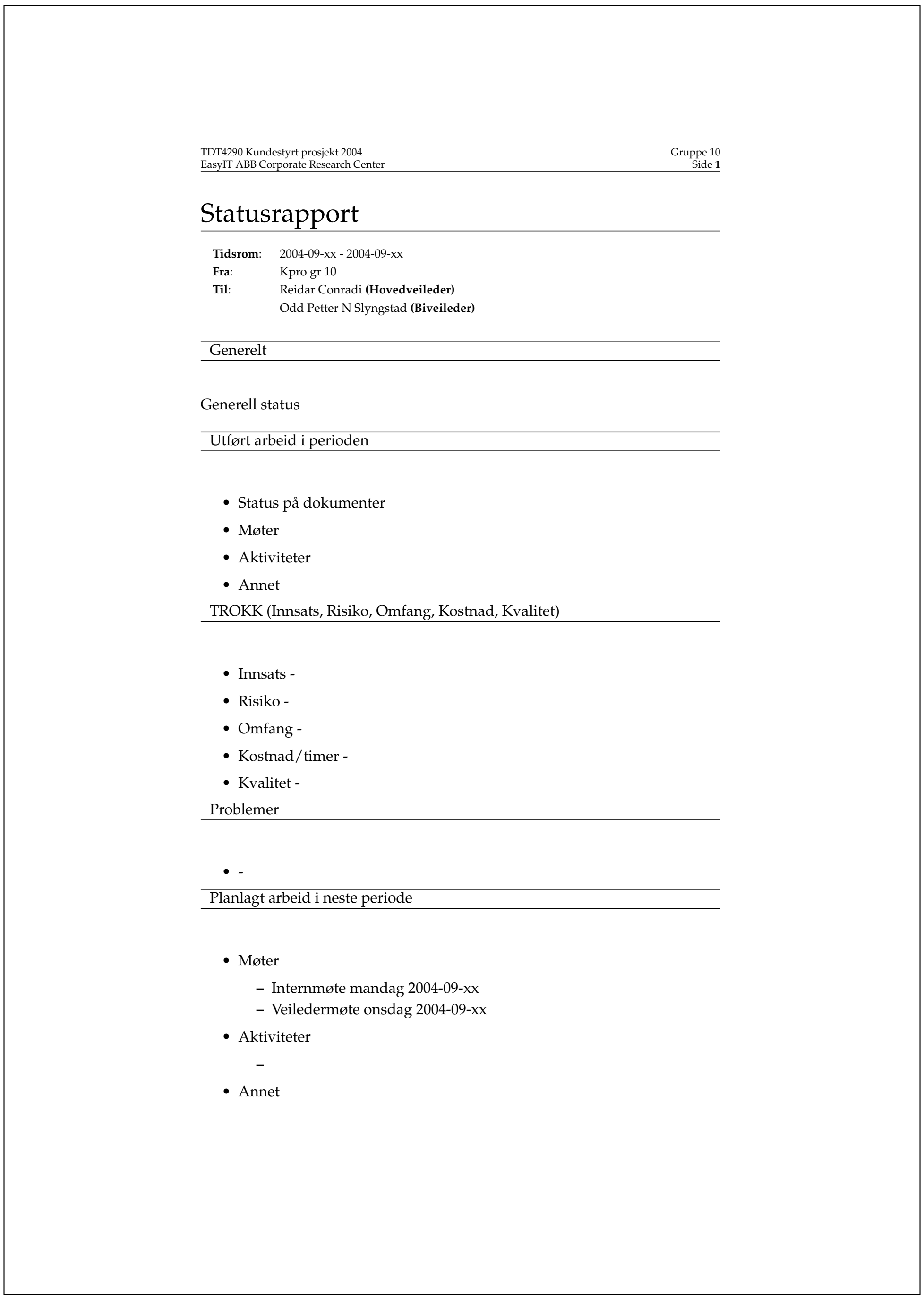
Møtereferat fra forrige kundemøte

Fra:	KPRO10
Gjelder:	Møte med kunde
Sted:	-
Dato:	dd.mm.2004
Kl.:	hhmm - hhmm
Møteleder:	Truls Jørgensen
Referent(er):	Harald Søvik
Deltagere:	Kristoffer Stenersen
	Øystein Ulseth
	Jan Ove Skogheim Olsen
	Thorvald Johannessen
	Hans Olaf Borch
	Truls Jørgensen
	Harald Søvik
	Karl Petter Lindegaard
	Odd Petter N Slyngstad
Sendes til:	Deltagerne
Neste møte:	dd.mm.04
Kl.:	hh.mm - hh.mm
Sted.:	-

sak 1

tekst 1

Figure D.4: The template for minutes from the meeting with the customer



TDT4290 Kundestyrt prosjekt 2004
EasyIT ABB Corporate Research Center

Gruppe 10
Side **1**

Statusrapport

Tidsrom:	2004-09-xx - 2004-09-xx
Fra:	Kpro gr 10
Til:	Reidar Conradi **(Hovedveileder)**
	Odd Petter N Slyngstad **(Biveileder)**

Generelt

Generell status

Utført arbeid i perioden

- Status på dokumenter
- Møter
- Aktiviteter
- Annet

TROKK (Innsats, Risiko, Omfang, Kostnad, Kvalitet)

- Innsats -
- Risiko -
- Omfang -
- Kostnad/timer -
- Kvalitet -

Problemer

- -

Planlagt arbeid i neste periode

- Møter
  - Internmøte mandag 2004-09-xx
  - Veiledermøte onsdag 2004-09-xx
- Aktiviteter
  -
- Annet

Figure D.5: The template for status reports

Part II

Pre study

Chapter 9
Introduction

9.1 Purpose

This part is a documentation of the process the project goes through in gaining an understanding of the problem, an overview of the possible solutions, the business demands, evaluation criteria, political and legal considerations. By reading this, the reader should be able to understand the major problems that must be solved and see different alternatives with their cons and pros.

9.2 Overview

The first part of this document consist of an outline of the current situation, the situation with an ideal solution applied, and what this ideal solution can consist of. Second, we formalize the parts of the ideal solution and transform them to evaluation criteria which can be applied to technical solutions. There has been some research performed on the current market to map already existing relevant solutions. Third, each possible solution is evaluated and one solution is chosen.

Chapter 10
Current situation

10.1 Introduction

ABB is originally a power company, but its main activities today concerns energy and automation systems. Our focus will be towards automation systems for monitoring and control of systems like chemical factories, oil installations and ships.

In an industrial process, the state is measured continuously by different sensors. These sensors can measure parameters like for instance the pressure in a pipe or the speed of a ship. All of these processes generate enormous amounts of data that need to be logged and organized. Logged data are used both for actively controlling and adjusting the systems, and for reporting and analyzing the performance of the systems.

Earlier a single contractor would typically provide the total engineering of the automation system for controlling and monitoring such industrial processes. Each vendor would use their own protocols for communicating between the different parts of the system. ABB for instance, had their own real-time network protocol for information exchange between nodes in the control network.

The most obvious problem with these proprietary protocols was incompatibility with other vendors. This increases both the short and long term cost of the installation. This is because each vendor has their own development- and maintenance departments, and support and service on existing installations can almost exclusively be performed by the installation contractors themselves.

In today's market the focus is towards open signal standards for exchanging process data. This applies especially to the bottom layers of the technology, making tough demands on resilience and response rates. A standard of current interest is OLE for Process Control (OPC), built on Microsoft's COM technology. This technology is covered in detail in the chapter on OPC later in the pre study.

10.2 Technical overview

A typical network for gathering data from industrial sites works as follows. The bottom layer consists of gauges connected to metering stations, which produce data tags. These run on real time operating systems like ViexWorks, connected to a control net for communicating with upper layers. This network use a real time protocol for exchanging data instead of OPC, because of the high real time demands. Connected to the control net are several application servers and databases for assembling and publishing the data. This middle layer contains the so called Information Management System (IMS). The IMS is responsible for logging the data tags from the bottom layers, and typically uses the Windows platform. The application servers provide the operating stations with a common interface for exchanging data through OPC. The top layer consists of operating stations, which can be seen as control panels. These stations are used for controlling the different field devices. A conceptual model of this network is shown in Figure 10.1.

The metering stations output a vast number of tags, and have great demands on real time processing. That is why the controllers run specialized real time operating systems. The control network connections are redundant to be able to withstand breakdowns. The operating stations have fewer requirements to real time data, and are typically running Windows. The reason for choosing the Windows platform in the upper layers is simply that most operators are already familiar with the user interface. The migration to Windows at the operator stations has in time caused the migration from UNIX to Windows in the middle layer as well. The communication between the upper layers depends

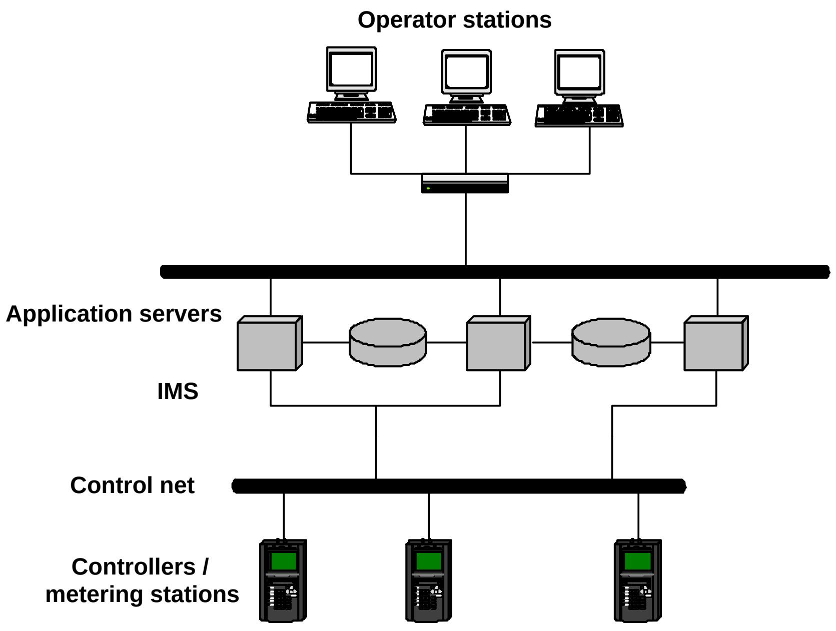


Figure 10.1: Conceptual model of the current situation

heavily on OPC, which is the standard of choice in the industry today. As mentioned different closed standards were used earlier, but today the performance and widespread use of the open standard OPC causes it to be the only actual alternative.

10.3 Problems

The openness of the OPC standard comes at a price, because some software companies specialize in developing general purpose applications that are cheap and run on any OPC server. The cost for a company like ABB to produce custom-made solutions for one of its customers is not competitive. This creates a need for ABB to offer a general framework for publishing its data to customer-built applications.

The operator station software mainly consists of Visual Basic 6.0 applications and ActiveX-controls. The administration of the upper layers is costly because of the complexity in accessing the data from the application servers. Another problem is the scalability of Visual Basic applications, causing trouble when lots of applications are run in parallel on the data. None of today's systems use Microsoft's .NET technology, but it is to be introduced throughout ABB's systems within 2006.

ABB offers a system called Asset IT as a part of their automation software package Industrial IT. This software is still being developed, and therefore it has several drawbacks. As of today, it does for instance not support automatic monitoring of the status of the control devices directly (asset monitoring). Instead, an application for monitoring the age of the equipment is incorporated into the software. The complex algorithms needed for asset monitoring are still to be developed. Industrial IT is a large platform, and it is costly to implement. An alternative is to use stand-alone applications like Petrobase. These again come with their own drawbacks. Petrobase for instance doesn't support sampling rates shorter than 60 seconds, which is too coarse grained for some applications.

The IMS as described above performs logging of the data, and saves historical data over longer periods of time. The problem is that as a lot of data is stored for longer periods of time, the IMS will decrease the sampling rate to save storage space. This uncertainty in the sampling rate of the data you need, can cause problems for some applications relying on detailed historical data. The IMS hardware is expensive, and the software is rather complex to reconfigure.

The complexity of today's systems enforces manual work processes, in data access as well as data analysis. People even have their own Excel spreadsheets for importing data manually and performing ad-hoc plotting and analysis. This of course affects both the performance and accuracy of the analysis.

10.4 TODAY'S WORK PROCESSES

The functionality in the ABB automation system we are to improve, consists mainly of the following three tasks:

- Performing specific data analyses, including both gathering the appropriate data tags and storing the results
- Asset monitoring, being able to monitor the state of the control devices in an appropriate way
- Performing periodic calculations, showing all kinds of data and interesting results as various graphical plots

Gathering of data is being carried out manually in today's system. To access data logged by the Information Management System, you would have to perform some sort of query on the application servers. The result of this query is a comma-separated list, which can then be exported manually to for instance MS Excel. As the data might be spread out over multiple servers, one might have to manually assemble series of data gathered from multiple queries. This is of course a process prone to human error.

If you want to do asset monitoring, the Asset IT system of today provides you with relatively limited support. As mentioned above, it only supports monitoring the age of the equipment. Without a software tool for monitoring the status of control devices, operators are forced to analyses the situation manually. This is done for instance by analyzing the equipment age data available from Asset IT, and calculating the expected operating time remaining for the components manually. This of course is hardly as accurate as having an automated procedure, and it is quite time consuming as well.

To access for instance weekly calculation results from monitoring the input and output of a specific industrial process in today's system, you would have to start by acquiring a comma-separated list as described above. Then you would use a tool like MS Excel to create a custom made plot, either manually or with the help of a prewritten macro.

10.5 TODAY'S USE CASES

This section illustrates how the main tasks are carried out today. The term "manually" is often used, meaning without the support of a software tool. For instance performing an analysis manually means having to assemble the data and setting up the analysis by hand, and creating a custom plot from scratch in for instance MS Excel. On the contrary, performing it automatically means having automated procedures for doing this integrated into a user interface.

Use case 1

Describes performing general data analyses without software support. See Figure 10.2.The analysts first queries the IMS, the exports the resulting comma-separated list to MS Excel. Then he reassembles the data, if he had to perform multiple queries. The analysis then uses his own methods for analyzing the data and storing the results. These include using his own Excel macros or creating custom-made graphical plots.

Use case 2

Describes performing asset monitoring manually. See Figure 10.3.To perform asset monitoring, the analyst needs to acquire lists with information about equipment age et cetera through a manual routine. He then scans these lists manually in order to monitor the age of equipment, in search of components so old that they need replacement. The results are published manually. An operator then needs to find these published results and take the appropriate action (for instance ordering new components).

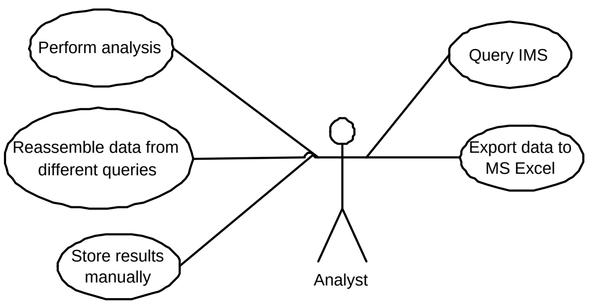


Figure 10.2: Pre study: Use case 1: Performing analysis today

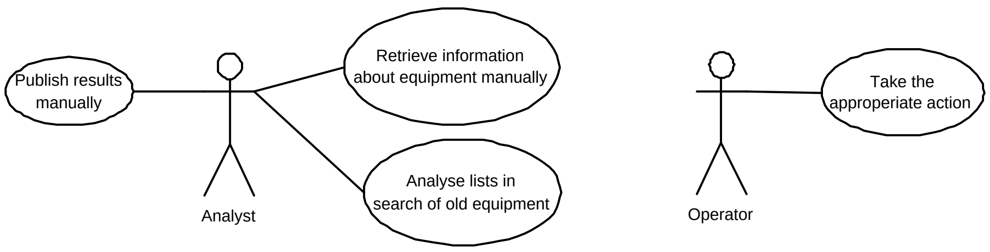

Publish results manually
Analyst
Retrieve information about equipment manually
Analyse lists in search of old equipment
Operator
Take the approperiate action

Figure 10.3: Pre study: Use case 2: Asset monitoring today

Use case 3

Describes performing periodic calculations and publishing the results. See Figure 10.4.As above, the analyst queries the IMS, exports the data to Excel and reassembles the data. If a complex macro is needed, a programmer would typically supply the macro. Then the analyst can then analyses the data using the macro. He stores the results of the analysis and publishes them manually. Other analysts wanting to view the results need to acquire the manually. This process is then repeated each time the results need to be updated.

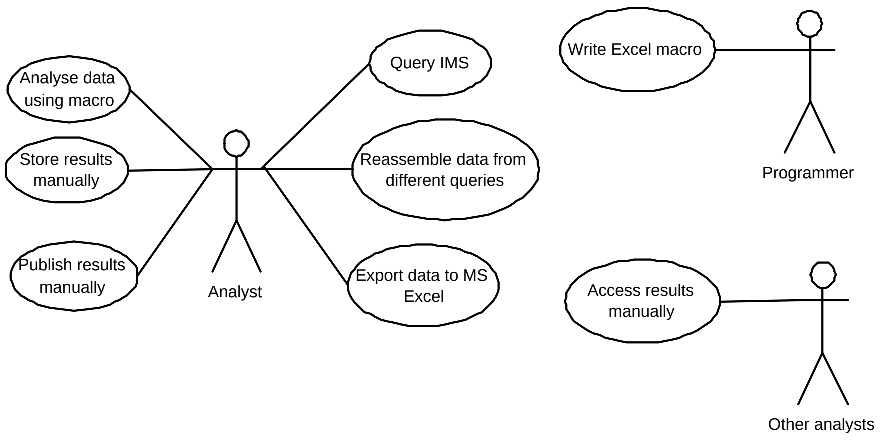


Figure 10.4: Pre study: Use case 3: Performing periodic caluculations today

Chapter 11
Desired solution

11.1 Introduction

As the data from the metering stations will be more easily accessible, ABB wants to process them further. The purpose of studying these historical data, describing the present and former running of the industrial processes, is to:

- Identify and characterize the process, for optimizing the production and thus increase the earnings.
- Generate reports to document the operating of the business. These are needed for instance when facing environmental requirements to traceability imposed by the government.
- Identify damaged or broken equipment, supporting improved maintenance procedures. This reduces down-time for the production unit, and thus increases the earnings.

Many of these activities can be automated, having predefined applications that output results of calculations automatically. Another application might be to allow ”analysts” to go through historical data manually in search of new ways of improving the processes.

11.2 System modules

The goal of our prototype is to develop a solution containing the following three modules:

- An infrastructure for gathering and storing large amounts of sampling and process data in an appropriate way.
- A framework for developing applications that shall operate on the collected data, and write results back to the system.
- A graphical user interface for presenting collected raw data, as well as the data generated by the applications mentioned above.

The customer's main interest is having a framework that is easily configurable, and easy to expand with new applications for utilizing the collected data. It should be user friendly when it comes to accessing the data, and cheap to implement.

11.3 Overall system description

ABB wishes to employ data from the metering stations in a wide range of analysis applications. The idea is to have a central application for logging data from different OPC servers. This information should be available through a generic interface, so that any number of analysis applications can utilize it without having to reconfigure the original application.

The applications will need data from different servers at different rates, so the data interchanger should be flexible and configurable with respect to the gathering of data. The applications should be able to subscribe to the data they want at their chosen rate.

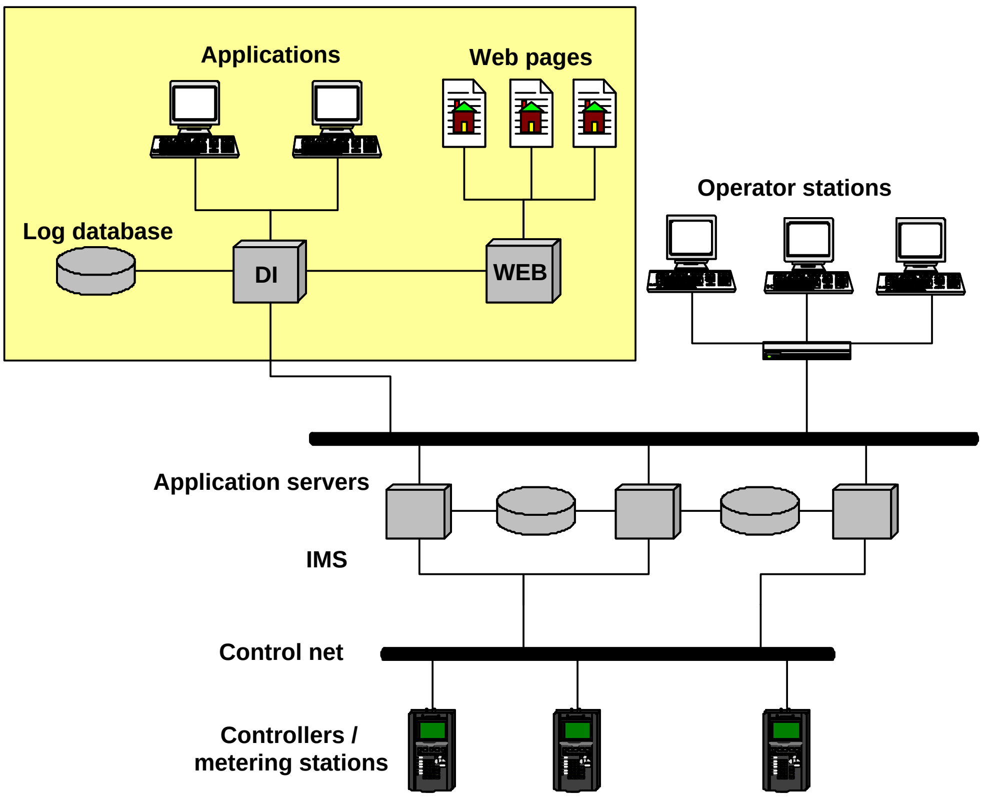


Figure 11.1: Conceptual model of system to be developed. The existing system is outside the box.

To gather the required data, a common interface to access different OPC servers is needed. The data interchanger for storing data from the OPC servers in a separate database will serve as a basis for publishing the data. This interchanger should also provide an interface to applications that need the data, both for analysis and for calculations that should be available to other applications.

In addition to this, a way to access the data tags directly through a simple user interface is requested. Analysts might want to access the data without having to install new software, and therefore it might be practical to develop an application accessed through the World Wide Web. A generic web portal is described as the most attractive solution by the customer. Each of the scheduled applications mentioned above will need its own web pages for displaying the data and results generated by the corresponding application, as well as historical data from the database. The portal should be able to present different results and data requested by the user, to serve as a basis for further analysis. Each web application will need to be configured in the same way as the standalone applications are.

A conceptual model of the overall structure is shown in figure 11.1. Our part of the system is inside the box.

A common Application Programming Interface (API) for the core operations needed by the three modules should be developed. The customers wishes as much as possible of the functionality to be configurable by an operator.

11.4 Desired work processes

As mentioned in the last chapter, the functionality in the ABB automation system we are to improve consists mainly of three tasks. These are repeated below, and in addition to these comes the last task:

- Performing specific data analyses, including both gathering the appropriate data tags and storing the results

- Performing periodic calculations, showing all kinds of data and interesting results as various graphical plots
- Asset monitoring, being able to monitor the state of the control devices in an appropriate way
- Writing complex applications that combine the available data to produce results that could not be extracted from the old system

Our objective is to simplify and automate these tasks by making the data more accessible. First of all the new applications can request historical data to be logged in the database, and thus we now longer have the uncertainty in connection with availability of data.

Performing various analyses on data series can be done in two ways. A general purpose analysis application can be written to support tasks like regression analysis et cetera with a graphical user interface. Another way would be to write small scheduled applications that publish the appropriate data series to the web portal. The analyst could then through the web portal either view the data as a graphical plot, or export the data for further analysis.

To generate results from specific calculations periodically, scheduled applications can be written to perform calculations on the data series available. For instance, one or more applications can be set up to output say the daily average flow of fluid in a pipe each week. The results will then, once set up correctly, be generated automatically as long as the applications runs. They can then easily be made available through the web portal, so that any analyst interested in the data can access it without having to install any new applications.

The process of asset monitoring can be greatly supported by an application using our API. Instead of having to scan data and monitor the age of the equipment manually, a more complex application for performing all monitoring tasks can be written to utilize the functionality offered by our system. Specialized algorithms for analyzing data series available from the equipment can be written to support estimation of remaining operating time for the unit. With more data available it is also easier to allow an application for instance to monitor the status of all the equipment on a specific industrial site.

The forth suggested task illustrates the potential flexibility in our new system. Allowing applications access to all logged data, developers might come up with new ways of combining the data to allow new information to be extracted from the system. For instance, applications for predicting future behavior of equipment based on historical data can be of great assistance in planning further activities.

11.5 Desired use cases

The following graphical use case diagrams illustrate how these new work processes could be implemented. Since our system will be generic, it allows a task to be carried out in more than one way. Therefore these use cases are just examples of how to use the system, not the only way to use it.

Pre study: Use case 4

Describes using a ”stand-alone” application for performing analyses. See Figure 11.2.A programmer starts by developing the application itself, and an administrator deploys the application in the appropriate place. Then the administrator configures the application to allow it to publish data on the web portal. The analyst then downloads the application and performs his analyses with the tools offered by the application. Other analysts can then view the data through the web portal.

Pre study: Use case 5

Describes using a scheduled application for outputting periodic results automatically. See Figure 11.3. A programmer develops applications that periodically processes data needed for the calculations. The application is then deployed and configured by an administrator. Analysts can then access the data through the web portal, and if needed, perform their own analyses.

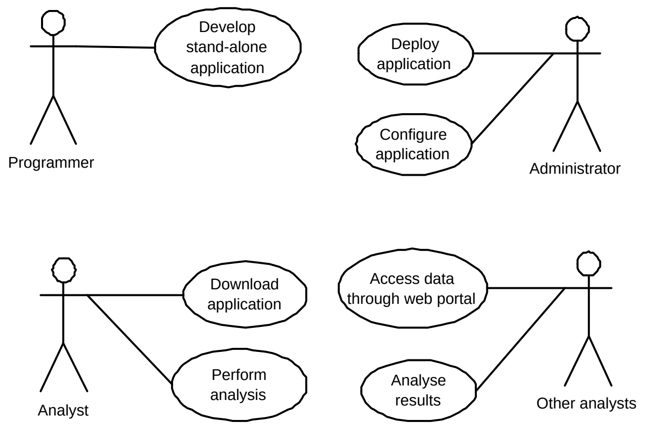


Figure 11.2: Pre study: Use case 4: Performing analyses using a stand-alone application

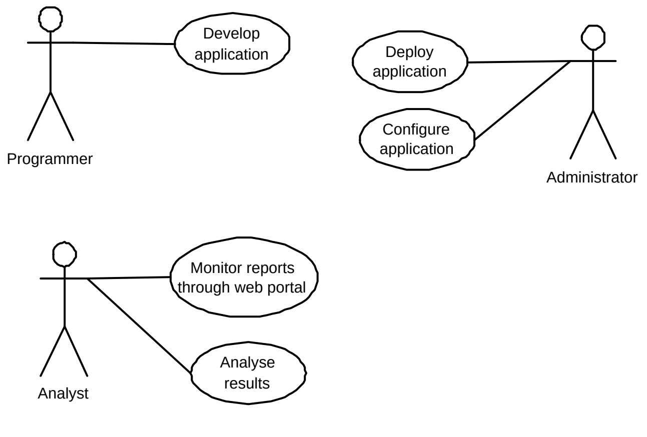


Figure 11.3: Pre study: Use case 5: Setting up and viewing results from automated calculations

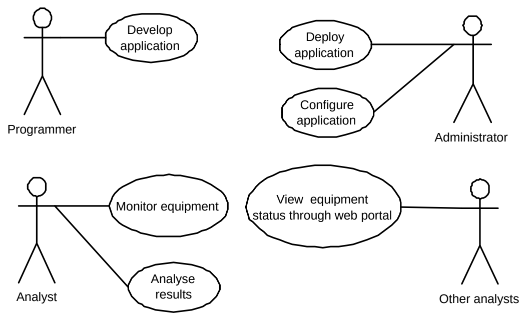


Figure 11.4: Pre study: Use case 6: Asset monitoring

Pre study: Use case 6

Describes using prewritten applications for asset monitoring. See Figure 11.4.As with the stand-alone application, it's written by a programmer and deployed and configured by an administrator. The monitoring of the equipment can then be done either through the application itself, or by monitoring the data available through the web portal.

Chapter 12
Business Related Requirements

12.1 Introduction

This chapter will describe the business related requirements of a solution to the customer‘s problem. These requirements are the basis for later functional requirements.

12.2 List of business related requirements

Based on our understanding of the problem, the list in table 12.1 contains the business requirements for the system.

Demand nr	**Description**
BRD-1:	The system must enable the user to access and collect real time numeric measurement data from several different sources.
BRD-2:	The system should enable the user to access historic numeric measurement data from several different sources.
BRD-3:	The protocol used to collect the measurements must be an open, non-proprietary standard(OLE for Process Control - OPC)
BRD-4:	The system must provide an easy and standardized way to develop applications that use the collected data.
BRD-5:	The system must provide scheduling functionality for the applications.
BRD-6:	The system must provide configuration functionality for the applications.
BRD-7:	The system must provide a GUI-gateway that enables the user to view collected data and data generated by applications. The gateway must be configurable to server specific content to specific users. Several simultaneous users must be able to access the gateway at the same time.
BRD-8:	The system must run in a environment primarily based on Microsoft products.
BRD-9:	The overall system must be inexpensive, as the competition is fierce.

Table 12.1: Business related demands

Chapter 13
Evaluation criteria

13.1 Criteria

Given the business related demands and further discussions with the customer, we have constructed a collection of evaluation criteria, see figure 13.1, for the different solutions. We have grouped the criteria in 4 different groups. The groups are based on the three modules of the system (DI, API and WEB) and some overall criteria for the entire system.

Criteria nr	**DI Criteria**
EC-1:	The system must be able to read OPC DA-data from several sources at different rates at the same time.
EC-2:	The system should be able to read OPC HDA-data from several sources at different rates at the same time.
EC-3:	The system must be able to write OPC DA-data.
EC-4:	The system must be able to store/log the data from EC1 and EC2.
Criteria nr	**API Criteria**
EC-5:	The system must make the data from EC1 and EC2 available for inspection by users and applications..
EC-6:	The system must let users and applications from EC5 to make calculations on the data from EC5..
EC-7:	The system must enable the results of the calculations from EC6 to be stored/logged just like EC4.
EC-8:	The system must be able to schedule applications to run at given intervals with collected data sets..
EC-9:	The system must have functionality for configuring applications to access it.
EC-10:	The system should allow inspection of and calculations on the historical data from EC4.
EC-11:	The system should expose the functionality in EC1 to EC10 through a programming API. The API should enable user to rapidly build applications to access it.
Criteria nr	**WEB Criteria**
EC-12:	The system should expose all collected and all calculated data in some kind of GUI-gateway for users to inspect it. Configurability is needed to allow specific data to be viewed by specific users only. Several users must be able to use the system at the same time.
Criteria nr	**Overall system criteria**
EC-13:	The system must run on Microsoft operating systems.
EC-14:	The system must not be very expensive.
EC-15:	The system should have the possibility to use other data sources than OPC.

Table 13.1: Evaluation criteria list

These criteria will be applied against the different possible solutions found during market research. An evaluation form, see figure 13.1, will be filled out for each of them. Table 13.2, is a quick overview of how each solution did in each criteria group. The basis for this table is the complete filled out forms that are included in F.1. For each of the possible solutions each criterion will be given a score from 1 to 5. 1 indicates a low score, the solution does not meet the requirements in this criterion at all. 5 means it fulfills all requirements of the criterion.

Product	
Producer	

Criteria	**What**		**Degree of fulfillment**
	DI criteria		
EC1	Read OPC-DA		
EC2	Read OPC-HDA		
EC3	Write OPC-DA		
EC4	Log data		
	API criteria		
EC5	Make available to users/app.		
EC6	Make calculations		
EC7	Store result of calculations		
EC8	Schedule applications		
EC9	Configurability		
EC10	Historical data		
EC11	API. Rapid development.		
	WEB criteria		
EC12	GUI-gateway		
	Overall system criteria		
EC13	Run on Microsoft		
EC14	Inexpensive		
EC15	Other data sources		
	Conclusion		

Figure 13.1: Evaluation form

Solution	**Conclusion**	**DI-group**	**API-group**	**WEB-group**	**Other criteria-group**
		Degree of criteria fulfillment			
DAQBench	The limitation of only making drag and drop components available make this solution undesirable for us.	Medium	Low	Medium	Low
GDAC	This system does not implement any logging functionality. This is not acceptable, no further evaluation is necessary. System discarded.	Low	Low	Low	High
LabView	International Instruments' LabView fulfills most of the evaluation criteria, apparently making it a very good solution candidate. However it's a large, extensive and expensive software, and thus it does not fulfill the inexpensive requirement.	High	High	High	Low
Matrikon ProcessNet	Matrikon ProcessNet fulfills most of the evaluation criteria, apparently making it a very good solution candidate. However it does not allow developers to access the collected data through an API, something which is one of the fundamental requirements for our task.	High	High	High	Low
OPC DA/HDA Archiver	Meets several of the important criteria, but does not fulfill crucial ones like exposing it's functionality through a programmable API for further application development.	High	Low	Medium	Low
OPC Systems Eldridge Engineering Comp. Inc	Offers some required functionality, but both the missing functionality and the prize makes this solution unacceptable to us.	High	Low	Medium	Low
OPC Toolbox for Matlab	Fulfills many requirements, but does not offer functionality like scheduling and extensive API outside Matlab. Requires Matlab, which is expensive.	High	Medium	Low	Medium
Iconics OPC Web client	Same problem as TrendWorX32 from same producer. No logging ability makes this system completely unacceptable	LOW	NA	NA	NA
OPenDA	No logging ability means the system will fail most of the other criteria. No further evaluation needed, system discarded.	Low	NA	NA	NA
PI Advanced Computing Engine	This solution looks promising, but the lack of schedulability functionality makes it undesirable.	High	Medium	High	Low
Trend software	Trend software fulfills many of the requirements, but the lacking API and the absence of scheduling functionality makes it unacceptable as a solution.	High	Medium	High	Low
Iconics TrendWorX32	No logging ability makes this system completely unacceptable. No further evaluation needed.	Low	NA	NA	NA

Table 13.2: Existing solutions evaluation summary

Chapter 14 Market research

14.1 Introduction

In this chapter we will do research to find out if there already exists a solution suitable for ABB. This is important for us in such a way that we are aware of what others offer, and their specifications may also impact our final choice of solution. We used mostly the Google search engine for finding software, but also opcfoundation.org's [10] product list. The search word we used when using Google was 'OPC'.

Historical view
The continuous development of both computer hardware and software has caused a constant change in the units offered for automation of process plants. A common situation today is that automation systems often include technologies from different eras and different providers. Compatibility between the different layers and physical units in the control system network has become a major issue for maintenance and further development of these systems. One of the main barriers is the numerous different data protocols used to access these units.

OPC and standardization
The high maintenance cost of automation systems has cleared the path for solutions like OPC. The development of OPC exemplifies the process that has been going on the past few years. Major companies puts a big effort in standardization of their products. While other minor companies exploits the current situation, by offering solutions that eases the integration of different technologies. The standardization that comes with OPC does not only provide lower maintenance cost, but also new functionality through the simplicity in developing new applications.

Demand for increased functionality
This situation creates a customer demand for increased functionality and flexibility. This market can be served by the major companies in automation systems, as well as new third party developers. The demand for development of our product is a result of this process, and it already exists several solutions. Many of them benefit from the new standardized OPC solution.

14.2 Solutions comparable to EasyIT

14.2.1 Quick overview

A quick summary of the products we looked into:

- Open Source: GDAC: Cross platform basis for custom OPC servers.
- Open Source: OpenDA: OPC DA server. Most functionality is not yet implemented.
- ProcessNet from Matrikon: Extensive toolbox. Among these, drag and drop development of process data web pages.
- Trend Software from Canary Labs: Tools for storing tags, calculate on, viewing and reporting them, as well as Internet access to them.

- OPC Toolbox from MathWorks: OPC plug in for MatLab. Gives MatLab read and write access to OPC DA.
- OPC Systems from Eldridge Engineering Inc: Packages for real-time trending and data logging.
- DAQBench from Agile Integration: Enables easy creation of activeX controls for measurement and automation of OPC systems.
- TrendWorX32 from Software Toolbox: An OPC client implemented with ActiveX/OLE Object Technology. The client provides trend analysis, of both historical and real time data.
- PI Advanced Computing Engine from OsiSoft: Allows programming of calculations, communication applications, data transfer programs. It allows users to develop calculations in Visual Basic. ACE consists of three components: Scheduler, Manager, and Wizard.
- LabView from National Instruments: NI LabVIEW is the graphical development environment for creating, test, measurement and control applications rapidly.

14.2.2 About open source software

The following two reviewed software solutions are open source. One definition of open source is "Generically, open source refers to a program in which the source code is available to the general public for use and/or modification from its original design free of charge, i.e., open. Open source code is typically created as a collaborative effort in which programmers improve upon the code and share the changes within the community. Open source sprouted in the technological community as a response to proprietary software owned by corporations." [9]

14.2.3 GDAC

GDAC was developed at CERN where its original name was SLiC. The GDAC (Generic Device Access and Control) framework is an extensible multithreaded cross platform (Linux (x86,VME)/Windows) C++ device control environment. It typically will be used as the basis for custom OPC servers or front-end control software.

It was developed to be used as the basis of software to control hardware devices for which there is no control solution commercially available. In general vendors of hardware devices normally provide OPC servers to make their devices accessible via the network. But as OPC is based on Microsoft's OLE technology this solution does not work on Linux VME crates which are quite commonly used in high energy physics experiments. Besides that the detector teams in a physics experiment often build their hardware themselves and they need to provide the control software (even if it is an OPC server) for these devices, too. [2]

14.2.4 OPenDA

A .NET toolkit providing publish subscribe and/or request response based Web Services. OPenDA [3] (OPc Data Access) implements the XML OPC DA middleware standard for industrial process control. Suitable for information distribution applications. It implements the OPC DA 1.0 standard.

This project aims to produce an OPC DA server which is embeddable in a micro controller. Currently version is 0.2 alpha, and most functionality is not yet implemented.

14.2.5 About commercial software

The rest of the software reviewed is commercial software. Commercial software is software that is neither freeware nor shareware. Companies or individuals produce commercial software in order to generate profit. Commercial software is almost never open source software. The intellectual property associated with commercial software is usually protected by copyrights and patents.

14.2.6 ProcessNet from Matrikon

From Matrikon's web page on ProcessNet [20]: "ProcessNet is an out-of-the-box decision support solution that delivers web-based access to process data in real time, regardless of the data's original format or operating system. ProcessNet securely distributes your data to whoever needs it, wherever they are."

ProcessNet has tools for drag and drop development of process data web pages, tools for report organization, suitable for integrating with Microsoft Word and Excel and customizable trending tools. It gives access to multiple data sources through a single user interface, the web browser; HTTP/XML over Internet. One part of ProcessNet, NetDraw, is a thin client process graphics tools for creating graphical representations of data series and sensor data. Has drag and drop support; non-technical users should be able to create their own web pages. A screen shot of NetDraw is shown in Figure 14.1 below.

All together five modules put together ProcessNet. One other module is NetTrend, a tool for analyzing trends in process data. A screen shot of NetTrend is shown in Figure 14.2 below. Figure 14.3 gives a system overview of ProcessNet's system architecture. The ProcessNet server gives a common interface for all low level producers of process data, but also a common interface for the application modules mentioned above.

14.2.7 OPC DA HDA Archiver from Integration Objects

Integration Objects [21] offers several software products for connectivity solutions between real-time devices and applications. The most noticeable in this discussion are perhaps OPC Data Access / Historical Data Access Archiver. This tool allows you to "interface with several OPC servers for accessing production data from various processes and archives this data into CSV, XML files or an ODBC-compliant database. This archive can be used for trend analysis..." It also allows write backs to the network, synchronous or asynchronous." Figure 14.4 shows some screen shots of the application

Further on, Integration Objects also offers an "OPC Data Access Explorer". This tool "allows you to interface with several OPC DA servers for accessing production data from the various process and automation data sources connected to your network."

14.2.8 Trend Software from Canary Labs

Canary Labs [19] is a relatively small company located in Pennsylvania. Canary Labs' product family is built around OPC technology. It's divided into four parts: Trend Historian, Trend Calc, Trend Link and Trend Web.

The Trend Historian software acquires OPC tags and stores them in a historical database and is compliant with OPC historical data access. It provides setup for different number of tags to be stored and has a capacity of 16,000 tags/second in a sustained logging scene. It also support export to Microsoft Excel. Figure 14.5 shows a screen shot from the logger part of this software's setup screen.

Trend Calc is a calculation engine, built on top of the Trend Historian. Example applications of this tool is to scale sensor input, compute difference between inputs, use inputs in mathematical formulas, compute averages etc. Figure 14.6 shows a screen shot of the Trend Calc with examples of different tag calculations.

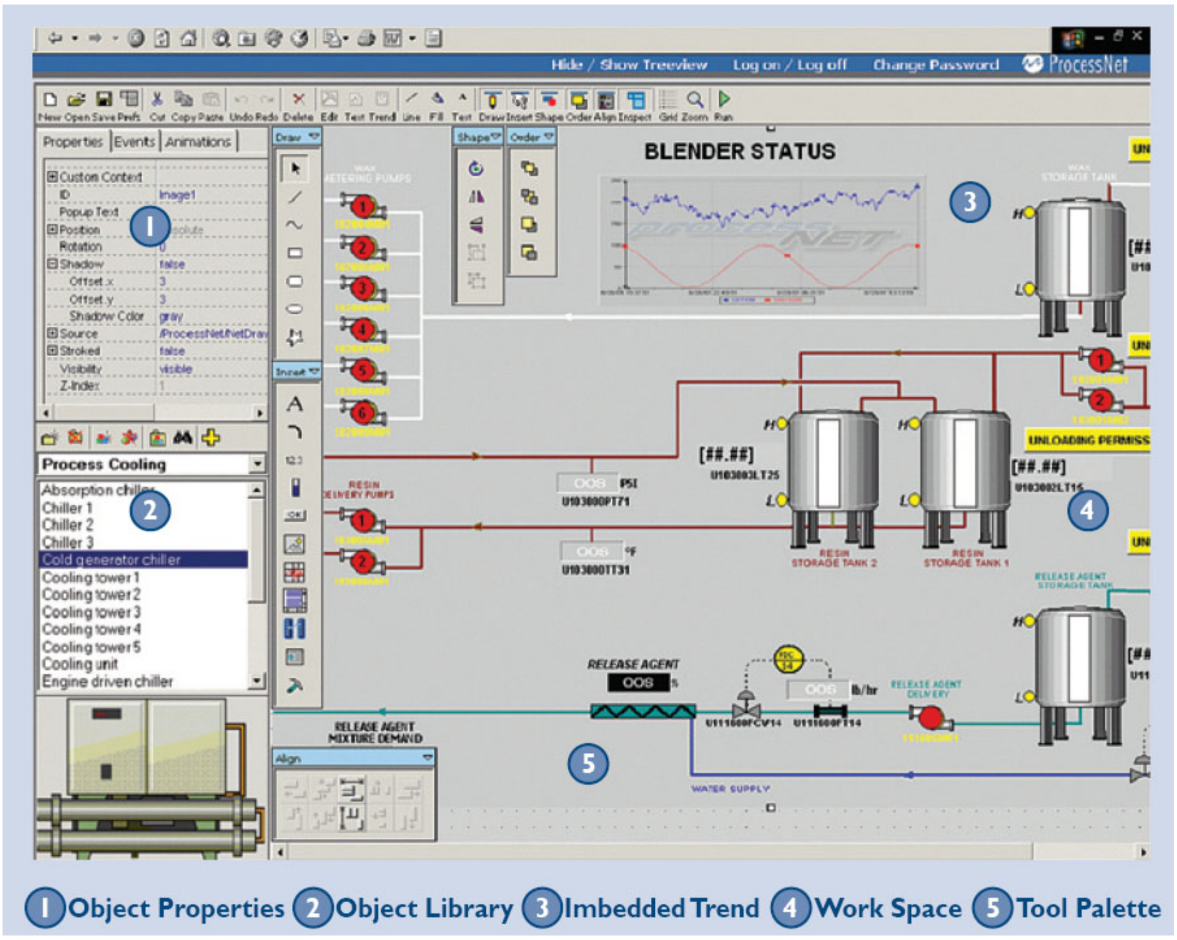


Figure 14.1: ProcessNet's NetDraw

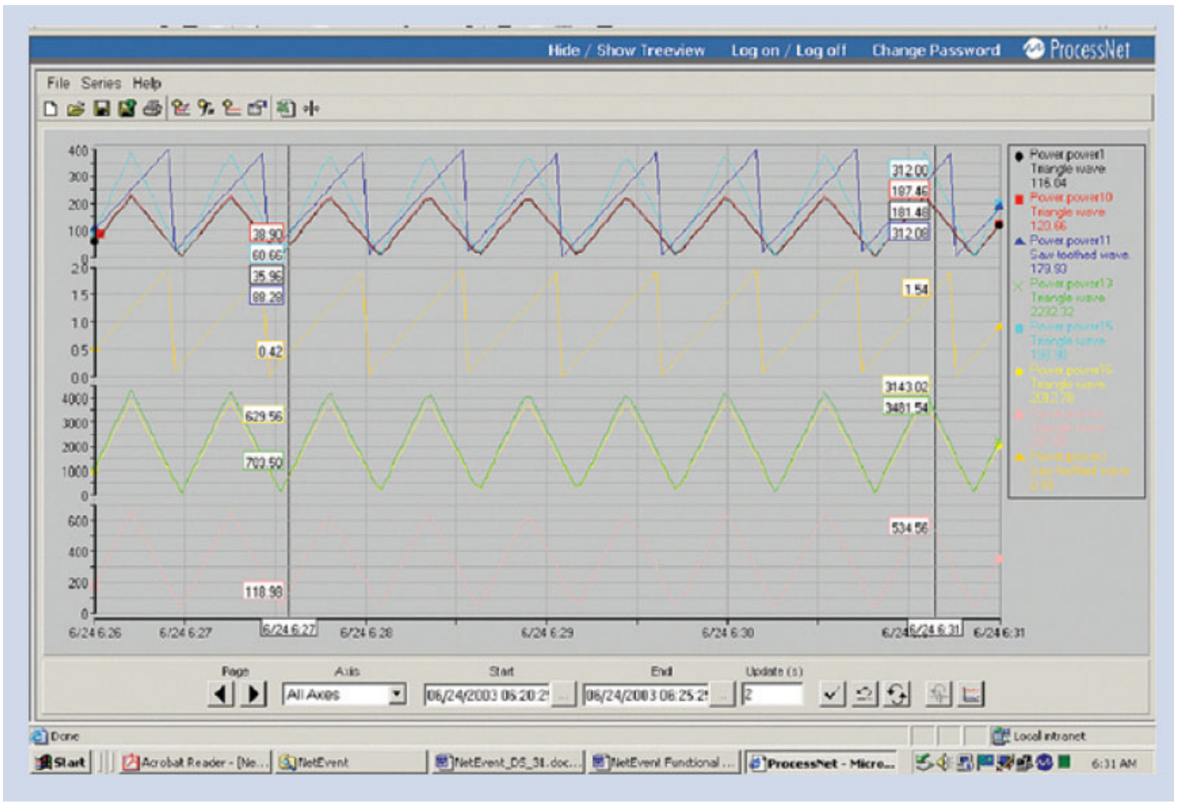


Figure 14.2: ProcessNet's NetTrend

Trend Link is a tool for viewing, analyzing and reporting trend information. It has built in Microsoft Excel export, as well as the ActiveX components can be viewed in an Internet browser. Figure 14.7 shows a screen shot of the Trend Link graph window.

The last tool, Trend Web, is a tool for accessing and view trend charts from the Internet. Figure 14.8 shows an example of an online ActiveX application. The plots are automatically updated when new tags are added to the database.

14.2.9 OPC Toolbox from MathWorks

The OPC Toolbox is an extension the MATLAB [13] technical computing environment with tools for interacting with OPC servers. It is possible to read, write, and log OPC data from devices that support the OPC Data Access standard. The toolbox enables manufacturing processes to retrieve plant data into MATLAB for analysis, visualization, simulation, and prototyping of algorithms. It enables MATLAB to read from and write to OPC servers using synchronous or asynchronous operations. A screen shot of this application is given in 14.9.

14.2.10 OPC Systems from Eldridge Engineering Inc

They call this an "all in one" OPC client. It includes packages for real-time trending, data logging, X axis/circular/3D plotting, data logging to OLE-DB, event driven data logging, data routing support for multiple OPC servers. [12]

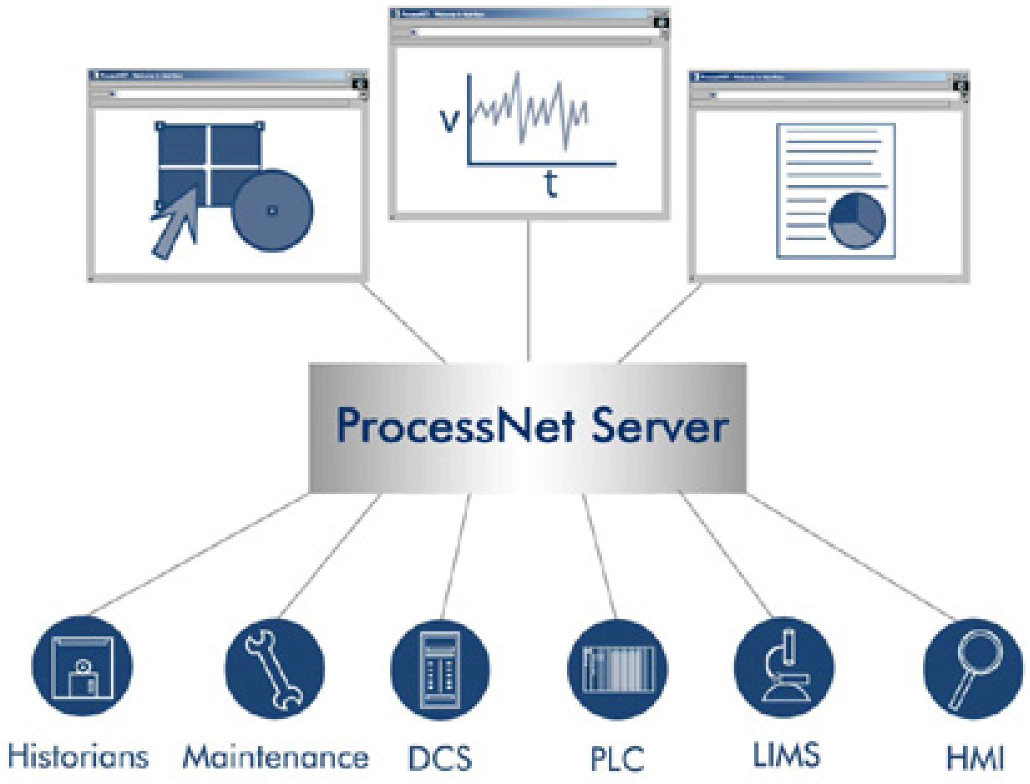


Figure 14.3: ProcessNet's System Architecture

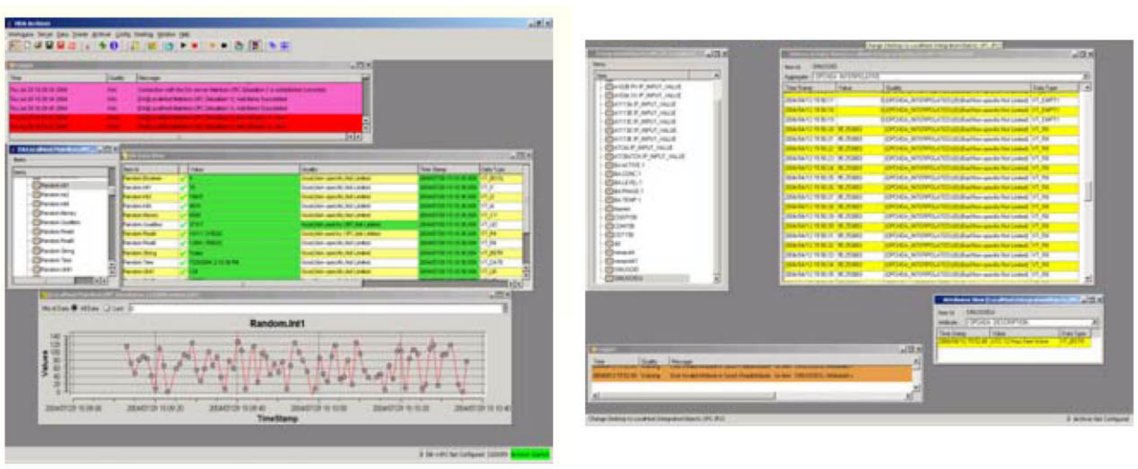
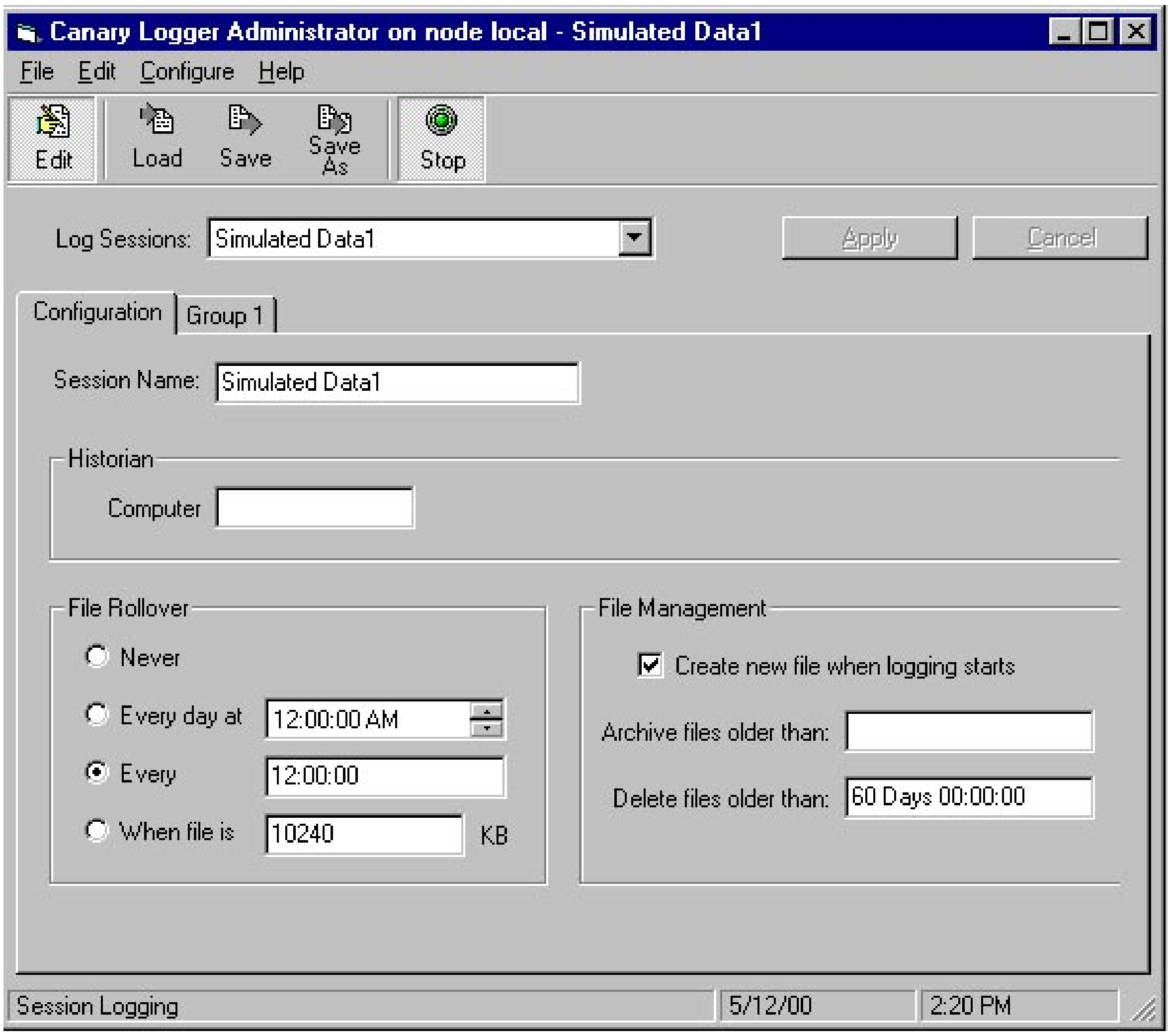


Figure 14.4: Integration Objects' Data Access Archiver


Canary Logger Administrator on node local - Simulated Data1

Log Sessions: Simulated Data1

Session Name: Simulated Data1

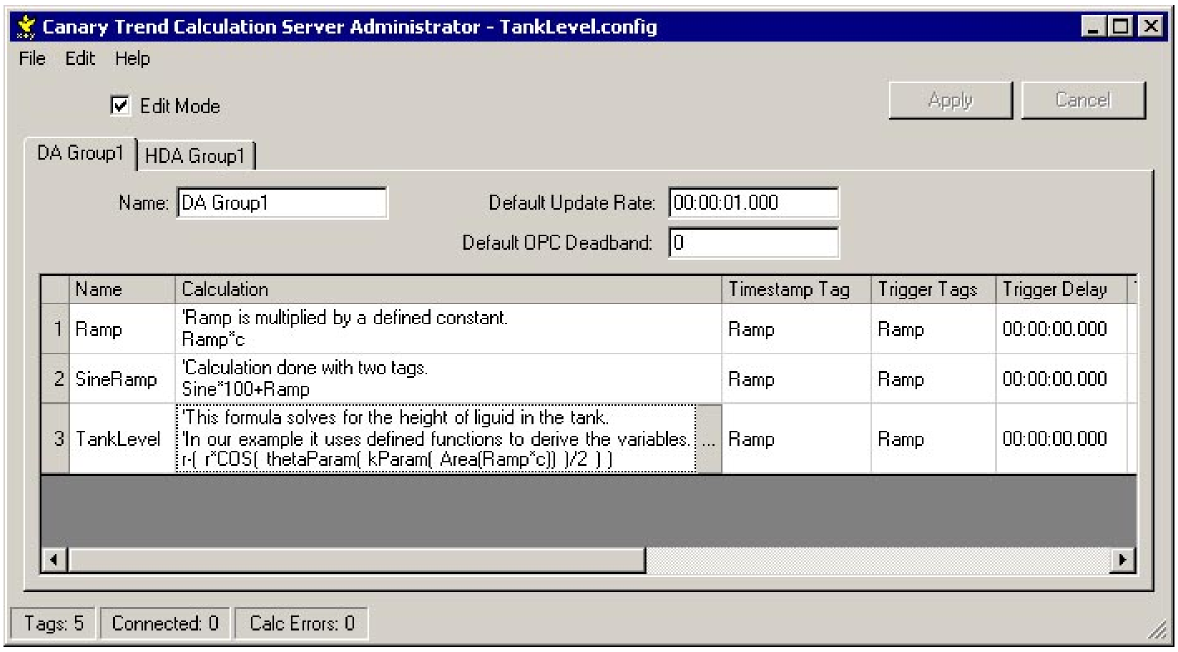
Historian
Computer:

File Rollover
- Never
- Every day at 12:00:00 AM
- [✓ Every] 12:00:00
- When file is 10240 KB

File Management
- [✓] Create new file when logging starts
- Archive files older than:
- Delete files older than: 60 Days 00:00:00

Session Logging | 5/12/00 | 2:20 PM

Figure 14.5: Canary Labs' Trend Historian screen shot


Canary Trend Calculation Server Administrator - TankLevel.config
File
Edit
Help
Apply
Cancel
DA Group1
HDA Group1

☑ Edit Mode

Name: DA Group1

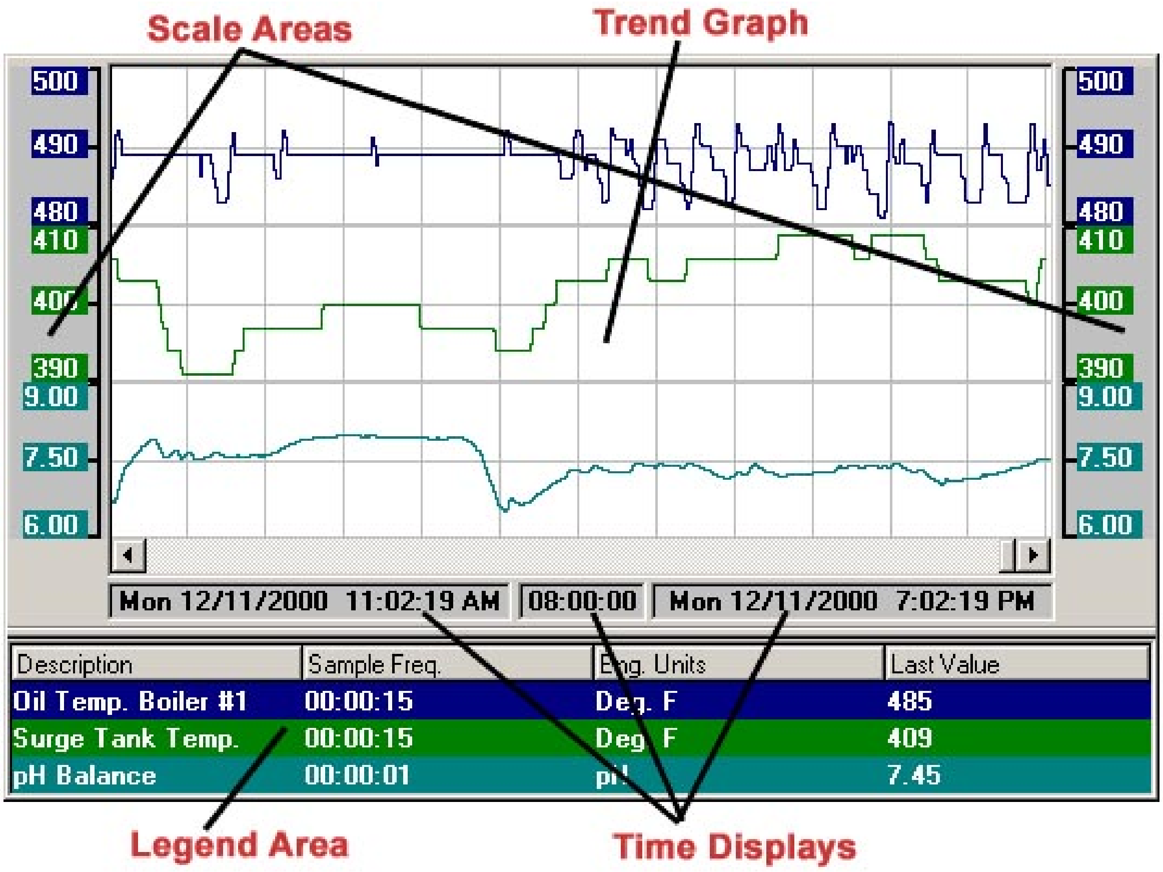
Default Update Rate: 00:00:01.000

Default OPC Deadband: 0

	Name	Calculation	Timestamp Tag	Trigger Tags	Trigger Delay
1	Ramp	'Ramp is multiplied by a defined constant. Ramp*c	Ramp	Ramp	00:00:00.000
2	SineRamp	'Calculation done with two tags. Sine*100+Ramp	Ramp	Ramp	00:00:00.000
3	TankLevel	'This formula solves for the height of liquid in the tank. 'In our example it uses defined functions to derive the variables. r-(r*COS(thetaParam(kParam(Area(Ramp*c)))/2))	Ramp	Ramp	00:00:00.000

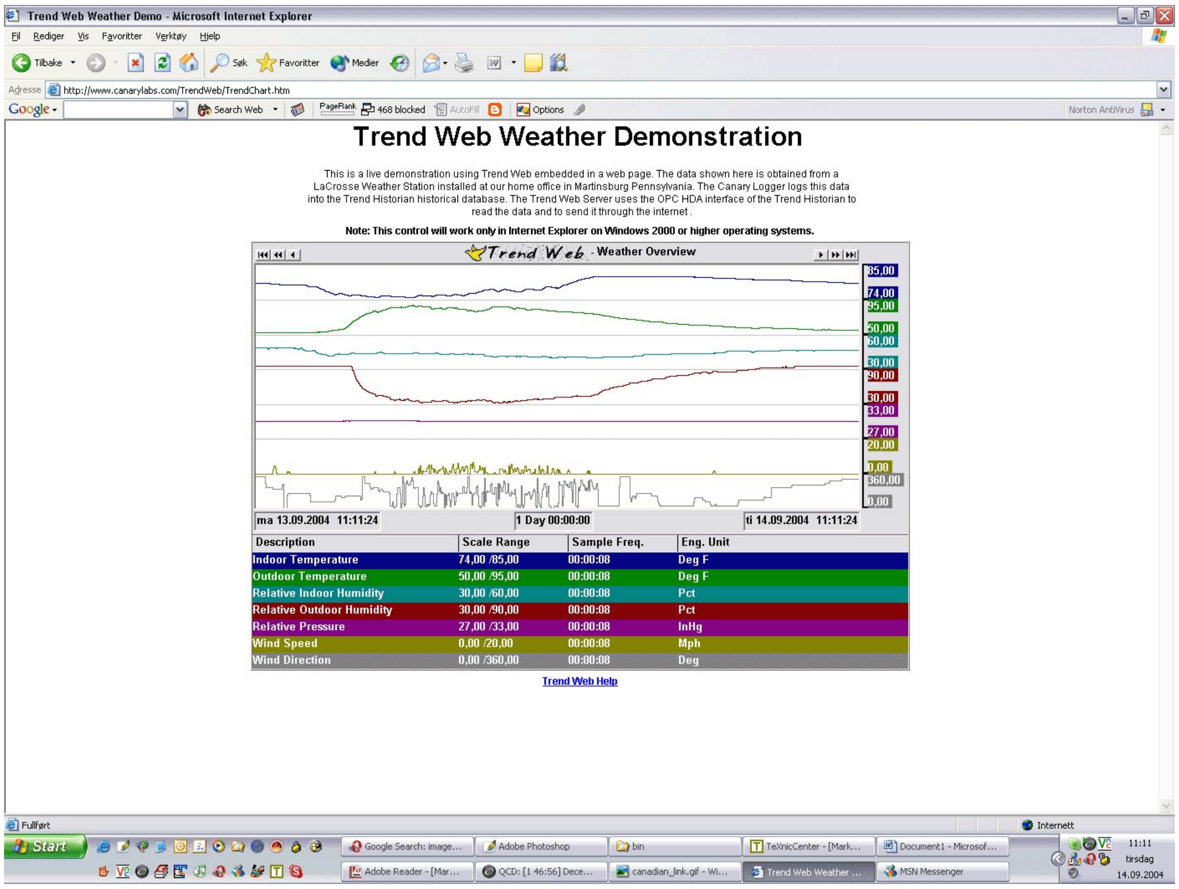
Tags: 5 | Connected: 0 | Calc Errors: 0

Figure 14.6: Canary Labs' Trend Calc screen shot



Description	Sample Freq.	Eng. Units	Last Value
Oil Temp. Boiler #1	00:00:15	Deg. F	485
Surge Tank Temp.	00:00:15	Deg. F	409
pH Balance	00:00:01	pH	7.45

Figure 14.7: Canary Labs' Trend Link screen shot



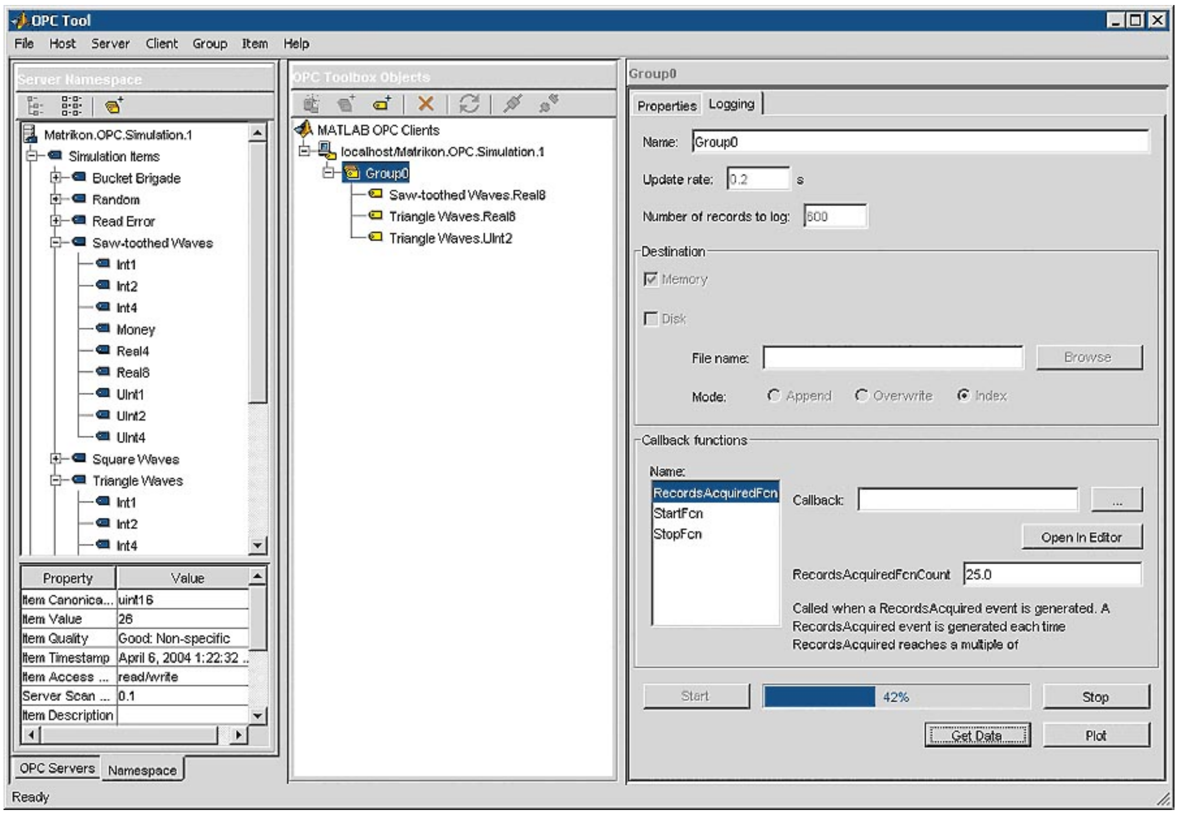
Trend Web Weather Demonstration

This is a live demonstration using Trend Web embedded in a web page. The data shown here is obtained from a LaCrosse Weather Station installed at our home office in Martinsburg Pennsylvania. The Canary Logger logs this data into the Trend Historian historical database. The Trend Web Server uses the OPC HDA interface of the Trend Historian to read the data and to send it through the internet .

Note: This control will work only in Internet Explorer on Windows 2000 or higher operating systems.

Description	Scale Range	Sample Freq.	Eng. Unit
Indoor Temperature	74,00 /85,00	00:00:08	Deg F
Outdoor Temperature	50,00 /95,00	00:00:08	Deg F
Relative Indoor Humidity	30,00 /60,00	00:00:08	Pct
Relative Outdoor Humidity	30,00 /90,00	00:00:08	Pct
Relative Pressure	27,00 /33,00	00:00:08	InHg
Wind Speed	0,00 /20,00	00:00:08	Mph
Wind Direction	0,00 /360,00	00:00:08	Deg

Figure 14.8: Canary Labs' Trend Web screen shot



Matrikon.OPC.Simulation.1
- Simulation Items
  - Bucket Brigade
  - Random
  - Read Error
  - Saw-toothed Waves
    - Int1
    - Int2
    - Int4
    - Money
    - Real4
    - Real8
    - UInt1
    - UInt2
    - UInt4
  - Square Waves
  - Triangle Waves
    - Int1
    - Int2
    - Int4

Property	Value
Item Canonica...	uint16
Item Value	26
Item Quality	Good: Non-specific
Item Timestamp	April 6, 2004 1:22:32 ..
Item Access ...	read/write
Server Scan ...	0.1
Item Description	

MATLAB OPC Clients
- localhost/Matrikon.OPC.Simulation.1
  - Group0
    - Saw-toothed Waves.Real8
    - Triangle Waves.Real8
    - Triangle Waves.UInt2

Name: Group0
Update rate: 0.2 s
Number of records to log: 600

Destination
[✓ Memory]
Disk
File name:
Mode: Append Overwrite [✓ Index]

Callback functions
Name:
RecordsAcquiredFcn
StartFcn
StopFcn
Callback:
RecordsAcquiredFcnCount 25.0
Called when a RecordsAcquired event is generated. A RecordsAcquired event is generated each time RecordsAcquired reaches a multiple of

42%

Figure 14.9: OPC Toolbox 1.1 from MathWorks screen shot

14.2.11 DAQBench from Agile Integration

DAQBench [11] is 32-bit ActiveX controls for measurement and automation. DAQBench programmers can use development tools such as Visual Basic and Delphi to build their own applications. Within these development tools, ActiveX controls deliver a property page interface for configuration; events and methods for programmatic control. DAQBench controls can be used within any ActiveX control container, such as Visual C++, Borland C++ Builder, Excel, Microsoft Internet Explorer etc. See 14.10 for some of the ActiveX components provided by DAQBench.

14.2.12 Software Toolbox

Software Toolbox is a company that provides several inexpensive OPC solutions, amongst other software packages for industry. Many of the products are single modules used for further development and customization at each customer. We will here mention two of these modules.

TrendWorX32 from Software Toolbox An OPC client implemented with ActiveX/OLE Object Technology. The client provides trend analysis, of both historical and real time data. Logging of OPC data to any ODBC source. Supports also data logging to MS Access using ADO technology.

The client is meant to function as an ActiveX control with Microsoft Visual Basic and Visual C++ applications. Is also compatible with other ICONICS OPC Clients such as GraphWorX32 and AlarmWorX 5.0. TrendWorX32 claims also to be compatible with other OPC applications delivered by other companies.

OPC Web Client from Software Toolbox The OPC Web Client can be used with IIS Web Server making one connection to an OPC Server or each Web Browser can make it's own connection to the

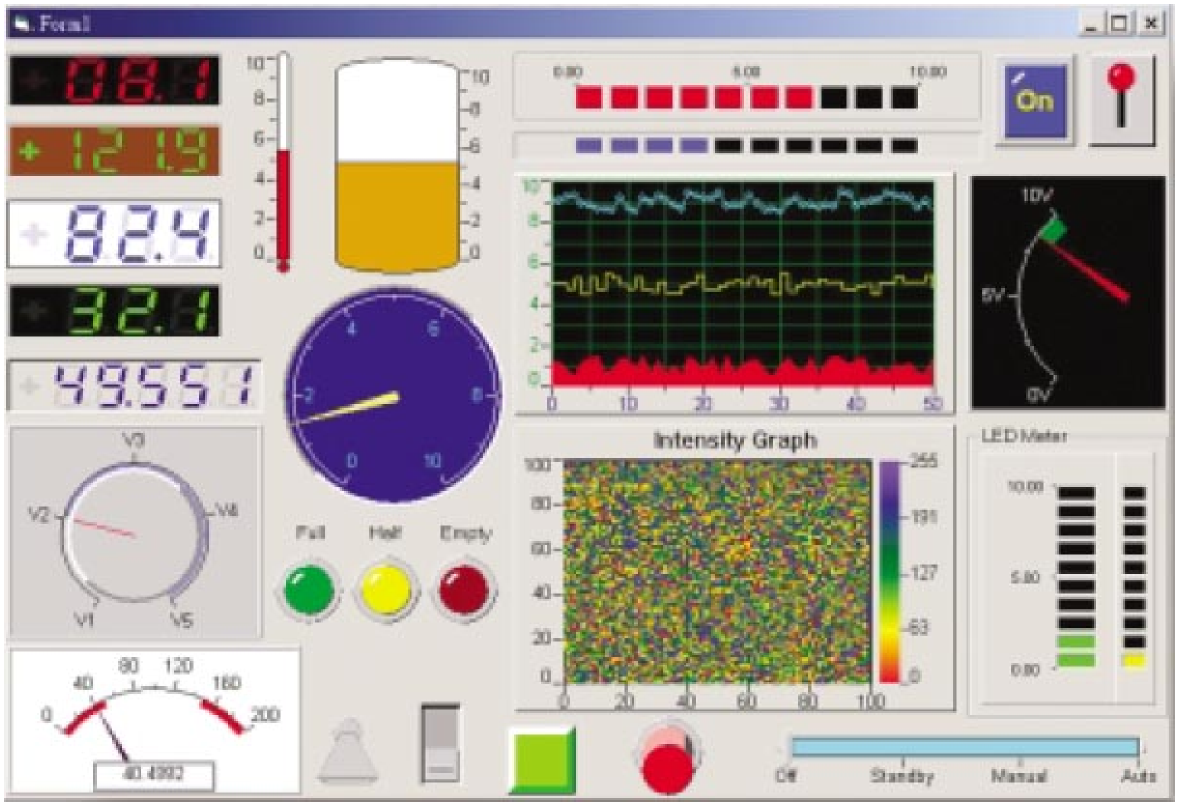


Figure 14.10: DAQBench from Agile Integration screen shot

OPC Server. The client component provides an interface which makes it possible to integrate it with VBScript, JScript, ASP,or from a client script in the browser.

14.2.13 PI Advanced Computing Engine from OsiSoft

[14]: "PI Advanced Computing Engine (ACE) allows programming of calculations (e.g., heat and material balances, data reconciliation, real-time cost accounting, batch summary, etc.), communication applications (e.g., alarming, emailing, paging, etc.), data transfer programs. It allows users to develop calculations in Visual Basic. ACE consists of three components: Scheduler, Manager, and Wizard."

This software also provides functionality for retrieving/sending data to tags or other systems, various scheduling scenes, allows clamping and bad value substitution of inputs and outputs, ability to call COM and .NET objects. It also has the ability to monitor performance of the individual equations, as well as exposing calculations as a Web service.

14.13 shows a screen shot from the Visual Studio .Net application construction environment, 14.14 shows a application used for tag value testing.

14.2.14 LabView from National Instruments

"NI LabVIEW is the graphical development environment for creating flexible and scalable test, measurement, and control applications rapidly and at minimal cost. With LabVIEW, engineers and scientists interface with real-world signals, analyze data for meaningful information, and share results and applications. Regardless of experience, LabVIEW makes development fast and easy for all users.

The LabVIEW Data logging and Supervisory Control (DSC) Module adds features and capabilities to LabVIEW to help create monitoring applications and data logging applications. The DSC Module

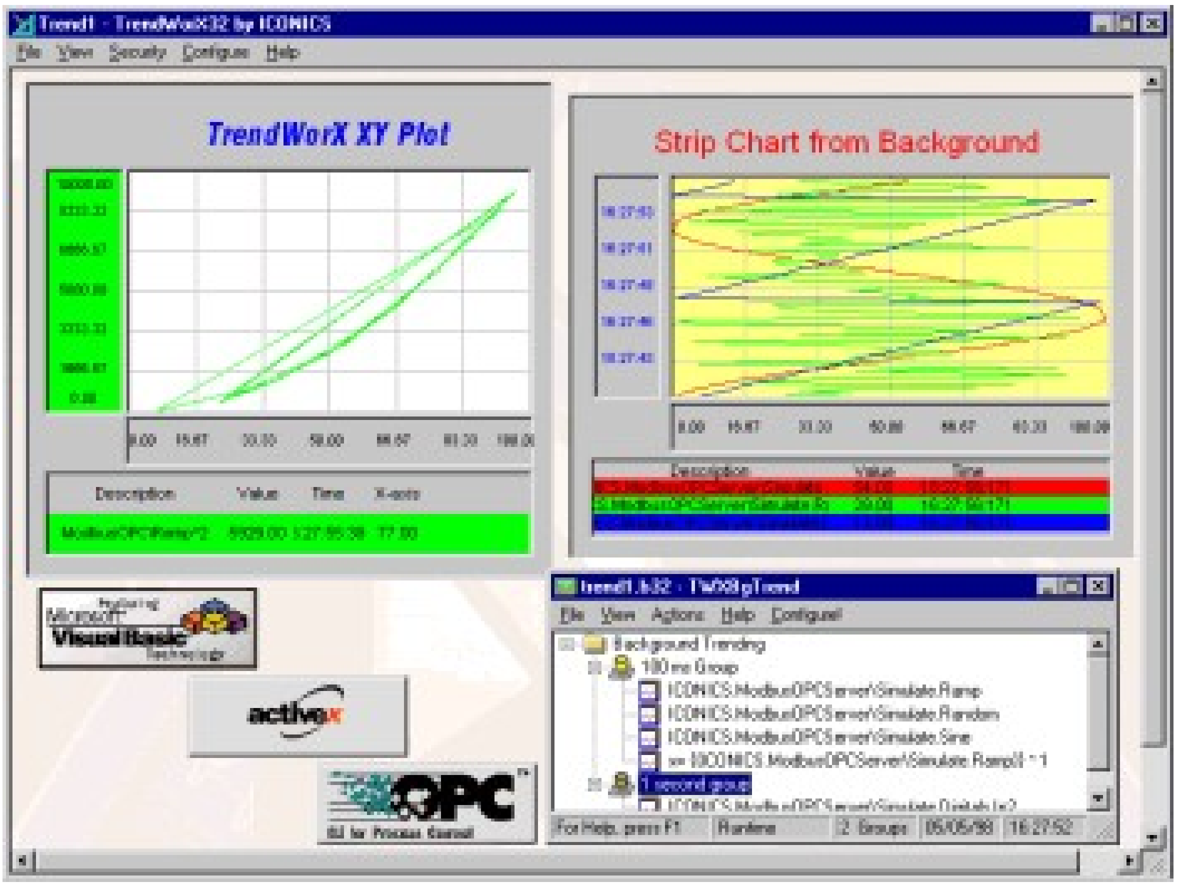

Trend1 - TrendWorX32 by ICONICS
File View Security Configure Help
TrendWorX XY Plot
Strip Chart from Background
Description Value Time X-axis
Description Value Time
trend1.t32 - TWX8gTrend
File View Actions Help Configure!
Background Trending
100 ms Group
ICONICS.ModbusOPCServer\Simulate.Ramp
ICONICS.ModbusOPCServer\Simulate.Random
ICONICS.ModbusOPCServer\Simulate.Sine
1 second group
For Help, press F1
Runtime
2 Groups
05/05/98
16:27:52
activeX
OPC
Microsoft Visual Basic Technology

Figure 14.11: TrendWorX32 screen shot from Software Toolbox

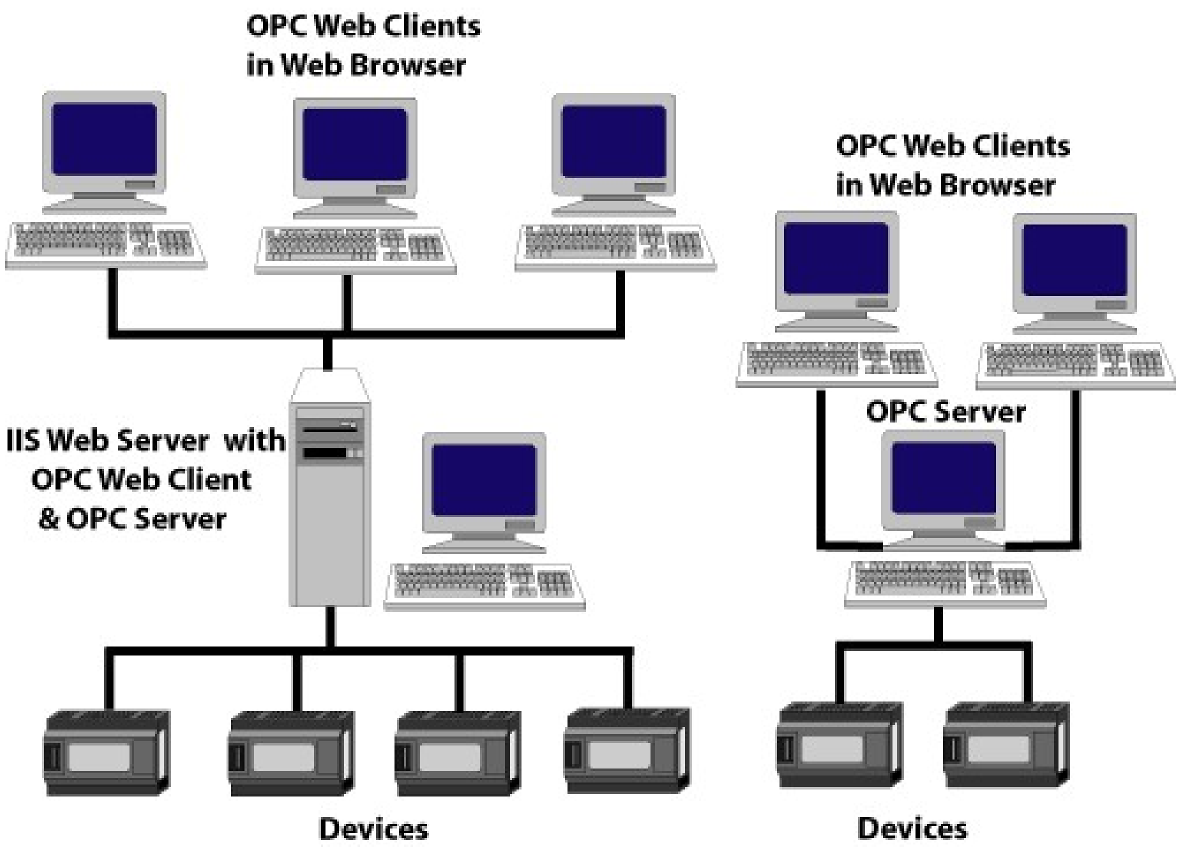
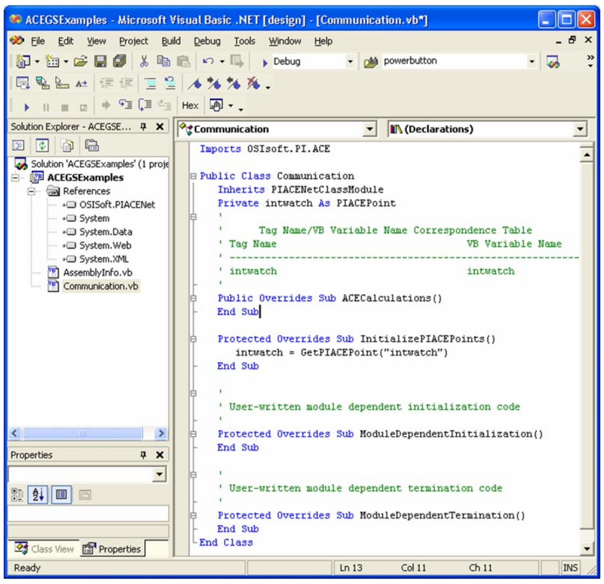
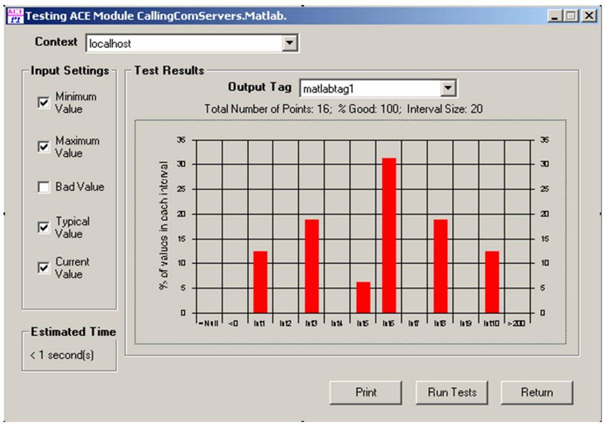

OPC Web Clients in Web Browser
OPC Web Clients in Web Browser
IIS Web Server with OPC Web Client & OPC Server
OPC Server
Devices
Devices

Figure 14.12: OPC Web Client from Software Toolbox screen shot



```
Imports OSIsoft.PI.ACE

Public Class Communication
   Inherits PIACENetClassModule
   Private intwatch As PIACEPoint
   '
   '      Tag Name/VB Variable Name Correspondence Table
   ' Tag Name                                   VB Variable Name
   ' ------------------------------------------------------------
   ' intwatch                                   intwatch
   '
   Public Overrides Sub ACECalculations()
   End Sub

   Protected Overrides Sub InitializePIACEPoints()
      intwatch = GetPIACEPoint("intwatch")
   End Sub

   '
   ' User-written module dependent initialization code
   '
   Protected Overrides Sub ModuleDependentInitialization()
   End Sub

   '
   ' User-written module dependent termination code
   '
   Protected Overrides Sub ModuleDependentTermination()
   End Sub
End Class
```

Figure 14.13: ACE / VS .NET application development environment screen shot


Testing ACE Module CallingComServers.Matlab.

Context: localhost

Input Settings
- [✓] Minimum Value
- [✓] Maximum Value
- [ ] Bad Value
- [✓] Typical Value
- [✓] Current Value

Test Results

Output Tag: matlabtag1

Total Number of Points: 16; % Good: 100; Interval Size: 20

Estimated Time: < 1 second(s)

Figure 14.14: ACE application example from OsiSoft

provides solutions for supervisory control of a wide variety of distributed systems using the flexibility of graphical LabVIEW programming.

National Instruments offers a number of LabVIEW development systems and programming tools to help you build professional applications. Choose from any of the development system or Developer Suite options on the left for system specifications, pricing, and ordering information to meet the demands of your instrumentation systems. " [18]

Screen shots of LabView from National Instruments is given in 14.2.14

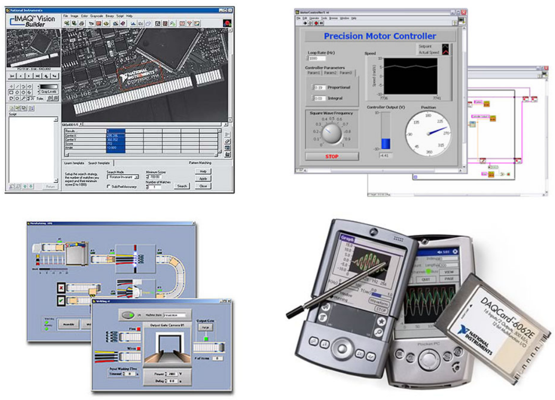


Figure 14.15: LabView from National Instruments screen shots

14.3 Summary of the Marked Research

These were some of the interesting solutions worth noticing in this section. We found several other tools more or less useful. Many of them where quite extensive, probably beyond the scope of our project. Please see Appendix F.1 for a thoroughly evaluation of the alternatives we found.

Chapter 15
Chapter summary

A typical network for gathering data from industrial sites produce large amounts of data from metering stations and gauges. The current system has no good way of logging, systemizing and exposing this data to non-operator personnel hoping to do advanced operations like trending. With the help of the customer we have identified the demands a solution to this problem would have and we have described how we think it should be solved.

All the existing solutions found during market research had to be scraped as none of them fulfilled all the requirements. However we should be able to get some inspiration from them, at least when it comes to designing a database for OPC data tags.

As none of the existing solutions were satisfactory, we have to develop our own. The technological pre study was meant as a startup of this and from the evaluations (see E.6) in it we have made the choice to base our solution on the Microsoft .NET framework and a Microsoft SQL Server 2000 database. Our job now is to crystallize the demands for, design, construct, implement and document our suggested solution.

15.1 Conclusion

After reviewing the already existing solutions, we have concluded that none of them fulfill all the requirements. The solutions that provide logging(OPC DA/HDA Archiver, OPC Toolbox for Matlab, Trend software from Canari Labs, DAQBench, Matrikon ProcessNet, and OPC Systems from Eldridge) of selected data tags to a database will however be studied in later phases of the project to understand how to best store OPC data efficiently in a relational database. If we want to fulfill all the criteria, we have to program the solution ourselves.

Appendix E
Technological pre study

E.1 Introduction

As OPC is an industry standard, development of software which utilize it can be done with a wide array of different tools and development environments. Virtually all programming languages used in large scale software development has the possibility of accessing OPC data. The packages used to connect the language/environment with OPC have traditionally been highly proprietary, expensive and non-free. The customer has expressed high emphasis on developing a cheap solution. The definition of cheap entails either free software or use of Microsoft products, which the customer has standardized on. Two big competitors crystallize themselves:

1. Microsoft .NET
2. Sun JAVA

E.2 Short introduction to OPC

E.2.1 The need for standardization

OPC (formerly OLE for Process Control) is an open standard bridging Windows based applications and process control hardware. In automation systems process control data come from various devices measuring temperature, pressure, movement of liquids, etc. The systems might read as well as write values to these controls and today there are a large number of vendors and field devices available on the market. Manufactures have many different data sources, e. g. PLCs, DCSs, databases, gages and RTUs. Additionally, field devices might be available through different connections like Ethernet, serial, radio and so on. Selected data is processed in applications which implement functionality like visualization and analysis or simply controlling (read/write) the process control system. Further complexity is added when you take into account the different platforms these applications might run on (Windows, Dos, Linux). Certainly, one inevitable obstacle has to be passed; how to arrange easy and painless communication between applications and different data sources?

In the pre-OPC days every manufacturer (like ABB) provided proprietary interfaces supporting their hardware. Consequently, the software was developed with only a certain set of field controllers in mind and thus only supporting these devices. System properties like interoperability and scalability were basically non-existent. In practice (and often even in theory), once a customer relationship was established, there was no choice but choosing hardware from this vendor. If expansion or replacement was needed, choosing a device from another supplier would most likely require a complete or partly rewrite of existing software. Also, such architecture could potentially put an unreasonable load on the system. Imagine a control system with three control devices and three operator stations. Each of the three operator stations needs to gather data from each control device. Worst case scenario is sketched in figure E.1. Here each of the applications running on the operation stations requires data from all three field devices, a total of nine different connections are required. It goes without saying that such systems have a serious scalability problem and that the solution to these obstacles is standardization.

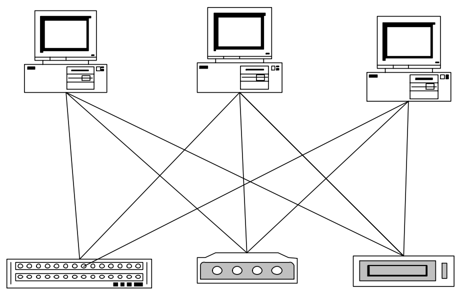


Figure E.1: Communicating using proprietary interfaces

E.2.2 Tags/points

The values from the data sources in control systems are usually referred to as tags or points. These are single valued data items. Each tag consists of three fields, namely a value, a quality and a timestamp. Please note that even though the OPC Servers are obliged to include timestamps in the tags, not all OPC compatible hardware timestamp their tags.

E.2.3 OPC Data Access

The OPC Foundation develops and maintains the specifications that constitute OPC. The first set of specifications was released in 1996 and the main goal was to ensure interoperability. Today, OPC consists of nine specifications. The original one is called OPC Data Access (OPC DA). OPC DA standardizes real-time data access in automation systems and is based on a client-server architecture. As figure 1.2 shows, the data in the control devices are available through an OPC server. Thus, regardless of hardware implementation, client applications only see an OPC interface. Adding a new device that supports OPC should therefore be without pain. Also, the system scales very well, since this expansion would only require one additional connection, see figure E.2.

OPC DA has four different communication mechanisms; synchronous and asynchronous calls, refresh and subscription. Synchronous and asynchronous calls operate by polling and require the client to specify what data the server should read or write. When a synchronous call is issued, the server does not release the caller until the requested data is fetched. Likewise, when an asynchronous call is issued, the server releases the client immediately and later returns the data in an appropriate callback path.

Clients may also subscribe to a set of values. The server then notifies the client when significant changes occur. Thus, subscription is event based. Refresh is another mechanism which requires the client to specify a set of variables for reading. When the client issues the refresh call, the server responds asynchronously by sending the data through the predefined callback path.

There are some essential parameters which define the communication between the server and the client. As mentioned above, when using the event mechanism (subscribing to data), only significant changes of the values are reported. The level of significance is set by the deadband parameter. This value is a percentage and set to 0 the server will notify the clients on any change. Also worth mention-

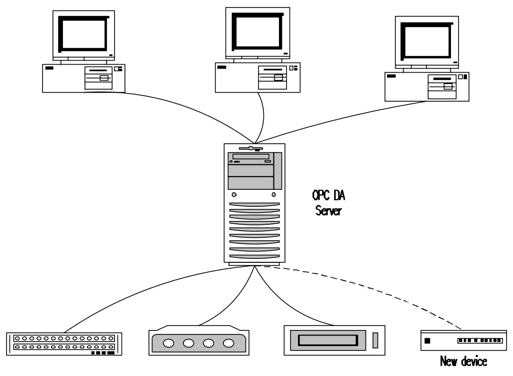


Figure E.2: Communicating using OPC Server

ing is the Update Rate parameter. This is also specified by the client and sets the rate at which subscribed parameters are updated. Regardless of significant changes, the callback method is not called more frequent than the update rate. Setting this parameter to 0 will cause the server to notify clients on any significant change.

E.2.4 OPC Historical Data Access

OPC DA standardizes real-time access to data in automation systems. In contrast, and as the name suggests, OPC Historical Data Access standardizes exchange of stored data. Such data would be applied in a wide range of analysis, like performance assessment, trend plotting, fault prediction and so on.

E.3 How the different technologies are evaluated

The different technologies are compared by evaluating them against the criteria listed in table E.1. The basis for these criteria is communication with the customer and earlier experience with software development. Each criterion is also assigned an importance priority: low, medium or high and a numerical representation of the priority, respectively 1.0, 1.5 and 2.0. This measures how important each criterion is compared to each other.

The numerical value of the priority assigned to each criterion will be multiplied with the evaluation score, which range from 1 to 5, that each of the different technologies gets on the different criteria. For example a technology that gets an evaluation score of 4 on "Technological suitability" will get a total score for this criterion of 4(evaluation score) * 2.0(Numerical value for high priority) of 8. The score for all the criteria of a given technology is added so we get a sum that gives us a rough estimation on how good a solution it is.

Criterion	**Description**	**Reason**	**Importance**
Technological suitability	The technology should be able to fulfill all the requirements of the product.	Important to fulfill requirements in order to get a good product.	High
Knowledge of technology	The members of the group should know and to a certain degree master the technology.	Learning new technology is time consuming. This project has a very limited time horizon, which should be spent on applying, not learning new technology.	Medium
Workload / Complexity of technology	The degree of complexity in utilizing the technology.	If we use more time than allocated, we will not finish in time.	High
Monetary cost	We want cheap/affordable technology.	Extremely limited monetary budget for this project.	Medium
Portability	The technology should not be tied to a particular software or hardware platform.	Ideally, we should be able to run the product on any operating system/hardware combo available.	Low
Integration	The technologies ability to integrate with customers other products.	The customer want to standardize on a set of core technologies for all it's products.	Medium
Support/help	The possibility of getting support and/or help in case of problems using the technology.	We need to be able to get help if the technology does not work as advertised or bad bugs are found.	Medium

Table E.1: Technological evaluation criteria

E.4 Development platforms to consider

E.4.1 Sun JAVA

Introduction

Java is Sun Microsystem's version of a modern software application development platform. Software development speed has always been held back by differences in the multitude of machines it should be able to run on. Code compiled for one architecture can rarely run on another machine with a different architecture. In reality this means all code must be compiled on each and every architecture. As networking, and especially internet, has become more and more popular, this has become a huge problem. The need to run code on any machine, independent of the architecture of that machine, has manifested itself as the traditional information channels on the internet (web, email and news) have proven incapable of providing the required degree of interactivity today's users demand. Sun spotted this early and noticed the need for a completely platform/architecture independent code standard that could be run on any computer. Their answer is what we call Java today. Java is very closely related to the C family of languages. The syntax is well known for everybody who has written C or C++. The new and exciting thing about Java is that it compiles source code not to machine code for one specific type of computer, but to a intermediate layer of what Sun calls "Java byte code". This code can run on any computer for which there is a Java Virtual Machine (JVM). Sun has made JVM available for free for multiple computer architectures (Windows, Linux, Solaris etc). The JVM translates the byte code to actual machine code at run time. By making JVM available for different platforms you can actually "compile once, run everywhere".

People quickly understood the great benefits of this and today JVM and Java Software Development Kit (J2SE) are downloaded from Sun several thousand times a day. Developers utilizing Java technology range in the hundreds of thousands. Everything from simple code segments on web pages to the most complex software systems in banking and financial businesses are now developed with

Java.

Today Sun can offer the following with Java: - An enormous standard library with already implemented functionality - Computer architecture independent code execution

Technological suitability

Java has become a defacto standard in most areas of application development. The standard library that comes with Java contains a vast collection of essential functionality. Clear and well structured interfaces and object orientation gives us a professional, productive and most importantly well tested development platform.

There are two different ways of connecting Java and OPC:

1. General DCOM bridges.
2. OPC wrappers

General DCOM bridges are software products which enables the use of all DCOM-objects in Java. DCOM is actually a Microsoft technology developed by Microsoft for use in their Windows operating systems. Quite a lot of work has to be done to enable Java, developed as an cross platform language, to use DCOM. A General DCOM bridge does this for us. The downside of general DCOM bridges is the fact that they are general. If we chose to use one for this project we have to spend time not only to learn OPC but also on how DCOM operates. The time available for this project is limited, so this is not something we want to do.

The following DCOM bridges for Java are available today. Functionality is more or less completely identical.

Intrinsyc J-Integra

- Link: http://www.intrinsyc.com/
- Cost: Free time limited version available. License for full use can be bought.

Interface Tool for Java/ Bridge2Java

- Link: http://www.alphaworks.ibm.com/tech/bridge2java
- Cost: License for full use must be bought.

JACOB by Dan Adler

- Link: http://danadler.com/jacob/
- Cost: open source

jacoZoom by infoZoom

- Link:http://www.infozoom.de/ie/jacozoom.html
- Cost: Free time limited version available. License for full use can be bought.

Java2COM by Neva Object

- Link:http://www.nevaobject.com/docs/java2com/java2com.htm
- Cost: Free time limited version available. License for full use can be bought.

Jawin

- Link: http://sourceforge.net/projects/jawinproject
- Cost: open source

R-JAX by Stryon

- Link:http://www.stryon.com/products.asp?s=2
- Cost: Free time limited version available. License for full use can be bought.

OPC wrappers are actually DCOM bridges developed especially for OPC DCOM objects. This hides technological details from developers not interested in other DCOM objects and gives them access to OPC through Java API interfaces. This is great for the developers, and us, because they can use spend their time utilizing OPC, not trying to get it to work in the first place. It comes at a price though, none of the OPC wrappers are available free of charge.

The following OPC wrappers are available. Descriptions are taken from http://www.opcconnect.com.

ErgoTech OPC Gateway

- Link: http://www.ergotech.com/
- Description:"ErgoTech is a producer of Java components for manufacturing, and the OPC Gateway is designed to work with that company's Virtual Instrumentation Beans (VIB) product. VIB is a collection of visualization and data server components designed to be used with real-time devices. "
- Cost: License for full use can be bought.

Odense Production Information (OPI) JOPCClient

- Link: http://www.opi.dk/1.6.software.htm
- Description:" Odense Production Information (OPI) of Denmark takes a different approach with its Java OPC client API. JOPCClient is restricted to the Windows platform only, and uses Sun's Java Native Interface (JNI). This technique allows Java developers to access the functionality of OPC using only Java calls and without needing knowledge of COM and DCOM. JNI is supported by most Java Virtual Machines (excluding the Microsoft JVM), and JOPCClient is compatible with JDK 1.1, 1.2 and 1.3. "
- Cost: Free version with limited functionality available. License for full use can be bought.

NetModule's JPC OPC-Driver

- Link: http://www.netmodule.com/e/produkte/opc.asp
- Description:"NetModule's JPC OPC-Driver allows OPC client programs to be coded in Java. Java classes are provided which mirror the server objects defined by the Data Access 2 specification. Java Native Interface (JNI) is used, along with an 'OPC-BridgeDLL' coded in C, to interface to the actual OPC server. NetModule claims compatibility with OPC servers from Siemens, Rockwell, B&R and K&W. "
- Cost: License for full use can be bought.

Wapice OPC2Any

- Link: http://www.wapice.com/wapice/opc2any.php?lang=en
- Description: "With OPC2Any, Finnish company Wapice offers connectivity to OPC Data Access servers via a Java API."
- Cost: License for full use can be bought.

DCOM is not a Java friendly technology, problems related to Java, DCOM and OPC are likely to surface.
Score: 4

Knowledge of technology

Everybody on the team knows Java; we have solid backgrounds in development with this technology. OPC in Java is unknown for everybody. Score: 4

Workload/Complexity of technology

Low compared to other choices. Java is well know, OPC is not.
Score: 3

Monetary cost

Java is free of charge from Sun. However using OPC in Java can be cheap or expensive depending on how much work you are prepared to do. General DCOM bridges are not very expensive, but a substantial amount of time will need to be invested in creating an OPC-API(OPC wrapper) in Java from such a bridge. Several of these wrappers are available, but they are expensive.
Score: 4

Portability

Java was developed with portability from day one and can actually be used to "compile once, run everywhere".
Score: 4

Integration

The customer has chosen to standardize on Microsoft products. Java is developed by Sun, one of Microsoft's biggest competitors. This is far from optimal; no support from Microsoft on integrating anything written in Java with their products can be expected.
Score: 3

Support/Help

Java is used by an enormous amount of developers world wide. Help is readily available online. Sun maintain a large database with Java related tips and tutorials. They also maintain Java Community, a forum where developers from everywhere can ask and answer questions. It is important to note that this kind of support is unofficial; you are dependant on other people to help you. However, you very seldom find unique problems. Somebody has usually experienced the problem and has already found a solution. You just have to find that somebody.
Score: 3

E.4.2 Microsoft .NET

Introduction

.NET is Microsoft Corporations version of a modern software application development platform. .Net is Microsoft's reply to Java. A technology that can run code independent of machine architecture and operations systems is a big threat to them. At first they tried to work with Sun, but Sun felt Microsoft were trying to make Windows specific changes to the language. Sun refused, and Microsoft responded by creating .NET and refusing to have anything more to do with Java.

Technological suitability

.NET is quickly becoming a strong opponent to Java. Microsoft has invested huge amounts of money on building a development framework which pretty much offers the same kind of functionality Java did and continues to do. A big standard library, object orientation and a comprehensive development environment, Visual Studio 2003, makes it hard for developers not to give .NET a serious try. Communication with OPC can be done through multiple software packages: Descriptions are taken from http://www.opcconnect.com

.NET API from OPC Foundation

- Link: http://www.opcconnect.com/dotnet.php
- Description: "the OPC .NET API .. support DA 2 and 3, DX and HDA. The .NET API provides a unified set of interfaces for accessing both COM and SOAP/XML servers, and also includes C# and VB.NET clients which exploit these interfaces."

- Cost: Free of charge for members of OPC Foundation, http://www.opcfoundation.org/. Our customer, and thus we, are members.

Industrial DOT NET "Local IO"

- Link: http://industrialdotnet.com/
- Description: "IDN Local IO is a .NET assembly which allows rapid interfacing of .NET applications with industrial devices. Support is provided for OPC DA 2.0, with additional protocols to be included in future versions (MODBUS TCP is currently planned)."
- Cost: Free time limited version available. License for full use can be bought.

KineticaRT OPC Client Library

- Link: http://www.kineticart.co.uk/KineticaRTOPC.asp
- Description: " KineticaRT's .NET OPC Client .. supports OPC DA connectivity using an architecture based on connectable .NET components."
- Cost: License for full use can be bought.

Metadynamics OPC.ClientX.NET

- Link: http://www.metadynamics.com/opcclientxnet_brochure_p1.htm
- Description: " OPC.ClientX.NET was the first commercial product to offer a native .NET component for Data Access client development. OPC.ClientX.NET presents an object model based closely on the existing Data Access Automation specification. Connection to both DA 1.0 and 2.0 servers is supported, and the package helpfully includes sample code for a complete application."
- Cost: Free version with limited functionality available. License for full use can be bought.

Northern Dynamic SLIK-DAC

- Link: http://www.nordyn.com/Main/Products/SLIKDAC/SLIKDAC_Overview.htm
- Description:" Northern Dynamic's SLIK-DAC provides a Windows Forms Control to enable rapid development of DA client applications."
- Cost: Free time limited version with limited functionality available. License for full use can be bought.

Softing OPC Toolbox .NET

- Link: http://www.softing.com/en/communications/products/opc/tools/dotnet.htm
- Description:" Softing's .NET Client Toolkit supports both Data Access 2.0x and 3.0, and is compatible with all .NET programming languages. Comprehensive documentation integrates with Visual Studio .NET (2002 or 2003), and commented examples are available for both VB.NET and C#."
- Cost: Free version with limited functionality available. License for full use can be bought.

Technosoftware OPCDA.NET, OPCHDA.NET, OPCAE.NET

- Link: http://www.tswinc.us/pc-1-3-opcdanet.aspx
- Description: " Technosoftware's OPC Data Access .NET Wrapper (OPCDA.NET) provides .NET wrappers for the OPC Data Access and Common custom interfaces. Comprehensive documentation is included, as well as source for client programs (C# and VB.NET), and wizards for simplified application creation. OPCDA.NET 3.0 adds support for Data Access 3.0, and includes a DA 3.0 test client as well as a simulation server supporting both DA 2.0x and 3.0."
- Cost: Free time limited version available. License for full use can be bought.

Technosoftware XMLDA.NET

- Link: http://www.tswinc.us/pc-2-3-xmldanet.aspx
- Description:" Technosoftware's XMLDA.NET is .. a .NET wrapper which provides an XML-DA application interface. Using this component, applications developed as XML-DA client programs are able to access (COM) DA servers. Unlike an XML-DA gateway, this is achieved without XML serialization. XMLDA.NET is particularly recommended for ASP client applications which need to access DA servers."
- Cost: Free time limited version available. License for full use can be bought.

Visavi Visual OPC .NET

- Link: http://www.visavisoftware.com/visual_opc_net.html
- Description:" Visual OPC .NET is designed to enable rapid development of OPC DA clients using any .NET language. Classes are included which wrap the OPC COM interfaces, as well as Windows Forms controls which support direct binding to OPC data items."
- Cost: Free time limited version available. License for full use can be bought.

OPC in .NET should not entail any large challenges.
Score: 5

Knowledge of technology

Only one person on the team has used .NET for software development.
Score: 2

Workload/Complexity of technology

This can be quite complex. Both .NET and OPC is unknown territory for most of the group. The similarities between .NET and Java will help to reduce this potential problem.
Score: 3

Monetary cost

The .NET-framework is freely downloadable from Microsoft. Visual Studio is free for students of IME faculty on NTNU through Microsoft's MSDN Academic Alliance. A .NET solution means the customer needs to run some kind of Windows operating system from Microsoft. This is not free, but the customer already decided on this when they chose to standardize on Microsoft products.
Score: 3

Portability

Very poor. Microsoft maintains that anybody is free to develop .NET-framework for other operating systems. In reality, this is not happening. Mono is a Linux based .NET-framework but it is still nowhere near complete.
Score: 2

Integration

The customer has standardized on Microsoft. A .NET solution will comprise of more or less of only Microsoft products. Integration with the rest of the customer ' s software and hardware should be easy.
Score: 5

Criterion	**Importance**	**Sun JAVA / Evaluation score**	**Sun JAVA / Criterion score**	**Microsoft .NET / Evaluation score**	**Microsoft .NET / Criterion score**
Technological suitability	H/2.0	4	8	5	10
Knowledge of technology	M/1.5	4	6	2	3
Workload / Complexity of technology	H/2.0	3	6	3	6
Monetary cost	M/1.5	4	6	3	4.5
Portability	L/1.0	4	4	2	2
Integration	M/1.5	2	3	5	7.5
Support/help	M/1.5	2	3	5	7.5
Sum		23	**36.0**	25	**40.5**

Table E.2: Development platform criteria evaluation

Support/Help

You can buy support for Windows, Visual Studio and the .NET framework from Microsoft. Not an option for this project, but the possibility is definitely there. There is also an enormous amount of .NET material available online (web pages with free code, help and forums)
Score: 5

E.4.3 Development platform summary

The most important demand on the development platform in our project is the ability to communicate with the OPC servers that we are supposed to read and write data to/from. While both platforms make this possible, one has to consider the fact that OPC is build on DCOM technology. This is highly Microsoft-specific technology. Microsoft is of course the most avid supporter of their own technology and this puts Sun JAVA at a clear disadvantage for OPC related work. OPC in Java is a Microsoft Windows-specific hack to Java's platform independent architecture, while OPC in .NET is OPC in its natural environment. Not only does .NET get the best overall score (table E.2), but the customer's standardization on Microsoft products strongly supports .NET as the best development platform.

E.5 Database systems to consider

E.5.1 MySQL

Introduction

MySQL AB, most commonly know as only MySQL, is today the most widely used open source database in the industry. This popularity is based on its easy of use and non complicated design compared to other proprietary databases available. MySQL developers, MySQL AB [1], has sacrificed what other database vendors consider important enterprise grade functionality to be able to deliver a basic block of functionality at lightning speeds. It does not support the following:

- stored procedures
- referential integrity

- triggers
- sub queries

It does however support the more basic operations of adding, modifying and deleting data very efficiently. This is more than enough for the typical MySQL user, who normally use the database for web related storage. Examples of use are guest books, counters or online forums on web pages. These are typically non-critical uses, and it is in these areas MySQL has captured most of its users. However the same users are constantly utilizing MySQL in other areas and this has prompted MySQL AB to start implementing extra functionality normally only found in proprietary databases like MS SQL and Oracle. The open source nature of MySQL prevents users from lock-in to a single company or database platform; it also ensures the possibility of running MySQL on all the popular hardware platforms available. Currently more than twenty are supported, including every major Linux distribution, Mac OS X, UNIX and Microsoft Windows.

Technological suitability

Our needs for a database are basically the ability to read, write and alter data. MySQL is more than efficient for this. It does not support more advanced functionality which would be nice, but not critically so, to have; like stored procedures and referential integrity.
Score: 4

Knowledge of technology

The team has extensive experience with MySQL usage, in both school and non-school related work. This is regarded as well known technology.
Score: 5

Workload/Complexity of technology

MySQL is non-complex and user friendly.
Score: 4

Monetary cost

MySQL is available as open source, for free. It can be licensed under other conditions, but this is not an option for us.
Score: 5

Portability

MySQL is open source; it can be freely ported to any platform. It is extremely flexible on this point.
Score: 5

Integration

MySQL is not a Microsoft product, but this should not be a problem. MySQL AB has made tools for the use and integration of MySQL on Microsoft Windows available for free with the database.
Score: 3

Support/help

MySQL under open source license does not have any official support available. Its enormous popularity means that there is huge amount of documentation, tips and help available online however. You can buy official support for MySQL from MySQL AB.
Score: 3

E.5.2 Oracle Database

Introduction

Oracle Corporation's [8] databases are currently the market leader in high efficiency, high demand, and high complexity databases. According to Oracle they support pretty much all possible features you can demand from a modern database, including encryption, clustering, high availability, data compression, analytical tools and highly grained security control mechanism.

Oracle Corporation was founded 27 years ago in California. Larry Ellison, now Chairman and CEO, was the first to commercialize a new kind of technology called a relational database. He and his co-founders, Bob Miner and Ed Oates, saw a huge business potential. Today Oracle supplies technology to virtually all kinds of industry around the globe, they can boast that 98 of the Fortune 100 companies utilize some of their products in highly critical operations. Its product line no longer comprises of only database technology; they know also deliver business applications, perform application development and produce a set of decision support tools. Oracle is number one in software for information management and the currently the second largest software company in the world. The 44000 employees of Oracle Corporation will help their employer to "innovate and to lead the industry - while always making sure that we're focused on solving the problems of the customers who rely on our software."

Technological suitability

Oracle is among the top, if not the number one, databases available for use today. Not only is it highly efficient at the basic database functionality of reading, writing and altering data, but it constantly redefines the notion of what a database solution should encompass. Functionality like

- stored procedures
- referential integrity
- triggers
- sub queries

is not only supported but in most cases it was pioneered and developed by Oracle Corporation in the first place. This project will not have any demands for a database that aren't met by an Oracle database.
Score: 5

Knowledge of technology

The group has limited experience with Oracle databases. It has been utilized in school projects earlier, but in a limited way. A substantial effort, ergo use of time, will be required to learn this technology before use in this project.
Score: 3

Workload/Complexity of technology

Oracle solutions are designed to meet any demand a customer has for a modern database. This necessarily leads to a complex, but powerful product. While we have no doubt that Oracle will work as advertised, we do think its complexity is not needed for this task.
Score: 3

Monetary cost

Oracle Corporation enables developers to download most of it products, including its databases for free. They allow you to use full versions of the products only while developing and prototyping applications. You have to buy a non-development license when you ship your product. This is not cheap and it makes it unusable for us.
Score: 2

Portability

Oracle databases are available for multiple versions of Microsoft Windows, Linux, Solaris, HP-UX, Mac OS X and IBM z/OS. It is not an open source solution however, so if you need to use it on any other platforms you have a problem.
Score: 4

Integration

Oracle delivers tools and guides for integration of their products with other applications. Our customer has chosen to standardize on another platform however, this pretty much guarantees some kind of problems.
Score: 3

Support/help

Excellent, if you pay for it. Oracle offers extensive support through email, telephone or sending an employee to help you. You need to have a support arrangement with them though, and this is expensive. There is a fair amount of non-official help/support available online, but the online community for Oracle is not as large as some of the other databases. This has to do with the fact that most Oracle customers pay for support from Oracle directly.
Score: 5

E.5.3 Microsoft SQL Server 2000 Developer/Enterprise Edition

Introduction

Microsoft SQL Server 2000(MS SQL Server 2000) is Microsoft's answer to the dominant, and highly profitable, position Oracle has in the database industry. MS SQL Server 2000 is the culmination of 10 years worth of database technology development by Microsoft. With innovative features such as self-optimizing and enterprise-class reliability and scalability, MS SQL Server has come a long way from its beginnings in the fall of 2003. The last several years MS SQL Server 2000 has had double-digit growth every year on the different Microsoft platforms. Today it has achieved a 40% market share and continues to steal more from its competitors. While earlier regarded as a basic database system not yet ready for prime time, it now competes with its biggest rival, Oracle, in all markets. In both functionality and pricing Microsoft claims an advantage.

Technological suitability

MS SQL Server of course supports all the standard operations of reading, writing and altering data via SQL-queries, and does this well. It also supports more advances features like stores procedures and referential integrity. Microsoft claims it has all the capabilities Oracle has, and then some. We do not require anything of our database that Microsoft SQL Server 2000 doesn't support.
Score: 5

Knowledge of technology

Small, some members have a little experience with Microsoft's SQL Servers. Most have none.
Score: 4

Workload/Complexity of technology

Microsoft is well known for making their products easy to use and comprehensible. SQL Server 2000 includes a set of tools for data management, free of charge. Setup and utilization of this database should be fairly straight forward and problem free considering we are using a Microsoft application

on a Microsoft operating system.
Score: 4

Monetary cost

MS SQL Server 2000 Developer Edition is free of charge for development and testing. Should the customer choose to put this project on a production system, a license for MS SQL Server 2000 Enterprise Edition will be required. These are generally cheaper than Oracle solutions of the same magnitude.
Score: 3

Portability

Very poor. MS SQL Server 2000 is a pure Microsoft product and therefore only runs on a Microsoft operating system.
Score: 2

Integration

Excellent. Microsoft designs their products to be highly integrable with other software and operating systems from Microsoft. The customer has chosen to standardize on Microsoft, so this should pose no big problems.
Score: 5

Support/help

Excellent. As a Microsoft customer you get official support from Microsoft on both MS SQL Server 2000 and the operating system.
Score: 5

E.5.4 Database summary

The different database systems pretty much support all the functionality we need and all of them are available for free for evaluation and development purposes. All of the databases could be used for our project, but we choose to use MS SQL Server 2000 mainly because of its ease of use, the high availability of support from its vendor and the overall best score in table E.3. As noted earlier, the customer's standardization on Microsoft products also makes this choice a good one.

E.6 SUMMARY OF TECHNOLOGICAL PRE STUDY

As seen in the development platform- (E.4.3) and database-summary(E.5.4) the result of this technological pre study is that our solution will be developed with a Microsoft SQL Server 2000 database on the development platform Microsoft .NET. We will also be using Microsoft Visual Studio .NET 2003 as a development environment.

Criterion	**Importance**	**MySQL Evaluation score**	**MySQL Criterion score**	**Oracle Evaluation score**	**Oracle Criterion score**	**MS SQL Evaluation score**	**MS SQL Criterion score**
Technological suitability	H/2.0	4	8	5	10	5	10
Knowledge of technology	M/1.5	5	7,5	3	4,5	4	6
Workload / Complexity of technology	H/2.0	4	8	3	6	4	8
Monetary cost	M/1.5	5	7.5	2	3	3	4,5
Portability	L/1.0	5	5	4	4	2	2
Integration	M/1.5	3	4.5	3	4.5	5	7.5
Support/help	M/1.5	3	4,5	5	7.5	5	7.5
Sum		29	**45**	25	**39.5**	28	**45.5**

Table E.3: Database criteria evaluation

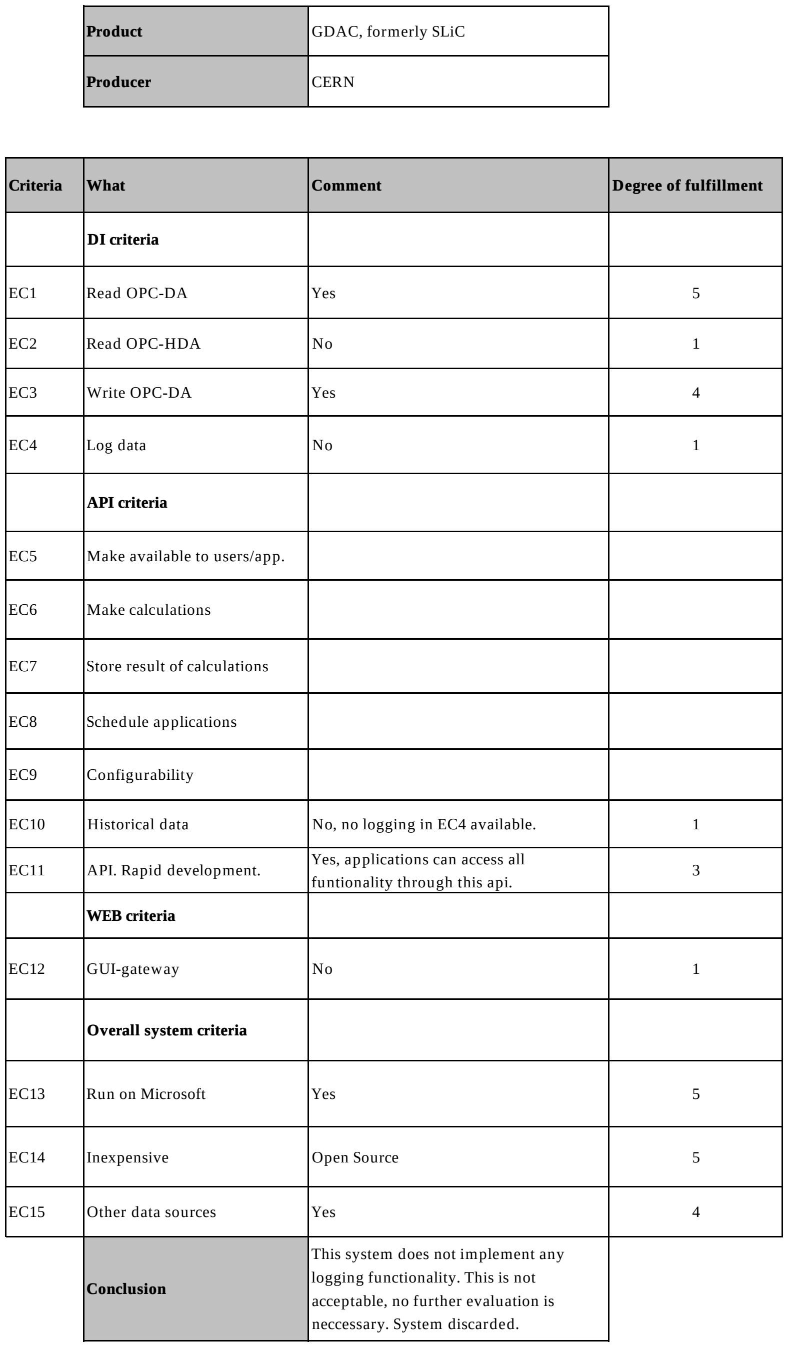
Appendix F
Evaluations forms

F.1 All evaluation forms

Product	DAQBench
Producer	Agile Integration

Criteria	What	Comment	Degree of fulfillment
	DI criteria		
EC1	Read OPC-DA	Yes	5
EC2	Read OPC-HDA	No	1
EC3	Write OPC-DA	Yes	4
EC4	Log data	Yes	4
	API criteria		
EC5	Make available to users/app.	Yes, built in Trending functionality provided. Drag and drop of prebuilt components available. No concept of user defined applications.	3
EC6	Make calculations	Yes, through drag and drop components.	3
EC7	Store result of calculations	Yes	4
EC8	Schedule applications	No	1
EC9	Configurability	No	1
EC10	Historical data	Yes	4
EC11	API. Rapid development.	Yes. Drag and drop components can be combined to make simple applications. No API other than the drag and drop components available.	3
	WEB criteria		
EC12	GUI-gateway	Yes, web interface available for data snapshots and remote monitoring.	5
	Overall system criteria		
EC13	Run on Microsoft	Yes	5
EC14	Inexpensive	NA	1
EC15	Other data sources	No	1
	Conclusion	The limitation of only making drag and drop components available make this solution undesirable for us.	

Figure F.1: DAQBench evaluation



Product	GDAC, formerly SLiC
Producer	CERN

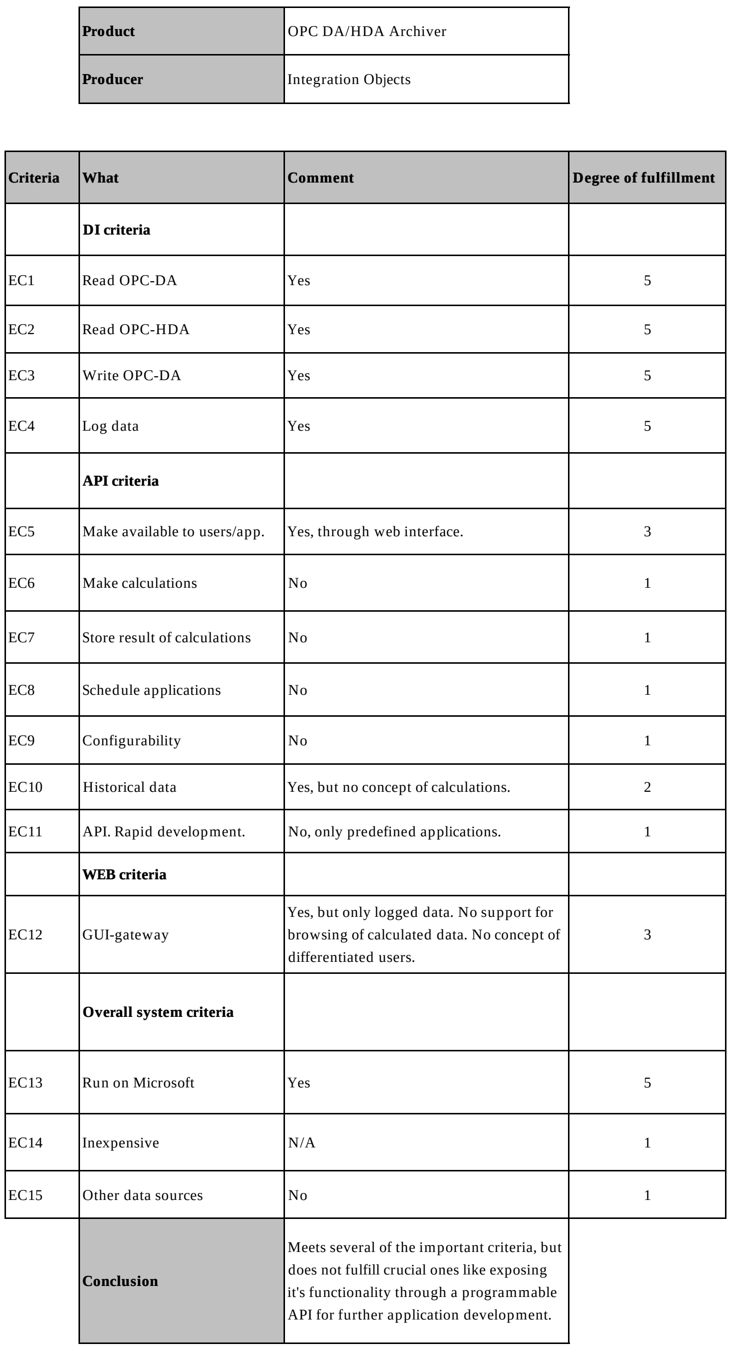
Criteria	What	Comment	Degree of fulfillment
	DI criteria		
EC1	Read OPC-DA	Yes	5
EC2	Read OPC-HDA	No	1
EC3	Write OPC-DA	Yes	4
EC4	Log data	No	1
	API criteria		
EC5	Make available to users/app.		
EC6	Make calculations		
EC7	Store result of calculations		
EC8	Schedule applications		
EC9	Configurability		
EC10	Historical data	No, no logging in EC4 available.	1
EC11	API. Rapid development.	Yes, applications can access all funtionality through this api.	3
	WEB criteria		
EC12	GUI-gateway	No	1
	Overall system criteria		
EC13	Run on Microsoft	Yes	5
EC14	Inexpensive	Open Source	5
EC15	Other data sources	Yes	4
	Conclusion	This system does not implement any logging functionality. This is not acceptable, no further evaluation is neccessary. System discarded.	

Figure F.2: GDAC evaluation

Product	ProcessNet
Producer	Matrikon

Criteria	**What**	**Comment**	**Degree of fulfillment**
	DI criteria		
EC1	Read OPC-DA	Yes	5
EC2	Read OPC-HDA	Yes	5
EC3	Write OPC-DA	Yes	5
EC4	Log data	Yes	5
	API criteria		
EC5	Make available to users/app.	Yes, Matrikon themselves make several products that work with the logged data.	5
EC6	Make calculations	Yes	5
EC7	Store result of calculations	Yes	5
EC8	Schedule applications	Yes, Event scheduler	4
EC9	Configurability	Yes, you can configure calculations through a drag and drop interface	3
EC10	Historical data	Yes	5
EC11	API. Rapid development.	Yes, drag and drop functionality for simple applications. No thorough API available.	3
	WEB criteria		
EC12	GUI-gateway	Yes	5
	Overall system criteria		
EC13	Run on Microsoft	Yes	5
EC14	Inexpensive	No, prices only available on request.	1
EC15	Other data sources	No, OPC only.	1
	Conclusion	Matrikon ProcessNet fulfills most of the evaluation criteria, apparently making it a very good solution candidate. However it does not allow developers to access the collected data through an API, something which is one of the fundamental requirements for our task.	

Figure F.3: Matrikon ProcessNet evaluation



Product	OPC DA/HDA Archiver
Producer	Integration Objects

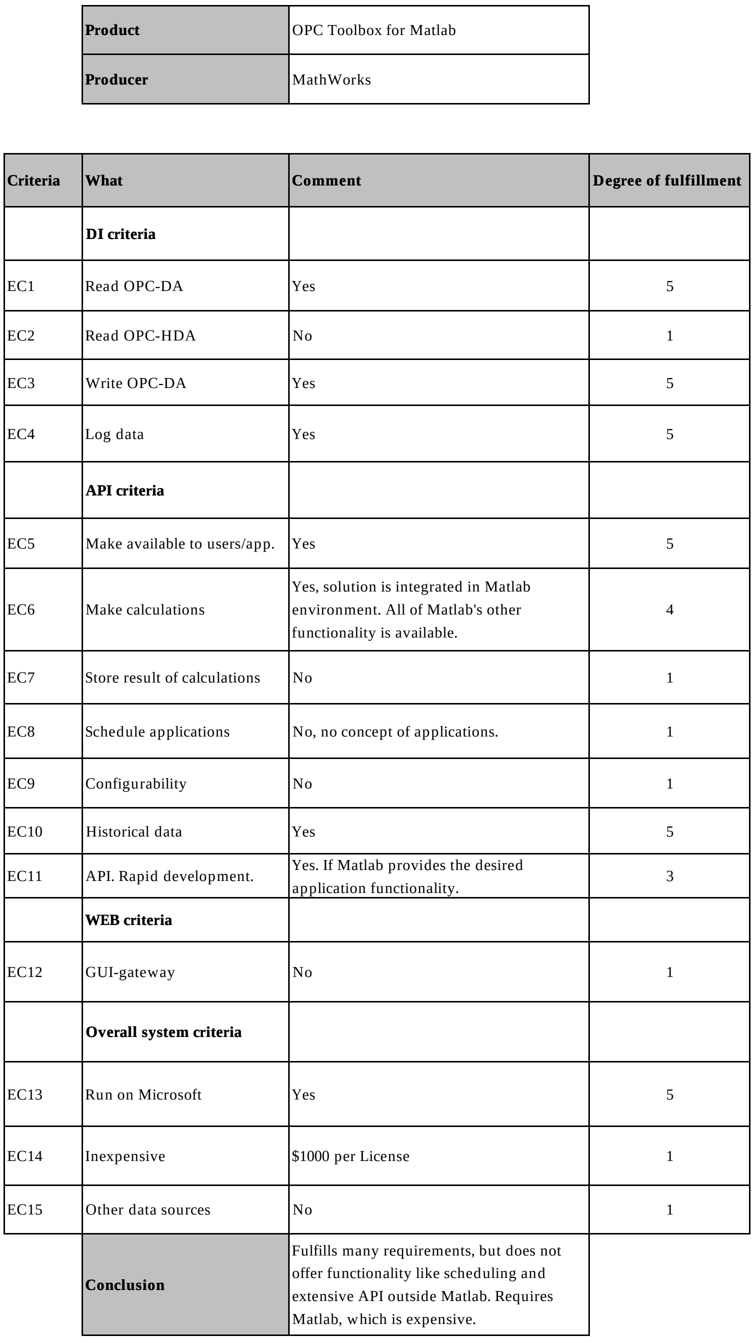
Criteria	**What**	**Comment**	**Degree of fulfillment**
	DI criteria		
EC1	Read OPC-DA	Yes	5
EC2	Read OPC-HDA	Yes	5
EC3	Write OPC-DA	Yes	5
EC4	Log data	Yes	5
	API criteria		
EC5	Make available to users/app.	Yes, through web interface.	3
EC6	Make calculations	No	1
EC7	Store result of calculations	No	1
EC8	Schedule applications	No	1
EC9	Configurability	No	1
EC10	Historical data	Yes, but no concept of calculations.	2
EC11	API. Rapid development.	No, only predefined applications.	1
	WEB criteria		
EC12	GUI-gateway	Yes, but only logged data. No support for browsing of calculated data. No concept of differentiated users.	3
	Overall system criteria		
EC13	Run on Microsoft	Yes	5
EC14	Inexpensive	N/A	1
EC15	Other data sources	No	1
	Conclusion	Meets several of the important criteria, but does not fulfill crucial ones like exposing it's functionality through a programmable API for further application development.	

Figure F.4: OPC DA HDA Archiver evaluation

Product	OPC Systems
Producer	Eldridge Engineering Comp. Inc

Criteria	**What**	**Comment**	**Degree of fulfillment**
	DI criteria		
EC1	Read OPC-DA	Yes	5
EC2	Read OPC-HDA	No	1
EC3	Write OPC-DA	Yes	5
EC4	Log data	Yes	5
	API criteria		
EC5	Make available to users/app.	Yes, but only to users. No concept of user defined applications.	4
EC6	Make calculations	No	1
EC7	Store result of calculations	No	1
EC8	Schedule applications	No	1
EC9	Configurability	No	1
EC10	Historical data	Yes, but no concept of calculations.	3
EC11	API. Rapid development.	No, but source code can be bought and an API can be added to this.	2
	WEB criteria		
EC12	GUI-gateway	Yes, collected data can be viewed in the program. No web interface available.	2
	Overall system criteria		
EC13	Run on Microsoft	Yes	5
EC14	Inexpensive	Yes, Corporate License must be bought. Prize: $14950	1
EC15	Other data sources	No	1
	Conclusion	Offers some required functionality, but both the missing functionality and the prize makes this solution unacceptable to us.	

Figure F.5: OPC Systems Eldridge evaluation



Product	OPC Toolbox for Matlab
Producer	MathWorks

Criteria	What	Comment	Degree of fulfillment
	DI criteria		
EC1	Read OPC-DA	Yes	5
EC2	Read OPC-HDA	No	1
EC3	Write OPC-DA	Yes	5
EC4	Log data	Yes	5
	API criteria		
EC5	Make available to users/app.	Yes	5
EC6	Make calculations	Yes, solution is integrated in Matlab environment. All of Matlab's other functionality is available.	4
EC7	Store result of calculations	No	1
EC8	Schedule applications	No, no concept of applications.	1
EC9	Configurability	No	1
EC10	Historical data	Yes	5
EC11	API. Rapid development.	Yes. If Matlab provides the desired application functionality.	3
	WEB criteria		
EC12	GUI-gateway	No	1
	Overall system criteria		
EC13	Run on Microsoft	Yes	5
EC14	Inexpensive	$1000 per License	1
EC15	Other data sources	No	1
	Conclusion	Fulfills many requirements, but does not offer functionality like scheduling and extensive API outside Matlab. Requires Matlab, which is expensive.	

Figure F.6: OPC Toolbox for Matlab evaluation

Product	OPC Web client
Producer	Iconics

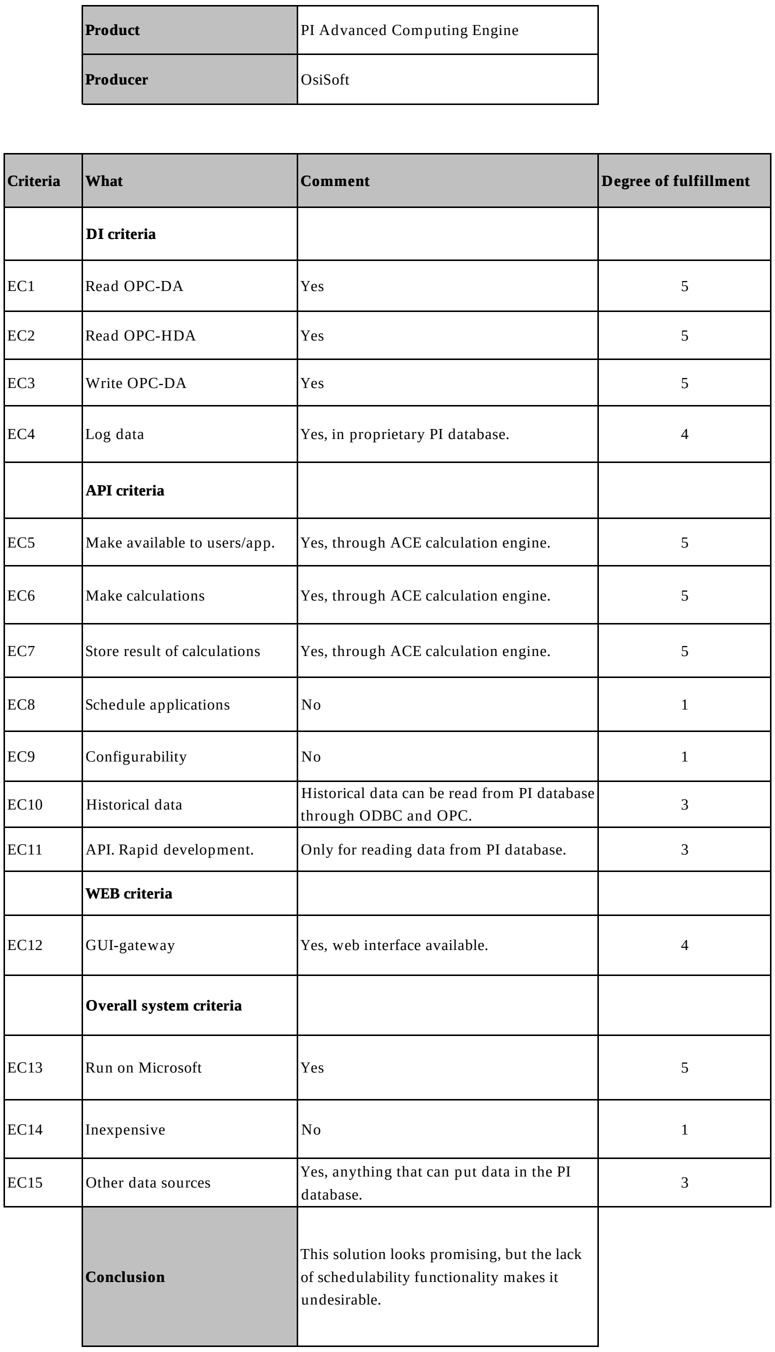
Criteria	What	Comment	Degree of fulfillment
	DI criteria		
EC1	Read OPC-DA	Yes	5
EC2	Read OPC-HDA	Yes	5
EC3	Write OPC-DA	No	1
EC4	Log data	No	1
	API criteria		
EC5	Make available to users/app.	Yes	4
EC6	Make calculations		
EC7	Store result of calculations		
EC8	Schedule applications		
EC9	Configurability		
EC10	Historical data		
EC11	API. Rapid development.		
	WEB criteria		
EC12	GUI-gateway		
	Overall system criteria		
EC13	Run on Microsoft		
EC14	Inexpensive		
EC15	Other data sources		
	Conclusion	Same problem as TrendWorX32 from same producer. No logging ability makes this system completely unacceptable, no further evaluation needed.	

Figure F.7: OPC Web Client evaluation

Product	OPenDA
Producer	open source collaboration

Criteria	**What**	**Comment**	**Degree of fulfillment**
	DI criteria		
EC1	Read OPC-DA	Yes	5
EC2	Read OPC-HDA	No, only DA 1.0 is supported.	1
EC3	Write OPC-DA	Yes	5
EC4	Log data	No, no logging functionality.	1
	API criteria		
EC5	Make available to users/app.		
EC6	Make calculations		
EC7	Store result of calculations		
EC8	Schedule applications		
EC9	Configurability		
EC10	Historical data		
EC11	API. Rapid development.		
	WEB criteria		
EC12	GUI-gateway		
	Overall system criteria		
EC13	Run on Microsoft	Yes	5
EC14	Inexpensive		
EC15	Other data sources		
	Conclusion	No logging ability means the system will fail most of the other critieria. No further evalutation needed, system discarded.	

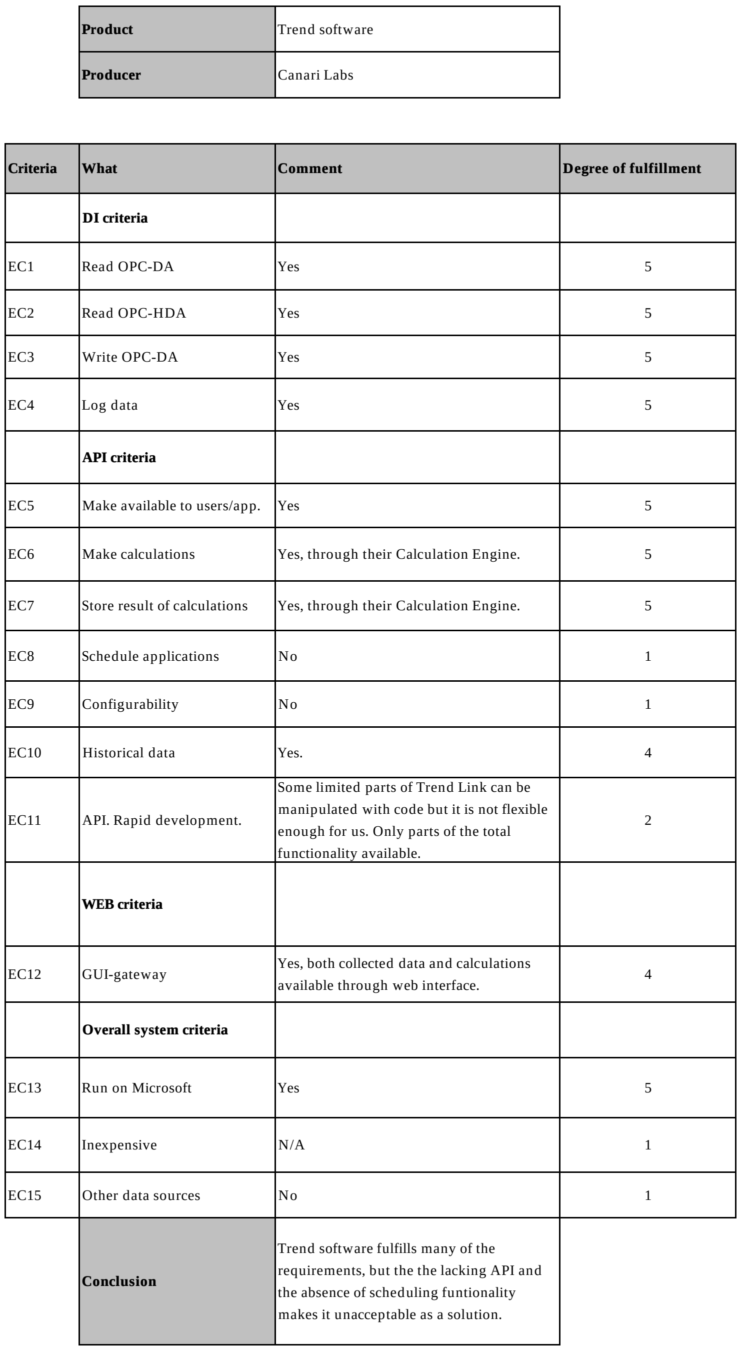
Figure F.8: OpenDA evaluation



Product	PI Advanced Computing Engine
Producer	OsiSoft

Criteria	What	Comment	Degree of fulfillment
	DI criteria		
EC1	Read OPC-DA	Yes	5
EC2	Read OPC-HDA	Yes	5
EC3	Write OPC-DA	Yes	5
EC4	Log data	Yes, in proprietary PI database.	4
	API criteria		
EC5	Make available to users/app.	Yes, through ACE calculation engine.	5
EC6	Make calculations	Yes, through ACE calculation engine.	5
EC7	Store result of calculations	Yes, through ACE calculation engine.	5
EC8	Schedule applications	No	1
EC9	Configurability	No	1
EC10	Historical data	Historical data can be read from PI database through ODBC and OPC.	3
EC11	API. Rapid development.	Only for reading data from PI database.	3
	WEB criteria		
EC12	GUI-gateway	Yes, web interface available.	4
	Overall system criteria		
EC13	Run on Microsoft	Yes	5
EC14	Inexpensive	No	1
EC15	Other data sources	Yes, anything that can put data in the PI database.	3
	Conclusion	This solution looks promising, but the lack of schedulability functionality makes it undesirable.	

Figure F.9: PI ACE evaluation

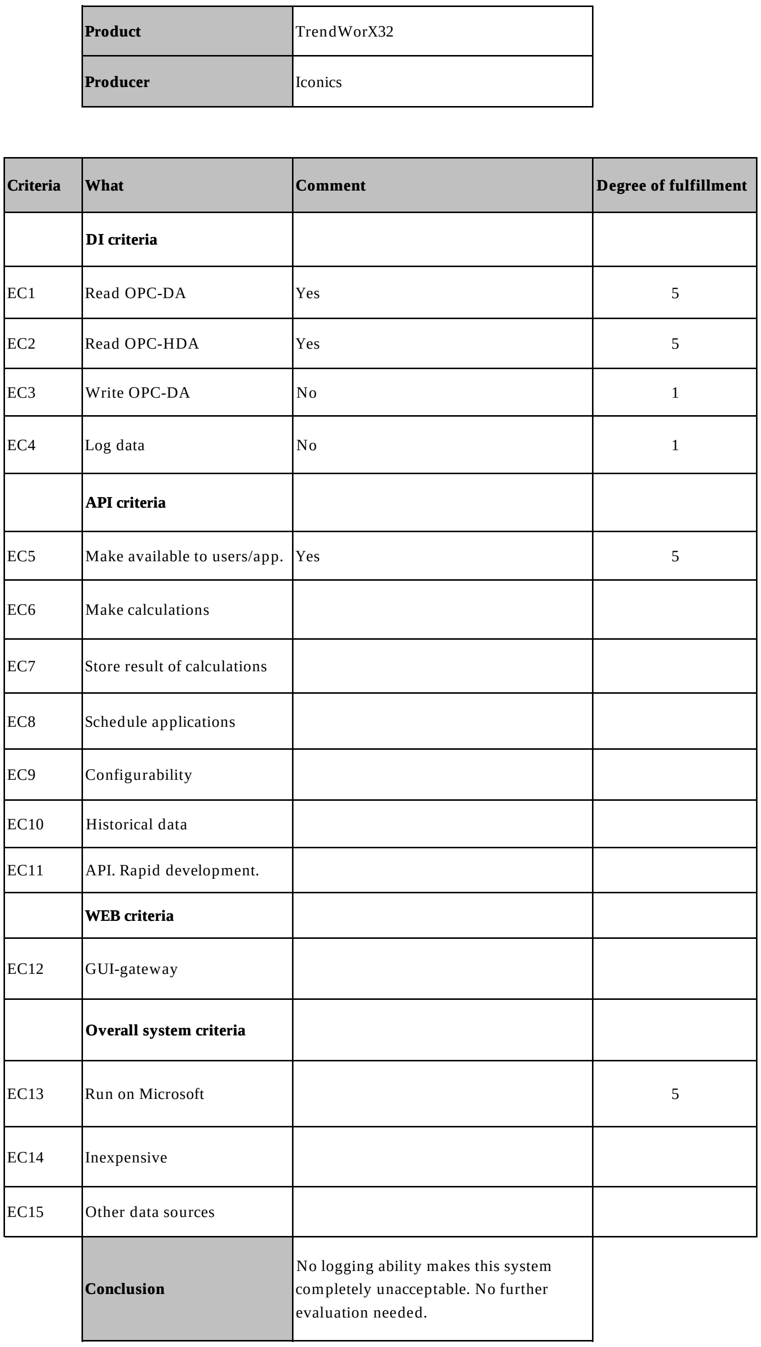


Product	Trend software
Producer	Canari Labs

Criteria	**What**	**Comment**	**Degree of fulfillment**
	DI criteria		
EC1	Read OPC-DA	Yes	5
EC2	Read OPC-HDA	Yes	5
EC3	Write OPC-DA	Yes	5
EC4	Log data	Yes	5
	API criteria		
EC5	Make available to users/app.	Yes	5
EC6	Make calculations	Yes, through their Calculation Engine.	5
EC7	Store result of calculations	Yes, through their Calculation Engine.	5
EC8	Schedule applications	No	1
EC9	Configurability	No	1
EC10	Historical data	Yes.	4
EC11	API. Rapid development.	Some limited parts of Trend Link can be manipulated with code but it is not flexible enough for us. Only parts of the total functionality available.	2
	WEB criteria		
EC12	GUI-gateway	Yes, both collected data and calculations available through web interface.	4
	Overall system criteria		
EC13	Run on Microsoft	Yes	5
EC14	Inexpensive	N/A	1
EC15	Other data sources	No	1

Conclusion	Trend software fulfills many of the requirements, but the the lacking API and the absence of scheduling funtionality makes it unacceptable as a solution.

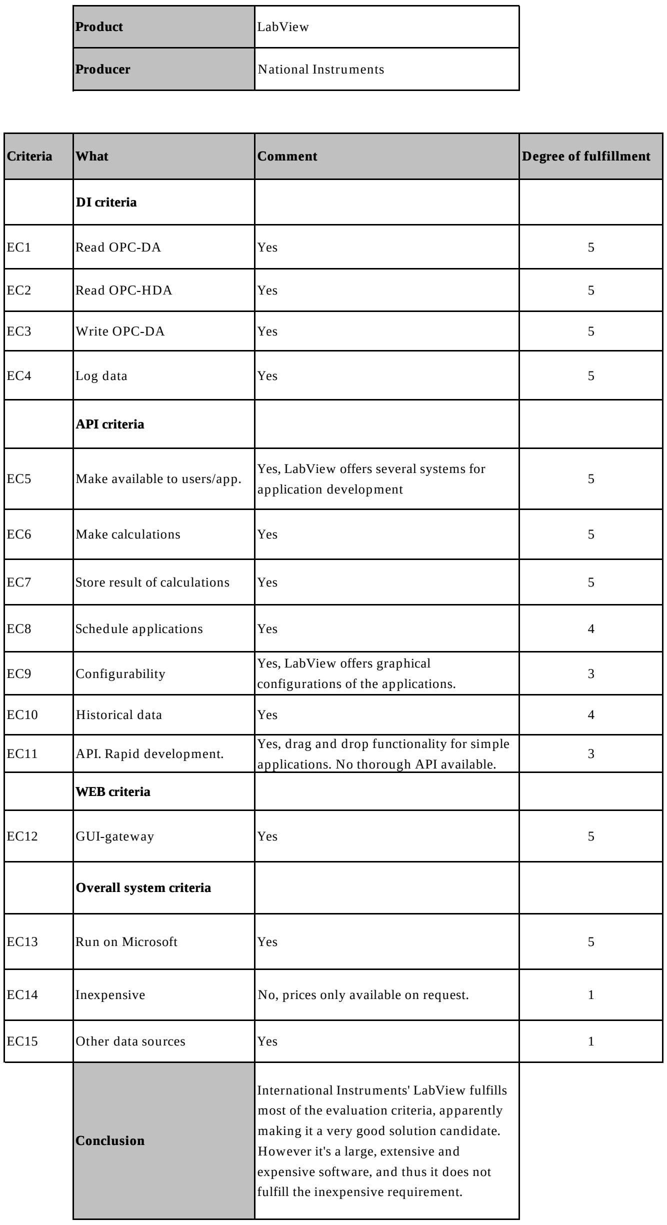
Figure F.10: Trend Software evaluation



Product	TrendWorX32
Producer	Iconics

Criteria	What	Comment	Degree of fulfillment
	DI criteria		
EC1	Read OPC-DA	Yes	5
EC2	Read OPC-HDA	Yes	5
EC3	Write OPC-DA	No	1
EC4	Log data	No	1
	API criteria		
EC5	Make available to users/app.	Yes	5
EC6	Make calculations		
EC7	Store result of calculations		
EC8	Schedule applications		
EC9	Configurability		
EC10	Historical data		
EC11	API. Rapid development.		
	WEB criteria		
EC12	GUI-gateway		
	Overall system criteria		
EC13	Run on Microsoft		5
EC14	Inexpensive		
EC15	Other data sources		
	Conclusion	No logging ability makes this system completely unacceptable. No further evaluation needed.	

Figure F.11: TrendworX32 evaluation



Product	LabView
Producer	National Instruments

Criteria	**What**	**Comment**	**Degree of fulfillment**
	DI criteria		
EC1	Read OPC-DA	Yes	5
EC2	Read OPC-HDA	Yes	5
EC3	Write OPC-DA	Yes	5
EC4	Log data	Yes	5
	API criteria		
EC5	Make available to users/app.	Yes, LabView offers several systems for application development	5
EC6	Make calculations	Yes	5
EC7	Store result of calculations	Yes	5
EC8	Schedule applications	Yes	4
EC9	Configurability	Yes, LabView offers graphical configurations of the applications.	3
EC10	Historical data	Yes	4
EC11	API. Rapid development.	Yes, drag and drop functionality for simple applications. No thorough API available.	3
	WEB criteria		
EC12	GUI-gateway	Yes	5
	Overall system criteria		
EC13	Run on Microsoft	Yes	5
EC14	Inexpensive	No, prices only available on request.	1
EC15	Other data sources	Yes	1
	Conclusion	International Instruments' LabView fulfills most of the evaluation criteria, apparently making it a very good solution candidate. However it's a large, extensive and expensive software, and thus it does not fulfill the inexpensive requirement.	

Figure F.12: LabView evaluation

Part III

Requirements Specification

Chapter 16 Introduction

16.1 Purpose

The goal of this document is to structure and simplify the work that is to be carried out in the construction phase and the implementation phase of the project.

16.2 Scope

Where the IEEE 830 has one section where the specific requirements are listed, we chose to divide the specific requirements in three sections, one section per module. The reason for this modification of the IEEE standard is that the product to be developed consists of three different parts. We found it appropriate to develop different requirements for each part, because each part has different levels of user visibility.

The software modules to be produced are:

- **DI** – Data Interchanger
  The DI collects data from one or several OPC servers. See section 18.2 for the specific requirements of this module.
- **APP API** – Application Programmer Interface for applications
  The APP API provides application developers with a single interface for accessing system functionality. See section 18.3 for the specific requirements of this module.
- **WEB** – The web portal
  A graphical user interface for presenting collected raw data, as well as the data generated by the applications mentioned above. See section 18.4 for the specific requirements of this module.

16.3 References

This document is based on the IEEE standard 830 [16]. The textual use cases we'll use for describing the specific requirements are following the guidelines from the course TDT4175 Information Systems [24].

16.4 Overview

The rest of the requirements specification is organized as follows: Chapter 17 contains an overall description of the system to be implemented. This includes a sketch of the solution we have come up with, and requirements to interfaces and user characteristics relevant to our system. Then the specific requirements are presented in chapter 18. The main part of this chapter is the detailed description of requirements organized by the three software modules described above. It also contains several other sections on specific requirements to the system in general, in compliance with the IEEE standard.

Chapter 17 Overall description

17.1 Product perspective

The following section will describe our solution, called the EasyIT-solution. In principle, it's ambition is to solve some of the limitations attached with current work processes. In order to succeed, it must offer real-time access to data in control systems. Data is gathered on different sampling rates and tags may be sampled as frequent as once per second. Several applications might sample the same tag at different rates. So, the solution is supposed to give analysts easy access to real-time data. Accordingly, effective development and testing of suitable algorithms for data-analysis and device-verification should be within reach. The system must also provide access to historical data if needed. In brief, such a system will have a wide range of applications.

Our solution consists of three parts: DI, APP API and WEB. Roughly, the data interchanger (DI) will serve as an interface to available data sources, mainly OPC-servers. Applications may access these data through an API on the DI. Eventually, results produced by these applications are visualized on web-pages (WEB).

17.1.1 System interfaces

Data Interchanger

The data interchanger will be the cornerstone of our solution. This part provides a uniform interface to one or more OPC-servers. In longer terms, configuration of the interchanger should be available through a graphical user interface. Possible configuration activities are adding/removing available OPC-servers and monitoring of applications. In general terms, the DI interface is accessible through an application programmer interface (API).

OPC-servers support both polling and events for data access. Consequently, the DI should also support such mechanisms. Each application is free to subscribe to tags at different sample rates and multiple applications may subscribe to the same tags. It is left to the DI to gather tags and distribute them at correct rates, and to minimize the load on the OPC-severs and the network in general.

Further, the interchanger must be able to accumulate data. The clients are required to explicitly specify which tags are to be saved. The DI will delete historical data older than a given amount of time, say 6 months.

Finally, the DI will implement functionality to save data as OPC-tags. Applications accessing the interchanger can therefore save produced and processed data, making data available for other applications through the DI-interface.

Application API

The actual purpose of the DI is twofold; to serve applications with real-time data from OPC-servers *and* save data produced by DI's scheduled applications. APP API serves as a layer between DI and the applications. Applications are basically scripts that will be scheduled for execution based on time. Some programs perform only routine operations, serving as tools for more complex applications. Organization and monitoring of these applications are handled by the data-interchanger. Figure 17.2 sketches the composition of the EasyIT-system. As shown, the DI-part consists of a database and an interface.

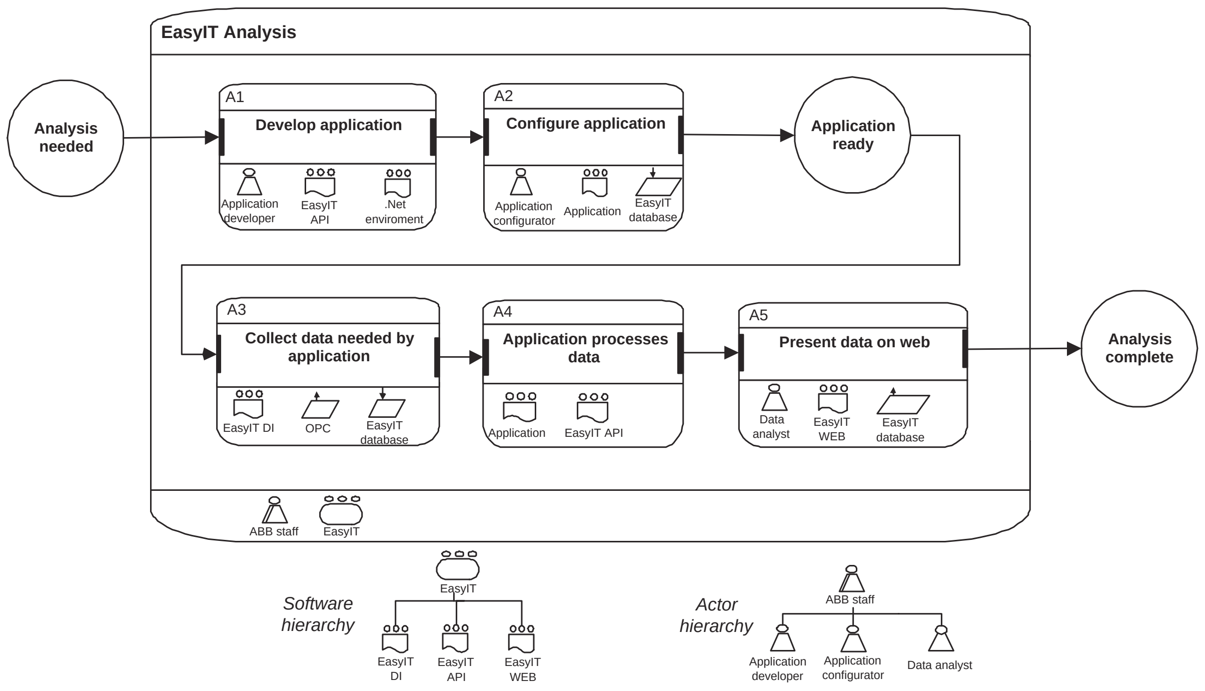


Figure 17.1: Action Port Model (APM) of the EasyIT-system

Every application in the system accesses the tags stored in the OPC-servers through the interface on the DI. This interface is referred to as the APP(-lication) API. Writing data back to the database is also carried out by the DI.

WEB API

Eventually, as the illustration shows, the web interface visualizes data. Please note that the web-pages obtain data from the DI through the WEB API and not from the applications themselves.

17.1.2 User interfaces

End users of **EasyIT** will interact with the system using a graphical interface in the WEB-module. Programmers using EasyIT as a skeleton, will (by their applications) interact through the APP API.

The main user interface of EasyIT is the web interface. The WEB will serve as a portal of web pages visualising application data. Also the WEB interface will facilitate configuration activities. The activities in question are configuration of user and user-access, applications and set-up of web-pages. Since this configuration can be performed manually by editing text files or database entries, please note that such graphical configuration interface do not have high propriety at the moment.

17.1.3 Hardware interfaces

Our system will not interface directly with any hardware. Nevertheless we will assume that the EasyIT system will run on PC servers, that is; A simple computer with x86-architecture. Most applications will run locally, but there are no major difficulties in implementing a remoting-possibility for running computation-intensive applications on dedicated servers.

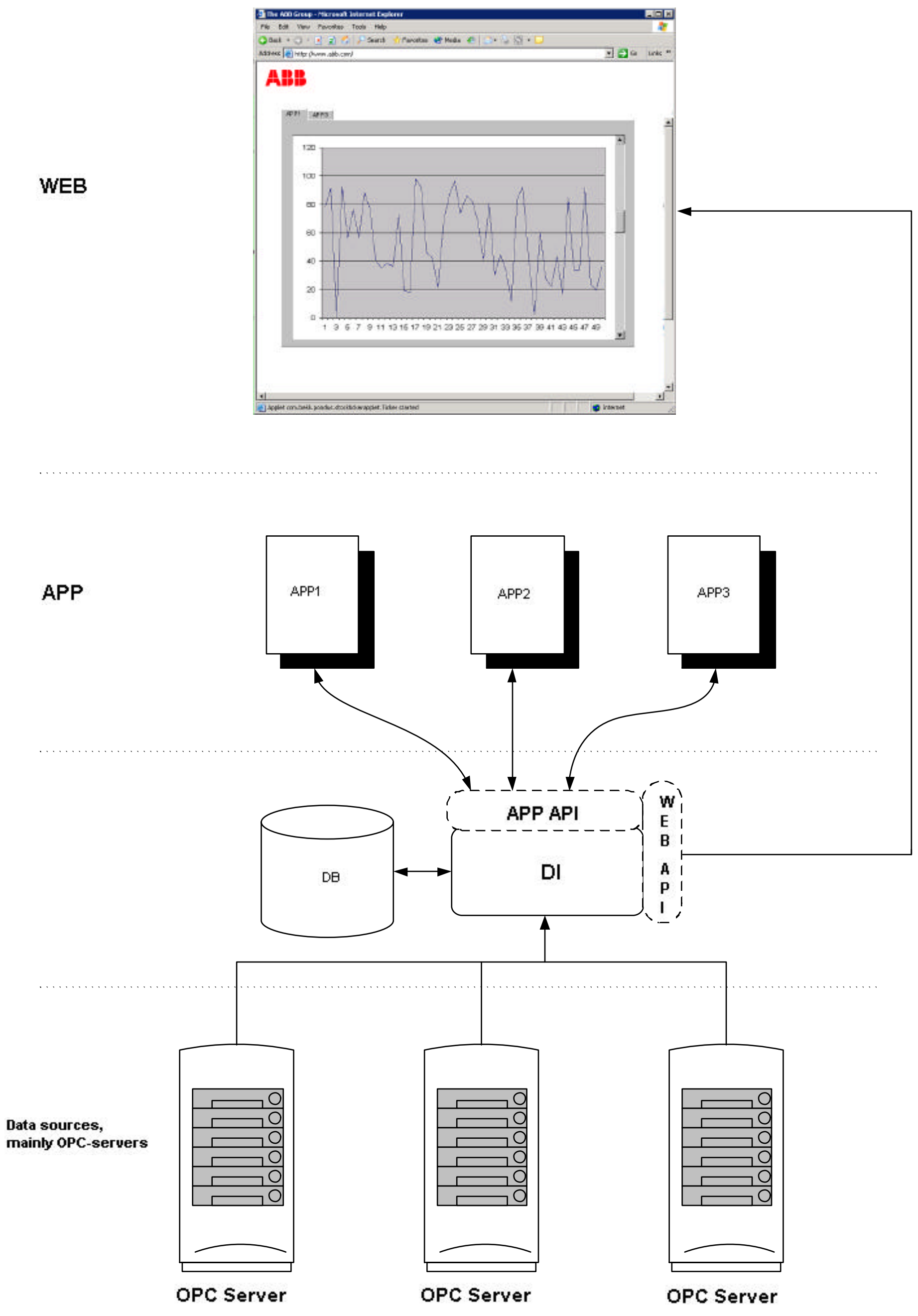


Figure 17.2: Conceptual model of EasyIT 1

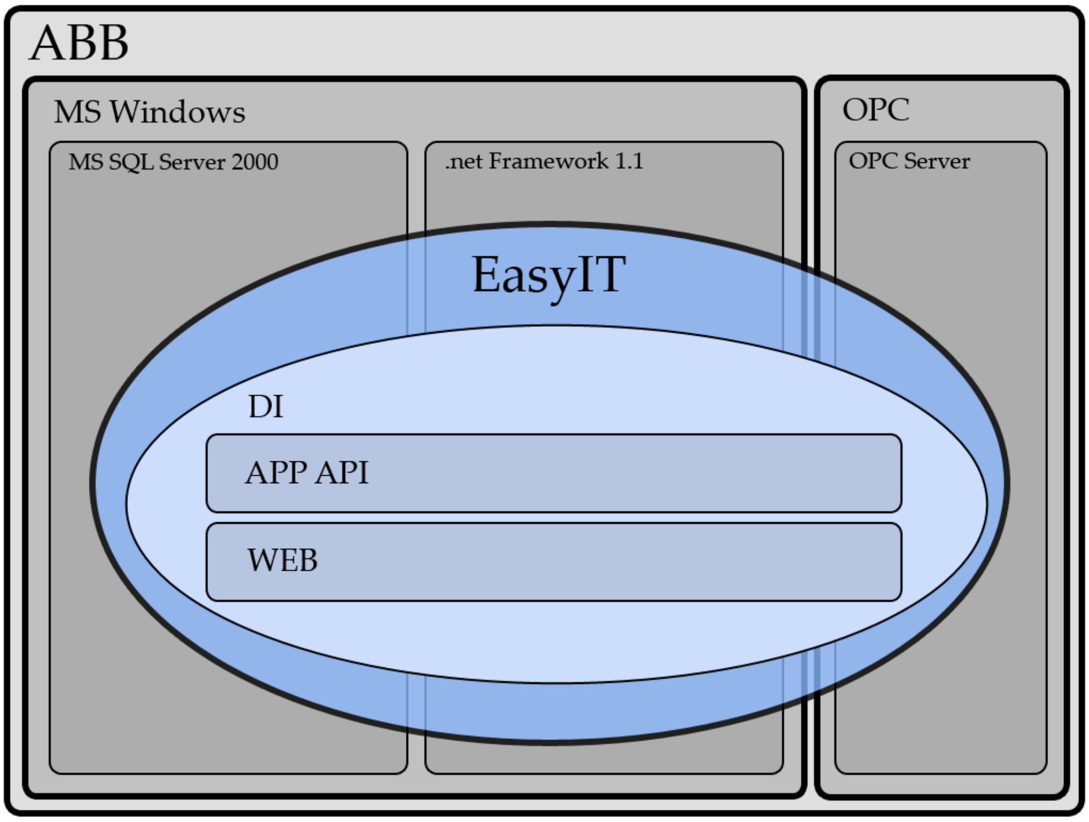


Figure 17.3: Conceptual model of EasyIT 2

17.1.4 Software interfaces

The EasyIT system consists of three different components, the DI, APP API and WEB API. All these components will communicate with each other and other software systems. This section will discuss some of the software interfaces involved in the system.

Operating system

As Figure 17.3 depicts, the entire system is running on a platform with Microsoft Windows operating system (at ABB; Windows Server 2003).

Programming language and development environment

The implementation of EasyIT also includes an arrangement for easy development of applications running on the system (using EasyIT as a skeleton). The API should provide easy access to data sources and abstract the technical difficulties with i.e. OPC-subscription and database connections. Also, the methods and mechanisms in the interface should be the same regardless of what kind of data source the application are gathering data from. As discussed in the pre-study, our choice of development environment is Microsoft Visual Studio .NET platform. This platform supports both web-development and application development.

Web server

The WEB system will run on a web server. The obvious choice is Microsoft Internet Information Server (IIS) which is an integrated part of Windows Server.

DBMS (Database management system)

The DI will gather data from data sources and store some of these data in a database. Also, configuration data and application data will also be stored in a database. Potentially, the rate of reads and writes could be very large, so a critical parameter when it comes to evaluation of DBMS is performance.

17.1.5 Communications interfaces

The server running DI must support TCP/IP for communicating with WEB and OPC.

17.2 User characteristics

Users of the EasyIT system may be divided into three categories and one person may hold more than one of these roles

- System administrator
- Application developer
- Data analyst

System administrator

System administrators have privileges to add and edit users. Also administrators are allowed to configure applications and the DI.

Application developer

The developers develop applications that is to perform analysis on real-time or historical data.

Data analyst

The data analysts study the output data from the applications. These data will typically be visualised on web-pages.

17.3 Apportioning of requirements

Some of the requirements specified in this specification have low priority. These requirements are not considered vital for the system and can be delayed for future versions.

17.4 APP (applications) and the WEB (presentation system)

Eventually, as the figure 17.2 shows, the web interface visualizes data. Please note that the web-pages obtain data from the (database within) DI and not from the applications themselves.

To summarize; the DI-part of the system reads requested real-time data from the data sources (mainly OPC). If necessary, selected data is stored by the interchanger. Applications process data obtained by the interchanger and write output data back to the interchanger. The interchanger then saves this data as tags, making output available for other applications. The web-interface supports various ways of visualising output data. At the time of writing, two possible methods are considered; table and plotting (as shown in the figure). So, a couple of problems need to be sorted out. First, how do the web pages know which data to deal with and how does the web page select an appropriate way to visualize data? Furthermore, some outputs are not intended for visualisation. How does the web-interface know which applications have data to visualize? Bringing focus to scheduled applications, the DI also needs a way to identify and organise these.

17.4.1 Configuration of applications

There is certainly a need for some method of application configuration. Basically, applications process data and web-pages visualise the output. Thus, in addition to the applications, both the DI- and WEB-part of the system needs access to the configurations. It is not decided yet whether the configurations will be stored as text-files or in some database-structure. At this point of the development process, the term *configuration* is therefore regarded as a collection of data specifying the behaviour of one application. The following paragraphs describe some of the elements in this configuration. The paragraph on input data is included for completeness.

Input data

The applications are free to access any data in the data sources, so there are no reasons to list input data in the configuration whatsoever.

Output data

Every time an application outputs some data, the DI saves these as tags. These tags will reside in the database with other tags obtained from various data sources (read: OPC-servers). Therefore, the interchanger must label output data in such a way that it is possible to separate these tags from OPC-data and identify the application which produced the tag. Thus, listing the output-data in the configuration might be redundant. Occasionally, there might be cases where such a listing would be favourable. Say the applications have never run. If that is the case, there would be no other way to determine what data the application will produce, but reading the configuration.

Published data

The EasyIT-solution makes it possible to visualise data from applications on web-pages. These visualized data are called *published data*. In mathematical terms, the application's published data is a subset of the output data. The configuration also needs to specify how the published data are supposed to be visualized. So, when a certain web page is requested, the web-page first reads the configuration (to figure out which way to represent the data). Then the web-system obtains data from the interchanger and finally prints data using the appropriate visualization method.

Scheduling information

The DI is also responsible for starting scheduled applications. One essential parameter of these applications is the rate at which the applications are fired. Imagine an application calculating the mean value of a number of tags. Other applications may need this value for further calculations, and the mean value is supposed to be refreshed at a certain rate, say every ten minutes. First of all, the configuration has to specify whether the associated application is scheduled. If so, the configuration also sets a refresh rate. The DI then has enough information to fire scheduled applications at appropriate time-intervals.

Chapter 18
Specific requirements

18.1 Introduction

The specific requirements for each module are specified with three elements.

- A graphical use case diagram showing all actors and major tasks to be carried out through the module
- A list of the functional requirements for the module
- Textual use cases describing important sequences of actions regarding the module

The textual use case descriptions are presented in tables, providing the following properties of the requirement:

- **Use case name:** Identifier of the use case.
- **Priority:** Either *High (H), Medium (M) or Low (L)*. All of the requirements are meant to be implemented, but we have categorized them into these three levels of importance. High priority requirements are critical for the project, requirements labelled "M" are still important, but not as critical to implement. If we run out of time, requirements labelled "L" are considered not important enough to be implemented.
- **Iteration:** Either *facade, filled, focused or finished.* "Filled" is the typical iteration value for the specific requirements stage.
- **Summary:** A short description of the use case
- **Accompanying requirements:** References to the requirements covered by this use case
- **Basic course of events:** The steps that the actors and the system go through to accomplish the goal.
- **Alternative paths:** Less common steps than the basic course.
- **Exception paths:** Paths taken when the errors occur.
- **Triggers:** The entry criteria for the use case, i.e., what initiates it.
- **Pre-conditions:** Conditions that must be true before the use case can be performed.
- **Post-conditions:** What will be true when the use case is completed.
- **Date:** The date when the textual use case was created.
- **Author:** The author of the textual use case

18.2 Requirements for the DI

18.2.1 Graphical overall use case for DI

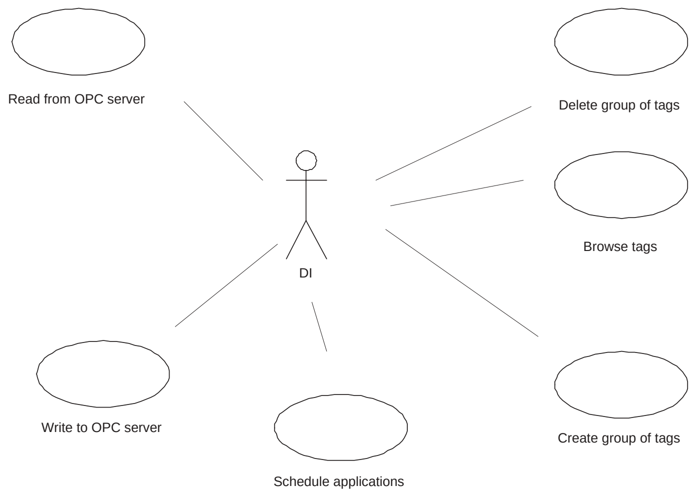


Figure 18.1: Overall use case for DI

18.2.2 Functional requirements for the DI

18.1 contains a list of the requirements on the DI module. Textual use cases follow for each of the requirements.

ID	Requirement	Priority	Use case
	Data read / write		
DI-1	DI reads synchronous from OPC-server.	H	DI-UC-18.2
DI-2	DI start asynchronous continual read (subscribes to a data tag)	H	DI-UC-18.3
DI-3	DI stops a continually asynchronous read. (A subscribed tag)	H	DI-UC-18.4
DI-4	DI pauses a continually asynchronous read. (A subscribed tag)	L	DI-UC-18.5
DI-5	DI continues a continually asynchronous read. (A subscribed tag)	L	DI-UC-18.6
DI-6	DI writes synchronous to OPC-server.	H	DI-UC-18.7
	Scheduling		
DI-7	DI schedules applications.	H	DI-UC-18.8
	Handling tags		
DI-8	Create group of tags.	M	DI-UC-18.9
DI-9	Delete group of tags.	M	DI-UC-18.10
DI-10	Browse tags.	L	DI-UC-18.11

Table 18.1: DI requirement list

18.2.3 Textual use case diagrams for the DI

Use case name:	**DI reads synchronous from OPC-server**
ID:	DI-UC-18.2
Priority:	H
Iteration:	Filled
Summary:	DI reads data from an OPC-server. The read is synchronous, and therefore blocking.
Accompanying requirements:	DI-1
Basic course of events:	1. DI is told directly by APP API or scheduler that a data tag, or group of data tags, needs to be updated. 2. DI checks tag, or group, and figures out which OPC-server(s) need to be contacted. 3. DI opens connections to the server(s) in 2) and asks for the current value of the tag, or group of tags, synchronously. 4. DI blocks until the server(s) reply with the current value of the tag, or group of tags. 5. The entity that started 1), either APP API or scheduler, is returned the value.
Alternative paths:	5a Logging is also requested. • 5a1. The returned value is logged to the database. • 5a2. The entity that started 1), either APP API or scheduler, is returned the value.
Exception paths:	• 2a If DI does not recognize the tag, or group, and therefore can not figure out which server(s) to contact, an exception occurs and the basic course of events is aborted. • 3a If DI can not open a connection to the server(s), an exception occurs and the basic course of events is aborted. • 4a If DI waits more than a given time length or connections to the server(s) are lost, an exception occurs and basic course of events are aborted. • When the basic course of events is aborted, we return an error code to the entity that started it.
Triggers:	APP API or scheduler tells DI it needs the current value of a tag, or group.
Pre-conditions:	DI is running.
Post-conditions:	
Date:	26.09.2004
Author:	Jan Ove S. Olsen

Table 18.2: DI-UC-18.2

Use case name:	**DI start asynchronous continual read(subscribes to a data tag)**
ID:	DI-UC-18.3
Priority:	H
Iteration:	Filled
Summary:	DI reads data from an OPC-server. The read is asynchronous and continual; therefore non-blocking.
Accompanying requirements:	DI-3
Basic course of events:	1. DI is told directly by APP API or scheduler that a data tag, or group of data tags, needs to be read continuously at a given rate. 2. DI checks tag, or group, and figures out which OPC-server(s) need to be contacted. 3. DI opens connections to the server(s) in 2 and asks for the value of the tag, or group of tags, to be returned to it repeatedly every n seconds. 4. DI continues with other tasks. 5. DI is interrupted by OPC-server(s) with a reply with the current value of the tag, or group of tags. 6. The entity that started 1), either APP API or scheduler, is given the value. 7. 5) and 6) is repeated over and over again until DI-UC-18.4, DI-UC-18.5 or an exception occurs.
Alternative paths:	6a. Logging is also requested • 6a1. The returned value is logged to the database. • 6a2. The entity that started 1), either APP API or scheduler, is returned the value. 6b. If the request is for just storing data to the database (for later historical trending), 6) and 7) become: • 6b1. The returned value is logged to the database. • 6b2. The entity that started 1) is returned an OK-message
Exception paths:	• 2. If DI does not recognize the tag, or group, and therefore can not figure out which server(s) to contact, an exception occurs and the basic course of events is aborted. • 3. If DI can not open a connection to the server(s), an exception occurs and the basic course of events is aborted. • 5. If DI's connection with the server(s) is lost, an exception occurs and the basic course of events is aborted. • When the basic course of events is aborted, we return an error code to the entity that started it.
Triggers:	APP API or scheduler tells DI it needs the continually updated value of a tag, or group.
Pre-conditions:	DI is running.
Date:	26.09.2004
Author:	Jan Ove S. Olsen

Table 18.3: DI-UC-18.3

Use case name:	**DI stops a continually asynchronous read.(A subscribed tag)**
ID:	DI-UC-18.4
Priority:	H
Iteration:	Filled
Summary:	DI stops a previously started asynchronous and continual read.
Accompanying requirements:	DI-4
Basic course of events:	1. DI is told directly by APP API or scheduler that a data tag, or group of data tags, it is currently reading, to stop reading it/them. 2. DI checks tag, or group, and figures out which of its open connections it needs to modify. 3. DI sends abort message to the server(s) in 2) and closes the connection to them. 4. The entity that started 1), either APP API or scheduler, is returned an OK-message.
Alternative paths:	
Exception paths:	• 2a. If DI does not recognize the tag, or group, and therefore can not figure out which server(s) to contact, an exception occurs and the basic course of events is aborted. • 3a. If DIs connection with the server(s) is lost, an exception occurs and the basic course of events is aborted. • When the basic course of events is aborted, we return an error code to the entity that started it.
Triggers:	APP API or scheduler tells DI is needs the continually updated value of a tag, or group.
Pre-conditions:	DI is running.
Post-conditions:	
Date:	26.09.2004
Author:	Jan Ove S. Olsen

Table 18.4: DI-UC-18.4

Use case name:	**DI pauses a continually asynchronous read**
ID:	DI-UC-18.5
Priority:	L
Iteration:	Filled
Summary:	DI pauses a previously started asynchronous and continual read.
Accompanying requirements:	DI-5
Basic course of events:	1. DI is told directly by APP API or scheduler that a data tag, or group of data tags, it is currently reading, to temporarily stop reading it/them. 2. DI checks tag, or group, and figures out which of its open connections it needs to modify. 3. DI sends pause message to the server(s) in 2), but keeps the connections to them open. 4. The entity that started 1), either APP API or scheduler, is returned an OK-message.
Alternative paths:	
Exception paths:	• 2a. If DI does not recognize the tag, or group, and therefore can not figure out which server(s) to contact, an exception occurs and the basic course of events is aborted. • 2b. If DI is not currently continually reading the tag or group, an exception occurs and the basic course of events it aborted. • 3. If DIs connection with the server(s) is lost, an exception occurs and the basic course of events is aborted. • When the basic course of events is aborted, we return an error code to the entity that started it.
Triggers:	APP API or scheduler tells DI it no longer needs the continually updated value of a tag, or group, but it will need it again later.
Pre-conditions:	DI is running.
Post-conditions:	
Date:	26.09.2004
Author:	Jan Ove S. Olsen

Table 18.5: DI-UC-18.5

Use case name:	**DI continues a previously paused continually asynchronous read**
ID:	DI-UC-18.6
Priority:	L
Iteration:	Filled
Summary:	DI continues a previously started asynchronous and continual read.
Accompanying requirements:	DI-6
Basic course of events:	1. DI is told directly by APP API or scheduler that a data tag, or group of data tags, it has currently paused reading from, to start reading it/them again. 2. DI checks tag, or group, and figures out which of its open connections it needs to modify. 3. DI sends continue message to the server(s) in 2). 4. The entity that started 1), either APP API or scheduler, is returned an OK-message.
Alternative paths:	
Exception paths:	• 2a. If DI does not recognize the tag, or group, and therefore can not figure out which server(s) to contact, an exception occurs and the basic course of events is aborted. • 2b. If DI is not currently continually reading and has not paused the reading of the tag or group, an exception occurs and the basic course of events it aborted. • 3. If DIs connection with the server(s) is lost, an exception occurs and the basic course of events is aborted. • When the basic course of events is aborted, we return an error code to the entity that started it.
Triggers:	APP API or scheduler tells DI it no longer needs the continually updated value of a tag, or group, but it will need it again later.
Pre-conditions:	DI is running.
Post-conditions:	
Date:	30.09.2004
Author:	Jan Ove S. Olsen

Table 18.6: DI-UC-18.6

Use case name:	**DI writes synchronous to OPC-server.**
ID:	DI-UC-18.7
Priority:	H
Iteration:	Filled
Summary:	DI writes data from an OPC-server. The write is synchronous, and therefore blocking.
Accompanying requirements:	DI-7
Basic course of events:	1. DI is told directly by APP API or scheduler that a data tag, or group of data tags, needs to be written. 2. DI checks tag, or group, and figures out which OPC-server(s) need to be contacted. 3. DI opens connections to the server(s) in 2) and writes the new values of the tag, group of tags, synchronously. 4. DI blocks until the server(s) return the call. 5. The entity that started 1), either APP API or scheduler, is returned an OK-message.
Alternative paths:	
Exception paths:	• 2a. If DI does not recognize the tag, or group, and therefore can not figure out which server(s) to contact, an exception occurs and the basic course of events is aborted. • 3a. If DI can not open a connection to the server(s), an exception occurs and the basic course of events is aborted. • 4a. If DI waits more than a given time length or connections to the server(s) are lost, an exception occurs and basic course of events are aborted. • When the basic course of events is aborted, we return an error code to the entity that started it.
Triggers:	APP API or scheduler tells DI is needs the current value of a tag, or group.
Pre-conditions:	DI is running.
Post-conditions:	
Date:	26.09.2004
Author:	Jan Ove S. Olsen

Table 18.7: DI-UC-18.7

Use case name:	**DI schedules applications**
ID:	DI-UC-18.8
Priority:	H
Iteration:	Filled
Summary:	DI must be able to start applications automatically at given time intervals.
Accompanying requirements:	DI-7
Basic course of events:	1. DI checks its scheduling database. 2. If it finds out some application should run now, it finds all the information it needs in the database required to start it. 3. DI runs the application.
Alternative paths:	
Exception paths:	If DI can not start the application in 3) an exception occurs and the basic course of events is aborted. In this case this means DI skips the current application.
Triggers:	DI is started..
Pre-conditions:	
Post-conditions:	
Date:	26.09.2004
Author:	Jan Ove S. Olsen

Table 18.8: DI-UC-18.8

Use case name:	**Create group of tags.**
ID:	DI-UC-18.9
Priority:	M
Iteration:	Filled
Summary:	DI must support creation of groups of tags.
Accompanying requirements:	DI-9
Basic course of events:	1. DI is told directly by APP API that a group consisting of a given set of tags needs to be created. 2. DI checks the tags and finds out which tags belong on the same servers. 3. DI creates sub-groups with all the tags on the same servers in 2). 4. DI creates a new super-group of all the sub-groups in 3. 5. 3) and 4) are saved to database.
Alternative paths:	
Exception paths:	• 2a. If DI does not recognize the tags, and therefore can not figure out which server(s) they are located on, an exception occurs and the basic course of events is aborted. • When the basic course of events is aborted, we return an error code to the entity that started it.
Triggers:	An APP tells DI through APP API that a new group must be created.
Pre-conditions:	DI is running.
Post-conditions:	
Date:	26.09.2004
Author:	Jan Ove S. Olsen

Table 18.9: DI-UC-18.9

Use case name:	**Delete group of tags.**
ID:	DI-UC-18.10
Priority:	M
Iteration:	Filled
Summary:	DI must support creation of groups of tags.
Accompanying requirements:	DI-10
Basic course of events:	1. DI is told directly by APP API that a given group must be deleted. 2. DI checks the group and finds out which sub-groups it consists of. 3. DI deletes stops all reading from the sub-groups from 2) and deletes the group from the database. 4. DI deletes the given group. 5. 3) and 4) are saved to database.
Alternative paths:	
Exception paths:	• 2a. If DI does not recognize the group, an exception occurs and the basic course of events is aborted. • When the basic course of events is aborted, we return an error code to the entity that started it.
Triggers:	An APP tells DI through APP API that a new group must be created.
Pre-conditions:	DI is running.
Post-conditions:	
Date:	26.09.2004
Author:	Jan Ove S. Olsen

Table 18.10: DI-UC-18.10

Use case name:	**Browse tags**
ID:	DI-UC-18.11
Priority:	M
Iteration:	Filled
Summary:	DI must be able to browse all tags on connected OPC servers.
Accompanying requirements:	DI-11
Basic course of events:	1. DI is told directly from APP API to list all tags for all available servers. 2. DI connects to all the servers it needs. 3. DI contacts server(s) and request a list of all the tags available. 4. DI returns this list of tags to the entity that requested it.
Alternative paths:	• 1a. DI is given the address of a particular server.
Exception paths:	2a. DI can not contact a server • 2a1. It drops the tags on that server. If alternative path 1a) was taken or none of the servers from 1 can be contacted, an exception is raised at this point and the basic course of events are halted.
Triggers:	APP API requests all available tags.
Pre-conditions:	DI is running.
Post-conditions:	
Date:	26.09.2004
Author:	Jan Ove S. Olsen

Table 18.11: DI-UC-18.11

18.3 Requirements for the APP API

18.3.1 Graphical overall use case for APP API

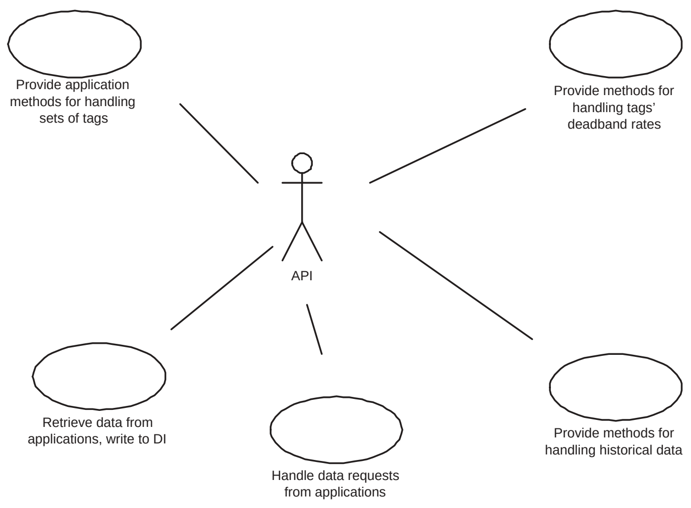

Provide application methods for handling sets of tags
Provide methods for handling tags' deadband rates
API
Retrieve data from applications, write to DI
Handle data requests from applications
Provide methods for handling historical data

Figure 18.2: Overall use case for API

18.3.2 Functional requirements for the APP API

Table 18.12 is a list of the requirements on the API module. Textual use cases follow for each of the requirements.

ID	Requirement	Priority	Use case
	Handling sets		
API-1	The API shall provide a method for building sets of tags.	H	API-UC-18.13
API-2	The API shall provide a method for adding tags to a set.	H	API-UC-18.14
API-3	The API shall provide a method for removing tags from a set.	M	API-UC-18.15
API-4	The API shall provide a method for splicing sets.	M	API-UC-18.16
API-5	The API shall provide a method for deleting a set.	M	API-UC-18.17
	Handling deadband		
API-6	The API shall provide a method for setting the limit of a variables change before an update-message is sent (deadband variable).	M	API-UC-18.18
	Read / write data		
API-7	The API shall provide a method for subscribing to a set of tags.	H	API-UC-18.19
API-8	The API shall provide a method for terminate the subscription of tags.	M	API-UC-18.20
API-9	The API shall provide a method for polling a set of tags.	H	API-UC-18.21
API-10	The API shall provide a method for writing a set of tags back to the DI.	H	API-UC-18.22
	Historical data		
API-11	The API shall provide a method for reading series of historical tags from the DI.	H	API-UC-18.23
	Configuration		
API-12	The API shall provide a method for reading configuration from the DI.	L	API-UC-18.24
API-13	The API shall provide a method for writing configuration to the DI.	L	API-UC-18.25

Table 18.12: API requirement list

18.3.3 Textual use cases for the APP API

Use case name:	**Define sets (of tags)**
ID:	API-UC-18.13
Priority:	H
Iteration:	Filled
Summary:	The API shall provide a method for building sets of tags
Accompanying requirements:	API-1
Basic course of events:	1. The application requests the API to create a set-object 2. The API returns the ID of the set-object
Alternative paths:	None
Exception paths:	In step 2, the DI may be out of resources, causing the call to fail
Triggers:	The application needs a set-object
Pre-conditions:	The application is connected to the DI through the API
Post-conditions:	The application has the ID of a set-object
Date:	24.09.2004
Author:	Øystein Ulseth

Table 18.13: API-UC-18.13

Use case name:	**Add tags to a set**
ID:	API-UC-18.14
Priority:	H
Iteration:	Filled
Summary:	The API shall provide a routine for adding tags to a set
Basic course of events:	1. The application compiles a list of tags 2. The applications calls the API. This call contains both the ID of the set and the list of tags.
Alternative paths:	None
Exception paths:	In step 2, if the ID provided is not valid, the call will fail. If so, the application first need to create a set as described in API-UC-18.13. Also, if the tags in the list are not available on the OPC-servers, the call will fail.
Triggers:	The application needs to add tags to a set
Pre-conditions:	The set must exist and the tags in the list must be available on the OPC-servers
Post-conditions:	New tags are added to the set
Date:	24.09.2004
Author:	Øystein Ulseth

Table 18.14: API-UC-18.14

Use case name:	**Remove tags from a set**
ID:	API-UC-18.15
Priority:	M
Iteration:	Filled
Summary:	The API shall provide a routine for removing tags from a set (equivalent to adding tags as described in API-UC-18.14).
Accompanying requirements:	API-3
Basic course of events:	1. The application compiles a list of tags 2. The application calls the API. The call provides both the ID of the set and the list.
Alternative paths:	None.
Exception paths:	See API-UC-18.14.
Triggers:	The application needs to remove tags from a set
Pre-conditions:	See API-UC-18.14.
Post-conditions:	Tags are removed from the set
Date:	24.09.2004
Author:	Øystein Ulseth

Table 18.15: API-UC-18.15

Use case name:	**Splice sets**
ID:	API-UC-18.16
Priority:	M
Iteration:	Filled
Summary:	The API shall provide a method for splicing two sets
Accompanying requirements:	API-4
Basic course of events:	1. The application calls the API and provides two sets 2. The API returns a new set with the following properties: • The new set is an union of the two sets • If tags intersect, only the most up to date are members of the new set.
Alternative paths:	None
Exception paths:	None
Triggers:	The application needs to splice two sets. Splicing the original set and the set returned in the subscription-event will create a complete set which is as up to date as possible.
Pre-conditions:	None
Post-conditions:	A new set which is the union of the two sets.
Date:	24.09.2004
Author:	Øystein Ulseth

Table 18.16: API-UC-18.16

Use case name:	**Delete a set**
ID:	API-UC-18.17
Priority:	M
Iteration:	Filled
Summary:	The API shall provide a routine for deleting a set
Accompanying requirements:	API-5
Basic course of events:	1. The application calls the API providing the ID of the set 2. The call returns
Alternative paths:	None.
Exception paths:	In step 2, if the ID is not valid, the call fails
Triggers:	The application need to delete a set, this might free resources in the DI.
Pre-conditions:	The set exists.
Post-conditions:	The set is deleted and the ID is no longer valid.
Date:	24.09.2004
Author:	Øystein Ulseth

Table 18.17: API-UC-18.17

Use case name:	**Setting the deadband variable**
ID:	API-UC-18.18
Priority:	M
Iteration:	Filled
Summary:	The deadband-variable is set on tags and not on the entire set
Accompanying requirements:	API-6
Basic course of events:	1. The application set the deadband variable on a tag
Alternative paths:	None
Exception paths:	None
Triggers:	None
Pre-conditions:	None
Post-conditions:	The variable is given a value (a percentage)
Date:	24.09.2004
Author:	Øystein Ulseth

Table 18.18: API-UC-18.18

Use case name:	**Subscribe to a set of tags**
ID:	API-UC-18.19
Priority:	H
Iteration:	Filled
Summary:	The API shall provide a method for subscribing to a set of tags at a given rate
Accompanying requirements:	API-7
Basic course of events:	1. The application calls the API. The call provides the ID of the set and the rate at which the set is subscribed. 2. In accordance to the rate, the API fires events in the application. The event contains only the tags which have changed significantly, consequently a subset of the subscribed set.
Alternative paths:	In step 2, if none of the tags in the set have changed significantly (no event is fired OR an empty set is returned).
Exception paths:	If the ID of the set does not exist, the call will fail.
Triggers:	The application needs to receive tags at regular time-intervals
Pre-conditions:	The ID of the set must be valid.
Post-conditions:	The API will fire events on the application at regular time-intervals.
Date:	24.09.2004
Author:	Øystein

Table 18.19: API-UC-18.19

Use case name:	**Terminate the subcription of tags**
ID:	API-UC-18.20
Priority:	M
Iteration:	Filled
Summary:	The API shall provide a routine for terminating the subscription of a set of tags.
Accompanying requirements:	API-8
Basic course of events:	1. The application calls the API. The call provides the ID of the set.
Alternative paths:	None
Exception paths:	• In step 1, if the ID is not valid, the call fails. • In step 1, if the set is not subscribed to, the call is ignored.
Triggers:	The application needs to terminate the subscription. This will cause the DI not to fire events in the application for this specific set of tags (eg. not pass on data from this set to the application, even though the data still might be received and passed on to other applications).
Pre-conditions:	The set must exists and the set must be subscribed to.
Post-conditions:	The subscription is terminated.
Date:	27.09.2004
Author:	Øystein Ulseth

Table 18.20: API-UC-18.20

Use case name:	**Synchronous poll of a set of tags**
ID:	API-UC-18.21
Priority:	M/L
Iteration:	Filled
Summary:	The API shall provide a method for polling a set of tags synchronously
Accompanying requirements:	API-9
Basic course of events:	1. The application calls the API. The call provides the ID of the set. 2. The API gathers data and returns the set. The control of the caller is not released until the set is returned.
Alternative paths:	The data may not exist, causing the call to time out.
Exception paths:	In step 1, if the ID of the set is not valid, the call fails.
Triggers:	The application needs a set of data and is willing to wait until the set is gathered.
Pre-conditions:	The set must exist.
Post-conditions:	The most up to date set is returned to the application.
Date:	27.09.2004
Author:	Øystein Ulseth

Table 18.21: API-UC-18.21

Use case name:	**Write data back to the DI**
ID:	API-UC-18.22
Priority:	H
Iteration:	Filled
Summary:	The API shall provide a function for writing a set of tags back to the DI. The write is synchronous.
Accompanying requirements:	API-10
Basic course of events:	1. The application calls the API with the set as argument. 2. The DI stores the data as tags. These tags will be available for other applications.
Alternative paths:	None
Exception paths:	None
Triggers:	The applications needs to save data for later use or make the data available for other applications.
Pre-conditions:	The set must be compiled.
Post-conditions:	The set is stored in the DI
Date:	27.09.2004
Author:	Øystein Ulseth

Table 18.22: API-UC-18.22

Use case name:	**Read historical data**
ID:	API-UC-18.23
Priority:	H
Iteration:	Filled
Summary:	The API shall provide a function for reading series of historical tags from the DI.
Accompanying requirements:	API-11
Basic course of events:	1. The application calls the API with the desired set ID (API-UC-18.13), desired sample rate/interval of the tags, and time lag 'from' and 'to' as arguments. 2. The API returns the desired sets.
Alternative paths:	None
Exception paths:	Some or all of the data may not exist. If some of the desired tags are not available, the call will fail.
Triggers:	The application needs historical data.
Pre-conditions:	The set must be compiled.
Post-conditions:	The series of sets are returned to the application.
Date:	27.09.2004
Author:	Kristoffer Stenersen

Table 18.23: API-UC-18.23

Use case name:	**Read configuration**
ID:	API-UC-18.24
Priority:	L
Iteration:	Filled
Summary:	The API shall provide functions for storing application configurations.
Accompanying requirements:	API-12
Basic course of events:	1. The application calls the API, requesting configurations. The application has to identify itself to the API, so that the right configuration can be returned. 2. The API returns the configuration to the application.
Alternative paths:	None
Exception paths:	The API does not recognize the application's identification. The call will fail.
Triggers:	The application needs its configuration information.
Pre-conditions:	The specified configuration must exist.
Post-conditions:	The configurations are returned to the application.
Date:	27.09.2004
Author:	Kristoffer Stenersen

Table 18.24: API-UC-18.24

Use case name:	**Write configuration**
ID:	API-UC-18.25
Priority:	L
Iteration:	Filled
Summary:	The API shall provide functions for storing application configurations.
Accompanying requirements:	API-13
Basic course of events:	1. The application calls the API, requesting a configuration write. The application identifies itself to the API, so that any old configurations will be updated. 2. If the configuration identification is unknown, the API creates a new configuration record for the specified configuration.
Alternative paths:	None
Exception paths:	None.
Triggers:	The application wants to store its configuration information.
Pre-conditions:	None.
Post-conditions:	The configurations are stored.
Date:	27.09.2004
Author:	Kristoffer Stenersen

Table 18.25: API-UC-18.25

18.4 REQUIREMENTS FOR WEB

18.4.1 Graphical overall use case for WEB

The graphical use case diagram in figure 18.3 shows the main actors and tasks within the WEB part of the system.

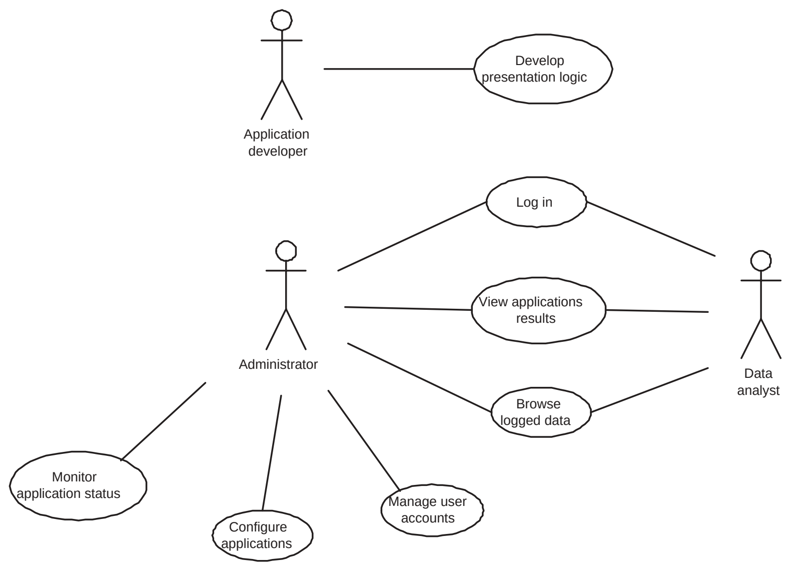


Figure 18.3: Overall use case for WEB

18.4.2 Functional requirements for the web portal

The functional requirements for the web portal are listed in table 18.26.

ID	Requirement	Priority	Use case
	General requirements		
WEB-1	All data series logged from OPC servers and data series written by applications shall be available through the web portal	H	
WEB-2	Every application shall be able to present its data on a set of customizable web pages	H	WEB-UC-2
WEB-3	When viewing data from applications operating on real time data, the user shall be presented with the newest data available at the time of his request.	L	
WEB-4	The web portal shall support user authorization, and offer customized functionality based on user groups	M	WEB-UC-1
WEB-5	The web portal shall support application status monitoring for administrators	L	WEB-UC-6
WEB-6	The web portal shall be able to run on a computer physically separated from the DI	L	
	Data presentation		
WEB-7	Users shall be able to navigate through application web pages easily, using some sort of tree structure.	L	WEB-UC-2
WEB-8	Users shall be able to browse all data available at the DI.	H	WEB-UC-3
WEB-9	When browsing data series, users shall be able to choose between table view and graphical time plot of the data	M	WEB-UC-3
	Configuration		
WEB-10	The web portal shall allow configuration of the set of custom web pages for each application. For each application, an arbitrary number of pages may be created.	H	WEB-UC-3 WEB-UC-4
WEB-11	The web portal shall support user administration	L	WEB-UC-5

Table 18.26: Functional requirements for WEB

18.4.3 Use cases for common user tasks

Textual use case diagrams for the following user tasks concerning WEB are listed in table 18.27 to table 18.32:

- WEB-UC-1 Authorize users
- WEB-UC-2 View results outputted by applications
- WEB-UC-3 Browse data series logged by DI
- WEB-UC-4 Configure applications
- WEB-UC-5 Manage user accounts
- WEB-UC-6 Monitor application status

Use case name:	**Authorize user**
ID:	WEB-UC-1
Priority:	Medium
Iteration:	Filled
Summary:	The web portal shall allow user to log in.
Accompanying requirements:	WEB-4
Basic course of events:	1. The web portal redirects the user to a login page, prompting the user for his username and password. 2. The user supplies his username and password. 3. User presses ”Log in” in the web interface. 4. The web portal accesses the user database through the web API. 5. The web portal receives a notice that the user has been logged in, and additional information about the user. 6. The web portal displays a start page giving an overview of the available applications.
Alternative paths:	6a. User is an administrator 6a1. The web portal displays a control panel allowing the user to choose between the available administrator tasks.
Exception paths:	4a. DI is offline • 4a1. The web portal recognizes the error, and displays an error message to the user. • 4a2. The interaction is aborted. 5a. User has supplied wrong user name or password • 5a1. The web portal recognizes the error, and refuses to log in the user • 5a2. The user is redirected to the login page, and an error message is displayed
Triggers:	The user accesses the web portal through a web browser.
Pre-conditions:	-
Post-conditions:	The web portal has authorized the user. User is forwarded to a page displaying the available applications.
Date:	23.09.2004
Author:	Hans Olaf Borch

Table 18.27: WEB-UC-1

Use case name:	**View results outputted by applications**
ID:	WEB-UC-2
Priority:	High
Iteration:	Filled
Summary:	The web portal shall allow users to view the data series published by all applications that have a web page.
Accompanying requirements:	WEB-2,WEB-7
Basic course of events:	1. The user selects the applications he wants to investigate further 2. The web portal displays a list of the available results from that application 3. The user selects results he wants to view, and is forwarded to the correct web page 4. The web portal gets the data series the selected application is set to publish 5. The web portal displays the data in the way specified by application programmers that have customized the web page 6. The user views the output of the selected application
Alternative paths:	The user can at all times change the application to be displayed in some kind of tree view. This will repeat the interaction from step 2 onwards.
Exception paths:	2a. DI is offline • 2a1. The web portal recognizes the error, and displays an error message to the user • 2a2. The interaction is aborted. 5a. The web page has not yet been customized • 5a1. The user is presented with for instance an ”under construction” page.
Triggers:	-
Pre-conditions:	The user is logged on to the system as a regular user (WEB-UC-1).
Post-conditions:	The web portal presented the user with the requested results.
Date:	23.09.2004
Author:	Hans Olaf Borch

Table 18.28: WEB-UC-2

Use case name:	**Browse data series logged by DI**
ID:	WEB-UC-3
Priority:	High
Iteration:	Filled
Summary:	The web portal shall allow users to view all data series logged by the DI.
Accompanying requirements:	WEB-8, WEB-9, WEB-10
Basic course of events:	1. The web portal gets the list of data series available for browsing 2. The user selects which data series he wants to browse 3. The user selects how the data series should be displayed (table or time plot) 4. The web portal gets the data series the selected application is set to publish 5. The web portal displays the data in the way specified by the user
Alternative paths:	-
Exception paths:	1a. DI is offline • 1a1. The web portal recognizes the error, and displays an error message to the user • 1a2. The interaction is aborted.
Triggers:	The user selects "browse data series".
Pre-conditions:	The user is logged on to the system.
Post-conditions:	The web portal presents the user with the requested data.
Date:	23.09.2004
Author:	Hans Olaf Borch

Table 18.29: WEB-UC-3

Use case name:	**Configure applications**
ID:	WEB-UC-4
Priority:	Medium
Iteration:	Filled
Summary:	The web portal shall allow administrators and application designers to configure applications and their web pages.
Accompanying requirements:	WEB-10
Basic course of events:	1. The administrator selects the application he wants to configure 2. The web portal displays the existing configuration of the chosen application 3. The user is allowed to edit which data series the application outputs and publishes 4. The user is allowed to edit whether the applications results are to be available through the web portal 5. The user is allowed to edit the web pages connected to the applications 6. The user submits his changes 7. The web portal reads the user input, and makes the appropriate changes to the configurations 8. The web portal receives a notice when the changes are saved 9. The web portal displays a confirmation to the user
Alternative paths:	-
Exception paths:	2a. DI is offline • 2a1. The web portal recognizes the error, and displays an error message to the user • 2a2. The interaction is aborted.
Triggers:	The user selects application configuration in the control panel.
Pre-conditions:	The user is logged in as an administrator (WEB-UC-1).
Post-conditions:	Changes to application configurations have been applied on the web portal.
Date:	23.09.2004
Author:	Hans Olaf Borch

Table 18.30: WEB-UC-4

Use case name:	**Manage user accounts**
ID:	WEB-UC-5
Priority:	Low
Iteration:	Filled
Summary:	The web portal shall allow administrators to manage user accounts.
Accompanying requirements:	WEB-11
Basic course of events:	1. User selects the user group he wishes to configure 2. The web portal displays the existing configuration of the chosen user group 3. The user is allowed to edit members of the group 4. The user is allowed to edit which applications the group is permitted to access 5. The user submits his changes 6. The web portal reads the user input, and makes the appropriate changes to the configurations 7. The web portal receives a notice when the changes are saved 8. The web portal displays a confirmation to the user
Alternative paths:	-
Exception paths:	2a. DI is offline • 2a1. The web portal recognizes the error, and displays an error message to the user • 2a2. The interaction is aborted.
Triggers:	The user selects user administration in the control panel.
Pre-conditions:	The user is logged in as an administrator (WEB-UC-1).
Post-conditions:	Changes to users and user groups have been applied on the web portal.
Date:	27.09.2004
Author:	Hans Olaf Borch

Table 18.31: WEB-UC-5

Use case name:	**Monitor application status**
ID:	WEB-UC-6
Priority:	Low
Iteration:	Filled
Summary:	The web portal shall administrators to view the status of all applications.
Accompanying requirements:	WEB-5
Basic course of events:	1. The web portal requests status of all the applications from the DI. 2. The web portal receives a list of status for all applications 3. The administrator is presented with a simple list of status for each of the applications
Alternative paths:	-
Exception paths:	2a. DI is offline • 2a1. The web portal recognizes the error, and displays an error message to the user • 2a2. The interaction is aborted.
Triggers:	The user selects application monitoring in the control panel.
Pre-conditions:	The user is logged in as an administrator (WEB-UC-1).
Post-conditions:	-
Date:	27.09.2004
Author:	Hans Olaf Borch

Table 18.32: WEB-UC-6

18.5 Design constraints

18.5.1 Operating system

Table 18.33 lists the constraints we have regarding operating systems.

OPS-1	The final product must run on a Microsoft Windows server.	H

Table 18.33: Operating system requirements

18.5.2 Hardware

Table 18.34 lists the minimum requirements for the hardware the final product will run on.

HW-1	The final product must run on a Pentium 4 class PC with 512MB of ram and 500GB free disk space.	H

Table 18.34: Hardware requirements

18.5.3 Programming platform

A programming platform is defined as the combination of a programming language, tools for easing development in the language and an extensive standard library of already implemented functionality. Table 18.35 lists the requirements a programming platform should fulfil.

ID	**Requirement**	**Priority**
PRO-1	The programming platform must be available for Microsoft Windows.	H
PRO-2	The programming platform must support communication with OPC servers through DCOM.	H
PRO-3	The programming platform must support accessing a relational database through SQL.	H
PRO-4	The programming platform must be high level, to enable a more efficient development experience.	M

Table 18.35: Programming platform requirements

18.5.4 Standards compliance

The system needs to be in compliance with the standards in table 18.36.

ID	Requirement	Priority
STD-1	The finished product must communicate with OPC server through the OPC DA 2.0 or OPC DA 3.0 standard.	H

Table 18.36: Standards compliance

18.6 Database requirements

The database requirements are listed in table 18.37

ID	Requirement	Priority
DB-1	The database must remain consistent at all times.	L
DB-2	The database must be able to handle writing a lot of data tags. For instance be able to log data from 500 tags being logged at a 1 second rate, 500 tags at a 10 second rate and 500 at a 10 minute rate.	H
DB-3	The performance of the database must not decrease noticeably when it contains large amounts of data. For instance the tags mentioned above for 6 months	M
DB-4	The database must support the Structured Query Language (SQL).	M
DB-5	The database must be able to run physically separated from the rest of our application	L

Table 18.37: Database requirements

18.7 Software system attributes

This section lists the various attributes the finished system should have and the requirements that has to be met for this to happen.

18.7.1 Reliability

The customer must be able to trust the system to do the job it is intended to do. Figure 18.38 lists all the demands that have to be fulfilled for this to be true.

ID	Requirement	Priority
REL-1	The finished product is delivered with installation instructions. These must be followed exactly.	M
REL-2	The finished product must fulfill all the security requirements in 18.40.	M
REL-3	The server that the product is installed on, must meet the hardware requirements listed in 18.34.	M
REL-4	The server that the product is installed to, must have the other software products mentioned in 18.33 pre-installed and verified as working.	M
REL-5	The servers the finished solution will interact with (OPC-servers) MUST be completely compatible with the OPC DA 2.0 or OPC DA 3.0 standard.	H

Table 18.38: Reliability requirements

18.7.2 Availability

The system, and all of its functionality, should be available all, or close to all, the time. This attribute of the system is called its availability. Figure 18.39 lists the demand that has to be fulfilled for this to happen.

ID	Requirement	Priority
AVA-1	In case of a system crash on the server, the finished product must be able to function again after a reboot of the operating system.	M
AVA-2	Should the product itself crash, it must be able to gracefully continue where it left of as soon as it is restarted.	M

Table 18.39: Availability requirements

18.7.3 Security

The customer should be able to feel safe about the security of data in the system. Table 18.40 lists all the demands that have to be fulfilled to call the system secure.

ID	Requirement	Priority
SEC-1	The finished solution must be delivered with source code, so that all code can be inspected by customer.	M
SEC-2	The finished solution should be delivered with documentation on how data is stored and accessed.	L
SEC-3	The finished solution shall not give more extended access to data than the level of access defined for each user by an administrator.	L
SEC-4	The finished solution must make use of a relational database that makes sure only authorized users can access its data.	M

Table 18.40: Security requirements

18.7.4 Maintainability

The product must be easy to maintain in the future. The groundwork for this is set during the construction and implementation of the product. Table 18.41 lists the demands put on the system and the development process in order to produce an easily maintained product.

ID	Requirement	Priority
MAI-1	All code should be written according to a common standard [7].	M
MAI-2	Code shall be commented while it is being produced, not as an afterthought.	L
MAI-3	All function, variable and parameter names should be in English.	M
MAI-4	All code should be in a high-level language.	H

Table 18.41: Maintainability requirements

18.7.5 Portability

The demands in table 18.42 must be fulfilled to call this system as portable as the customer wants.

ID	**Requirement**	**Priority**
POR-1	The finished product must be able to run on server that run operating systems from Microsoft.	H

Table 18.42: Portability requirements

Chapter 19 Estimation

19.1 The need for estimation

Estimation of hours needed is required for most projects in order to suceed with delivering the product on time. We have already done a (somewhat rough) estimation for the project as a whole vizualised in the Gantt chart presented in B.1. The requirements listed in the previous chapter calls however for a more precise estimation of hours needed in the design and implementation phase. The purpose of the estimation is to see if there is enough time to design and implement the requirements. If the estimated time exceeds the time available for these phases [1], low priority requirements will be discarded until hours needed equals hours estimated. We have planned to use a total of 702 hours for the design and implemention phases.

19.2 Estimation method

Our first approach was to use the Project Estimation Method developed by Bente Anda [5]. This method uses the Use Cases to estimate the time required for the two project phases design and implementation. The actual estimation is done by filling in a small number of values in a worksheet [4]. The method has sucessfully been applied to several projects, so we expected the method to also work well for our project.

The method suggested at total of 3045 hours needed for design and implementation. We suspected that that the method was overestimating person hours needed per use case point. To examine this suspection, we did prototype implementation on two of the requirements, namely the DI use case "Read from OPC server"(18.2), and the WEB use case "Authorize User" (18.27). When employing the method on "Read from OPC Server", the method suggested an estimate of 324 hours. Likewise, the "Authorize User" use case was estimated to require 365 hours. This contrasts with the actual hours needed on implementation: We used 10 hours on "Read from OPC server", and eight hours on "Authorize user".

A prototype implementation is often less time consuming than the actual implementation, so we expect the actual amount of hours spent to be higher than indicated by the prototyping. We expected the actual time needed to be somewhere in between what the Project Estimation Method suggested and what the prototyping indicated.

We settled for using the group members previous experiences, combined with the aforementioned observations to come up with an expert estimation for hours needed for the design and implementation phases.

19.3 Results

This section provides the results for both methods.

[1] An overview of hours in each phase is presented in B.1

19.3.1 Project Estimation Method

The Project Estimation Method suggested the the following amount of hours needed for the designing and implementation:

- DI: 1137 hours
- API: 528 hours
- WEB: 1380 hours

This sums up to 3045 hours.

19.3.2 Our own estimation

Estimated hours needed for each requirement part (18.2,18.3 and 18.4) are listed in the tables 19.1, 19.2 and 19.3.

ID	Requirement	Priority	Hours needed
	Data read / write		
DI-1	DI reads synchronous from OPC-server.	H	40
DI-2	DI start asynchronous continual read. (subscribes to a data tag)	H	40
DI-3	DI stops a continually asynchronous read. (A subscribed tag)	H	10
DI-4	DI pauses a continually asynchronous read. (A subscribed tag)	L	10
DI-5	DI continues a continually asynchronous read. (A subscribed tag)	L	10
DI-6	DI writes synchronous to OPC-server.	H	20
	Scheduling		
DI-7	DI schedules applications.	H	40
	Handling tags		
DI-8	Create group of tags.	M	20
DI-9	Delete group of tags.	M	20
DI-10	Browse tags.	L	20
	Total		220

Table 19.1: Estimation of the hours needed to fulfill the DI requirements

Each requirement sums up to:

- DI: 220 hours
- API: 160 hours
- WEB: 360 hours

Which, in turn, sums up 740 hours. We have planned to use 702 hours for the design and implemention phase, so the following requirements are skipped in order to fit hours spent into hours needed:

- Manage user accounts (from WEB, 18.31, WEB-11, estimated to take 20 hours)
- Browse tags (from DI, 18.11, estimated to take 20 hours)

These requirments will only be implemented if time perimts us to do so, but is in principle discarded.

ID	Requirement	Priority	Hours needed
	Handling sets		
API-1	The API shall provide a method for building sets of tags.	H	20
API-2	The API shall provide a method for adding tags to a set.	H	10
API-3	The API shall provide a method for removing tags from a set.	M	10
API-4	The API shall provide a method for splicing sets.	M	20
API-5	The API shall provide a method for deleting a set.	M	10
	Handling deadband		
API-6	The API shall provide a method for setting the limit of a variables change before an update-message is sent (deadband variable).	M	10
	Read / write data		
API-7	The API shall provide a method for subscribing to a set of tags.	H	10
API-8	The API shall provide a method for terminate the subscription of tags.	M	10
API-9	The API shall provide a method for polling a set of tags.	H	10
API-10	The API shall provide a method for writing a set of tags back to the DI.	H	10
	Historical data		
API-11	The API shall provide a method for retrieve information about historical data.	M	10
API-12	The API shall provide a method for reading series of historical tags from the DI.	H	10
	Configuration		
API-13	The API shall provide a method for reading configuration from the DI.	L	10
API-14	The API shall provide a method for writing configuration to the DI.	L	10
	Total		160

Table 19.2: Estimation of the hours needed to fulfill the API requirements

ID	Requirement	Priority	Hours needed
	General requirements		
WEB-1	All data series logged from OPC servers and data series written by applications shall be available through the web portal	H	15
WEB-2	Every application shall be able to present its data on a set of customizable web pages	H	35
WEB-3	When viewing data from applications operating on real time data, the user shall be presented with the newest data available at the time of his request.	L	10
WEB-4	The web portal shall support user authorization, and offer customized functionality based on user groups	M	20
WEB-5	The web portal shall support application status monitoring for administrators	L	20
WEB-6	The web portal shall be able to run on a computer physically separated from the DI	L	10
	Data presentation		
WEB-7	Users shall be able to navigate through application web pages easily, using some sort of tree structure.	L	20
WEB-8	Users shall be able to browse all data available at the DI.	H	50
WEB-9	When browsing data series, users shall be able to choose between table view and graphical time plot of the data	M	50
	Configuration		
WEB-10	The web portal shall allow configuration of the set of custom web pages for each application. For each application, an arbitrary number of pages may be created.	H	30
WEB-11	The web portal shall support user administration	L	20
	Framework		
Extra	To run, the WEB functions will need a comprehensive framework (underlying structure for navigating between pages and sharing user- and state-data).	H	80
	Total		360

Table 19.3: Estimation of the hours needed to fulfill the WEB requirements

Part IV

Software Design Description

Chapter 20 Introduction

20.1 Purpose

The design description makes a transition from the more abstract part III - "Requirements Specification", to a practicable specification. Therefore, part III is a prerequisite for this part. This part will result in a detailed, implementation-near description of the system and sub-systems.

20.2 Scope

This part will cover all aspects of building the system according to the requirements specified earlier. At this stage, we have reduced the number of top level modules. The APP API is no longer regarded as a top level module, as it fits naturally as an integrated part of the DI. That leaves us with two top level modules, namely DI and WEB. See figure 21.1 for clarification on this matter.

20.3 Definitions, acronyms and abbreviations

A number of definitions, acronyms and abbreviations will occur in the design description. The following lists explains each of these.

- **APP API**
  Interface provided by the DI. Applications will communicate with the DI through this.
- **DBC**
  Database Connector, entity for database access.
- **DC**
  Data Collector, entity for OPC-communication.
- **Design entity**
  An element/component of the design that is structurally and functionally distinct from other elements and is separately named and referenced.
- **Design view**
  A subset of design entity attribute information that is specifically suited to the needs of a software project activity.
- **DI**
  Data Interchanger, a top level module.
- **Module**
  A program unit that is discrete and identifiable with respect to compiling and loading with other units.
- **WEB**
  Web system, a top level module.
- **WEB API**
  Interfaces provided by the DI. The WEB-module will communicate to the DI through this.

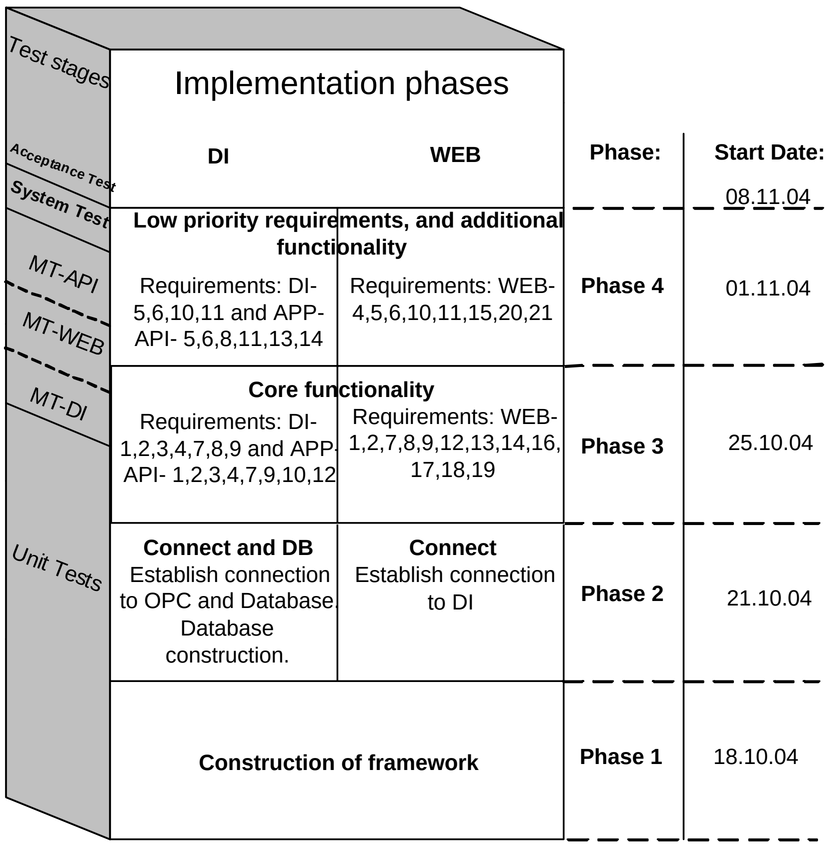

Test stages
Acceptance Test
System Test
MT-API
MT-WEB
MT-DI
Unit Tests
Implementation phases
DI
WEB
Phase:
Start Date:
08.11.04
Low priority requirements, and additional functionality
Requirements: DI-5,6,10,11 and APP-API- 5,6,8,11,13,14
Requirements: WEB-4,5,6,10,11,15,20,21
Phase 4
01.11.04
Core functionality
Requirements: DI-1,2,3,4,7,8,9 and APP-API- 1,2,3,4,7,9,10,12
Requirements: WEB-1,2,7,8,9,12,13,14,16,17,18,19
Phase 3
25.10.04
Connect and DB
Establish connection to OPC and Database. Database construction.
Connect
Establish connection to DI
Phase 2
21.10.04
Construction of framework
Phase 1
18.10.04

Figure 20.1: Implementation phases

20.4 References

This document is based on the IEEE standard 1016 ([15]). Also see IEEE 610.12, Standard Glossary of Software Engineering terminology, [17].

20.5 Overview

Chapter 21 deals with decomposition of the modules that EasyIT consists of. The purpose of this chapter is to sketch an overview of the systems modules. Chapter 22 describes inter-modular dependencies as well as other resources the modules depend on. Chapter 23 describes the interfaces. Finally, chapter 24 deals with detailed design of each module - objects attributes, processes and data.

20.6 Development

We have chosen a top-down approach to the development process. First we intend to construct the entire framework of the system, and in later phases gradually increase the functionality in accordance with the requirements. We will therefore have a running version of the system in every phase. This enables us to continuously evaluate the amount of time needed for the remaining implementation. Thus, re-scheduling and re-allocation of resources throughout the implementation is made possible. Top-down development also eases the incorporation of the test stages in the implementation phases. Modules running in order are a prerequisite for the module and integration tests.

20.6.1 Phase 1

The first phase will be construction of an overall framework for the system. First, this framework will consist of two independent modules, namely WEB and DI. **No specific requirements are realized in this phase.**

20.6.2 Phase 2

In the second phase connection between the two frameworks will be established. This connection will be implemented at this stage because of its necessity for the later implemented functions in the system. Connection from DI to OPC servers and Database will also be implemented in the current phase. After connection to database is established, this phase will also include construction of the database structure. **No specific requirements are realized in this phase.**

20.6.3 Phase 3

In the third development phase, selected core functionality for both WEB and DI will be constructed. Core functionality indicates that it is both vital for the general functionality of the system, and also is prioritized with High or Medium in the requirements specification.

WEB functionality implemented in Phase 3:

WEB-1 Make data series logged from OPC servers and data written by applications available through the web portal

WEB-2 Make functionality which enables application to present it's data on a set of web pages. Application developers shall be able to provide custom logic for displaying data.

WEB-4 Support for authentication of users, and customized functionality based on user groups.

WEB-8 Support browsing of all available data on DI, by user.

WEB-9 Support user browsing of data from DI through both table view and graphical time plot.

WEB-10 Support administrator configuration of the set of custom web pages for each application.

APP-API functionality implemented in Phase 3:

API-1 Building sets of tags

API-2 Adding tags to a set

API-3 Removing tags from a set

API-4 Splicing sets

API-7 Starting subscription of a set

API-9 Polling a set of tags

API-10 Writing back a set of tags to DI

API-12 Reading series of historical data

DI functionality implemented in Phase 3:

DI-1 Synchronous read from OPC

DI-2 Asynchronous read from OPC

DI-3 and DI-4 Start and stop of subscription from OPC

DI-7 Synchronous write to OPC

DI-8 Scheduling of applications

DI-9 Creation of tag groups

20.6.4 Phase 4

The final phase will be used for implementing additional functionality and correcting errors discovered in the module test stages. The requirements that have been scheduled for implementation in this phase are prioritized as Medium or Low.

WEB functionality implemented in Phase 4:

WEB-3 Support for guarantee of newest data when user browses real time data.

WEB-5 Support for application status monitoring by administrators.

WEB-7 Implement user navigation by a tree structure.

WEB-11 Support for user administration through the web portal.

APP-API functionality implemented in Phase 4:

API-5 Method for deletion of sets

API-6 Method for setting the deadband variable

API-8 Method for termination of tag subscription

API-11 Method for retrieval of information about historical data

API-13 and API-14 Method for read and write configuration data to DI

DI functionality implemented in Phase 4:

DI-5 and DI-6 Pause and continual of asynchronous read

DI-10 Delete groups of tags

DI-11 Browse tags

Chapter 21
Decomposition description

This chapter outlines the conceptual pieces of the system. In the real implementation they will probably be further decomposed. To see the complete physical implementation, see class diagram in Chapter 26. For each decomposed entity, a reference to the requirement specification is included. These references should provide complete traceability to the requirements specification.

21.1 Module decomposition

Figure 21.1 shows an overall architecture of the components of EasyIT. Easy IT has two main modules, the Data Interchanger (DI) and the Web-module. DI consists of four entities, namely *Application API*, *Web API*, *Database Connector* and *Data Collector*. The two APIs (Application and Web) share some functionality. These methods will be implemented in a separate entity, called *EasyIT Manager*. This entity is listed as an external entity. The Web-module also consists of four entities; the *Server Connector*, *Tag Browser*, *Configuration Tool* and *Application Monitor*.

21.1.1 DI - Data Interchanger

The DI module consists of three modules (Application API, Web API and Database Connector) and one sub-program (Data Collector). Together, these entities take care of data collecting and data storing.

Attribute	**Description**
ID:	Data Collector
Type:	Sub-program
Purpose:	The purpose of the DC is to control the flow of data to and from OPC. Related requirements: DI1 to DI7 and DI9 to DI11, see 18.1
Function:	• un/subscribe to OPC-updates • minimize the load on OPC-servers • store received set in the database • read instant value from OPC • pass instant value on to application • write tags to OPC
Subordinates:	none

Table 21.1: Decomposition of Data Collector

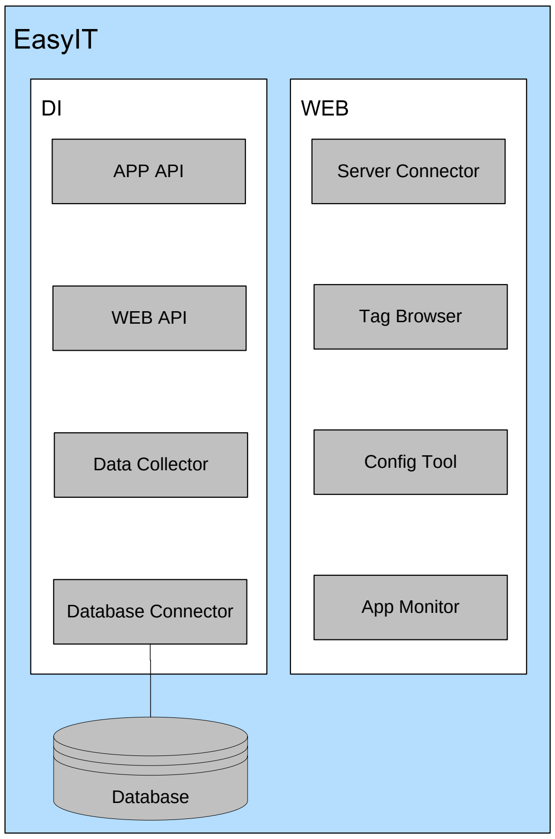


Figure 21.1: Overall figure of module decomposition

Attribute	**Description**
ID:	Database Connector
Type:	Module
Purpose:	This entity connects the database to any other entity which needs access to the database.
Function:	DBC connects other entities to the DB. With this approach, database access is reduced to one single point. The entity also implements connection pooling to DB in order to improve the overall performance of the system.
Subordinates:	none

Table 21.2: Decomposition of Database Connector

Attribute	**Description**
ID:	Application API
Type:	Module
Purpose:	This entity facilitates access to the DI for applications running on the system. Related requirements API-18.12
Function:	The Application API provides application-specific functionality to applications.
Subordinates:	none

Table 21.3: Decomposition of Application API

Attribute	**Description**
ID:	Web API
Type:	Module
Purpose:	The connection between WEB and DI needs a common interface. This is realized in the WEB API module. Related requirements WEB-18.26
Function:	Web API is the interface between DI and WEB. Web API provides functionality for Web such as tag browsing and application monitoring. Web API also handles user authentication and provides logic for presenting data on Web.
Subordinates:	none

Table 21.4: Decomposition of Web API

21.1.2 Web - Web Interface

This section describes the modules constituting the web interface (user interface) part of EasyIT. This part of the system form a flat system structure, the components involved are not depended on other WEB components. Thus, the entities are regarded as being on the same level in the system hierarchy.

Attribute	**Description**
ID:	ServerConnector
Type:	WEB-module
Purpose:	This module exists because WEB needs a uniform way to read data from the database, possibly located at another server. The ServerConnector was created to realize requirement WEB-1.
Function:	This entity establishes connection to the Web API located at the server. This connection is made available to all the web pages.
Subordinates:	none

Table 21.5: Decomposition of ServerConnector

Attribute	**Description**
ID:	TagBrowser
Type:	WEB-module
Purpose:	This module exists because users need to be able to view the data logged by DI. This is required by requirement WEB-8.
Function:	This module provides the users with at way to browse all data series logged by the DI. The data is made available either as a table or as a graphical time plot.
Subordinates:	none

Table 21.6: Decomposition of Tag Browser

Attribute	**Description**
ID:	ConfigTool
Type:	WEB-module
Purpose:	This module exists because administrators shall be able to set up and configure the web pages containing results written by applications. ConfigTool originates from the requirement WEB-10
Function:	This modules allows the administrator to set up whether an application shall present its data on the web portal. It should also allow the creation of empty skeletons for custom web pages associated with this application.
Subordinates:	none

Table 21.7: Decomposition of ConfigTool

Attribute	**Description**
ID:	AppMonitor
Type:	WEB-module
Purpose:	A way to monitor the status of applications is needed through the web interface. This is described in requirement WEB-5.
Function:	The AppMonitor module queries the DI for status on all applications. These data are presented to the user.
Subordinates:	none

Table 21.8: Decomposition of ServerConnector

21.1.3 External entities

Attribute	Description
ID:	Database
Type:	Data storage
Purpose:	We need to store data in a structured way. We do this in a relational database. Related requirements are API-11 to API-14 (18.12)
Function:	The entity gets data as input and stores them in tables. It can also fetch data from these tables and return this as output.
Subordinates:	Not applicable. The different modules of a modern database is beyond the scope of this document.

Table 21.9: Decomposition of Database

Attribute	Description
ID:	EasyIT Manager
Type:	Sub-program
Purpose:	We need a class to keep track of all instances, so they can communicate, i.e. launch subprograms and applications. Related requirements are DI-8, 18.1
Function:	Start the DC, make modules available to each other.
Subordinates:	none

Table 21.10: Decomposition EasyIT

21.2 Concurrent process decomposition

Figure 21.2 gives a conceptual overview of what physical processes will run when the EasyIT-system are launched. The subprocesses represent the launch of the prosess, not the process itself. EasyIT Manager spawns it's subprograms, and continue to act as a connection between them. Applications are started simultaneously, and control themselves regarding scheduling. Notice that the "APPS" process either can be absent (no apps registered) or multiple (more than one app registered).

21.3 Data decomposition

Figure 21.3 shows the internal representation of OPC-data. A series can contain zero or more sets. A set can be contained by zero or more series. A set contains zero or more tags. A tag can be contained by zero or more sets. All other data types used within the system are well known and defined in the API of the programming language.

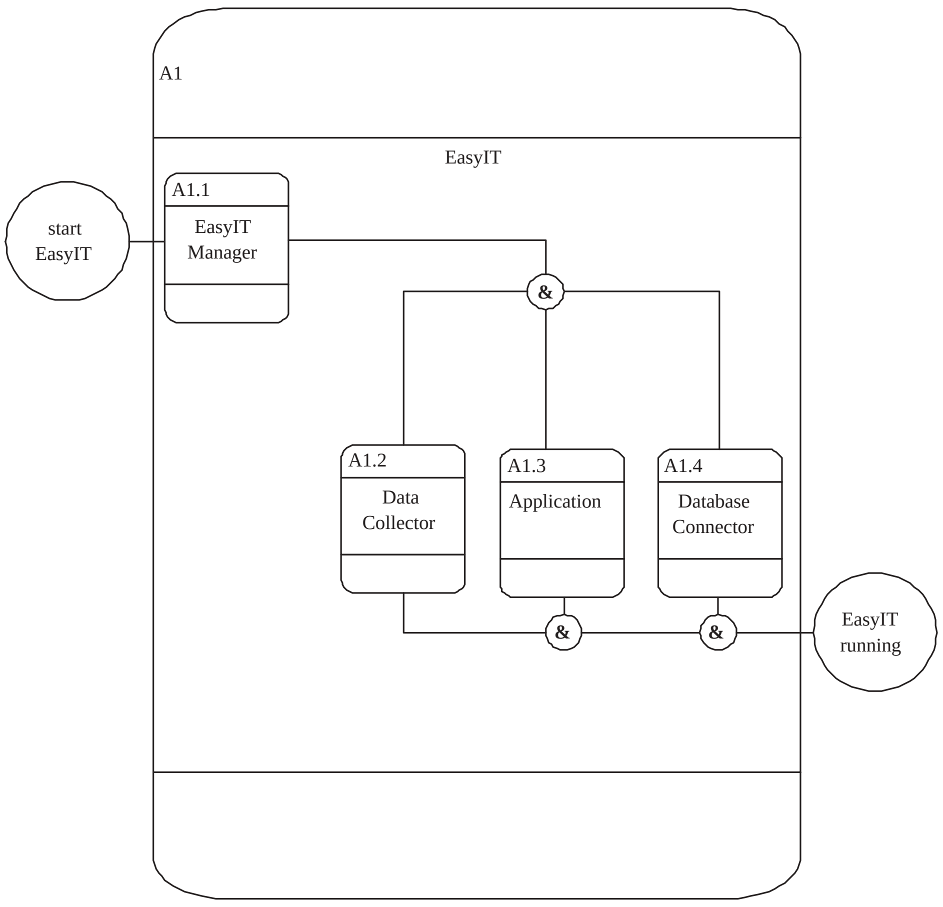


Figure 21.2: Processes spawned at startup

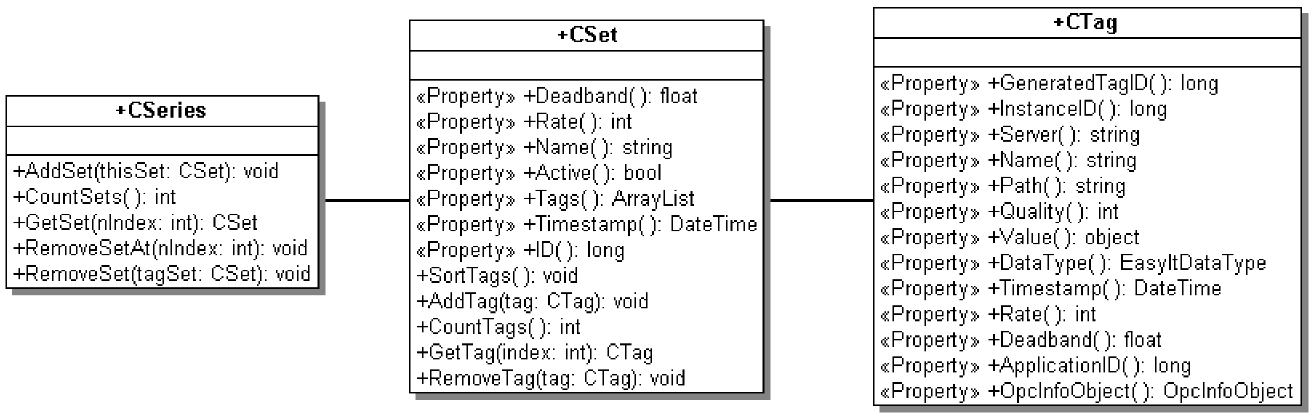

+CSeries
+AddSet(thisSet: CSet): void
+CountSets( ): int
+GetSet(nIndex: int): CSet
+RemoveSetAt(nIndex: int): void
+RemoveSet(tagSet: CSet): void
+CSet
«Property» +Deadband( ): float
«Property» +Rate( ): int
«Property» +Name( ): string
«Property» +Active( ): bool
«Property» +Tags( ): ArrayList
«Property» +Timestamp( ): DateTime
«Property» +ID( ): long
+SortTags( ): void
+AddTag(tag: CTag): void
+CountTags( ): int
+GetTag(index: int): CTag
+RemoveTag(tag: CTag): void
+CTag
«Property» +GeneratedTagID( ): long
«Property» +InstanceID( ): long
«Property» +Server( ): string
«Property» +Name( ): string
«Property» +Path( ): string
«Property» +Quality( ): int
«Property» +Value( ): object
«Property» +DataType( ): EasyItDataType
«Property» +Timestamp( ): DateTime
«Property» +Rate( ): int
«Property» +Deadband( ): float
«Property» +ApplicationID( ): long
«Property» +OpcInfoObject( ): OpcInfoObject

Figure 21.3: Representation of OPC-data

Chapter 22 Dependency description

This chapter describes entities and external resources needed to perform as described in the previous chapter. To illustrate this, we have used structural charts and UML sequence diagrams.

22.1 Intermodule dependencies

Data collecting and application execution are the most basic operations performed by the system. Data are stored and structured using three datatypes; tags, sets and series. These types are described in Figure 21.3. In brief, tags are used to store single data values. Further, a number of tags may be bundled up in sets. If a new level of structuring is needed, sets are bundled up as series. So, in an application running on the EasyIT-system, the programmer should group tags of interest to form a set. A trend curve is then easily represented as a series of sets. The set data type will have methods for adding and removing tags. Likewise, the series type will have functions for adding and removing sets.

SQL is a de facto standard for database queries. In *.Net* the *DataAdapter* object is suitable for reading the result of a SQL-query. Method calls are synchronous. Figure 22.1 illustrates inter-modular dependencies in the system. The arrow end refer to the module needed by the pointing module. The arrows to the EasyIT Manager illustrate that some modules need reference to instances of other modules managed by EasyIT. The arrows to ServerConnector illustrate communication with the EasyIT-module.

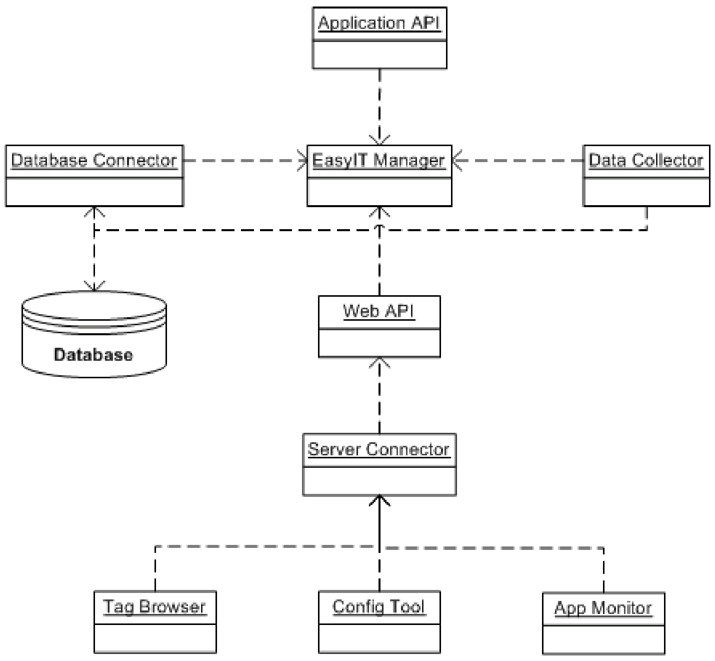


Figure 22.1: Intermodule dependencies

22.1.1 DI - Data Interchanger

This section will describe the dependencies between the modules in the decomposed *Data Interchanger* (DI). Each module and its dependencies are described in one table.

Attribute	**Description**
ID:	Data Collector
Type:	Sub-program
Purpose:	The purpose of the DC is to control the flow of data to and from OPC. Related requirements DI1 to DI7 and DI9 to DI11, see 18.1
Dependencies:	The DC requires the presence of the database connector.
Resources:	• a connected network with TCP/IP availability • an operating system with TCP/IP stack and network interface • enough RAM and processing power to handle all received tags • OPC servers

Table 22.1: Dependencies for Data Collector

Attribute	**Description**
ID:	Database Connector
Type:	Module
Purpose:	This entity is needed to interconnect the database with any other entity needing to store or retrieve data..
Dependencies:	Depends on DB, for access to the database.
Resources:	This entity uses no additional external resources.

Table 22.2: Dependencies for Database Connector

Attribute	**Description**
ID:	Application API
Type:	Module
Purpose:	The system needs a way to expose it's functionality to applications accessing EasyIT. Related requirements API-18.12
Dependencies:	DBC to connect to the database to read and write data for applications (Tags/Sets/Series and configuration information).
Resources:	.NET Remoting to communicate with applications.

Table 22.3: Dependencies for Application API

Attribute	**Description**
ID:	WEB API
Type:	Module
Purpose:	The connection between WEB and DI needs a common interface. This is realized in the WEB API module. Related requirements WEB-*: 18.26
Dependencies:	DBC to connect to the database to read and write for WEB.
Resources:	This entity uses no additional external resources.

Table 22.4: Dependencies for WEB API

Sequence diagrams for basic DI functionality

The following sequence diagrams sketches some of the basic DI functionality. Each diagram indicates what entities take part in the operation and the flow of data between these entities. Please note that the level of details is highest on the leftmost part of the diagrams. The rightmost part deals with implementation issues not yet explored. The methods called in the sequence diagrams are described in more detail in the next chapter, *Interface description*.

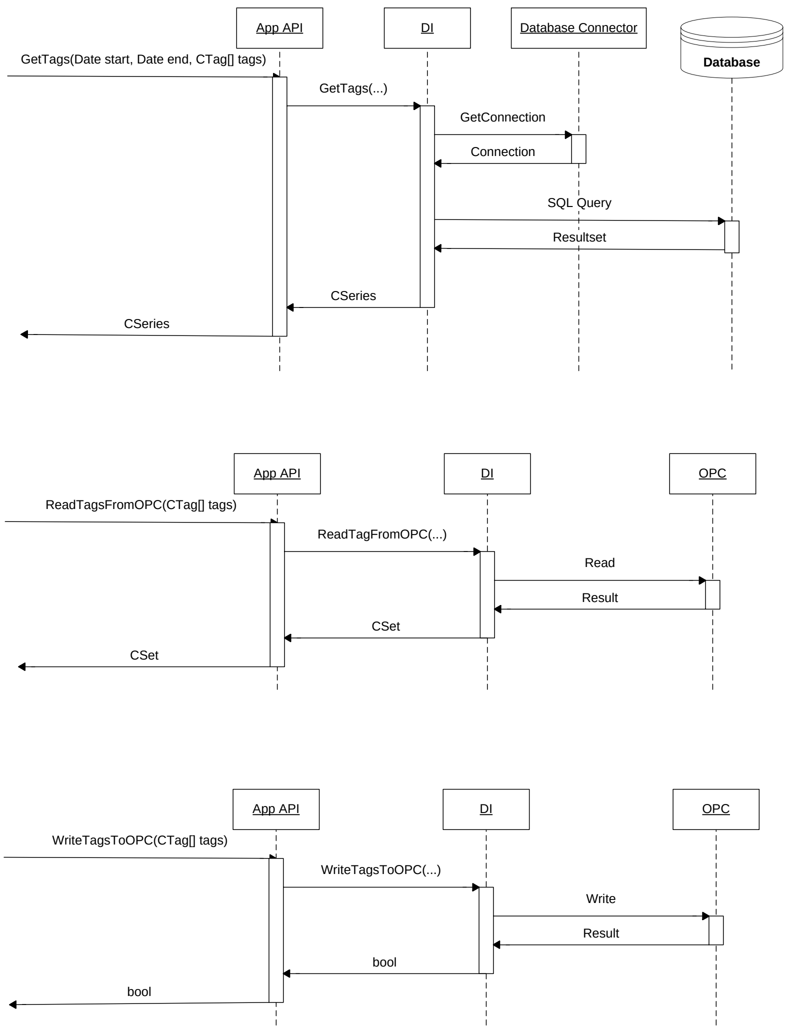


Figure 22.2: Reading and writing synchronous from/to OPC

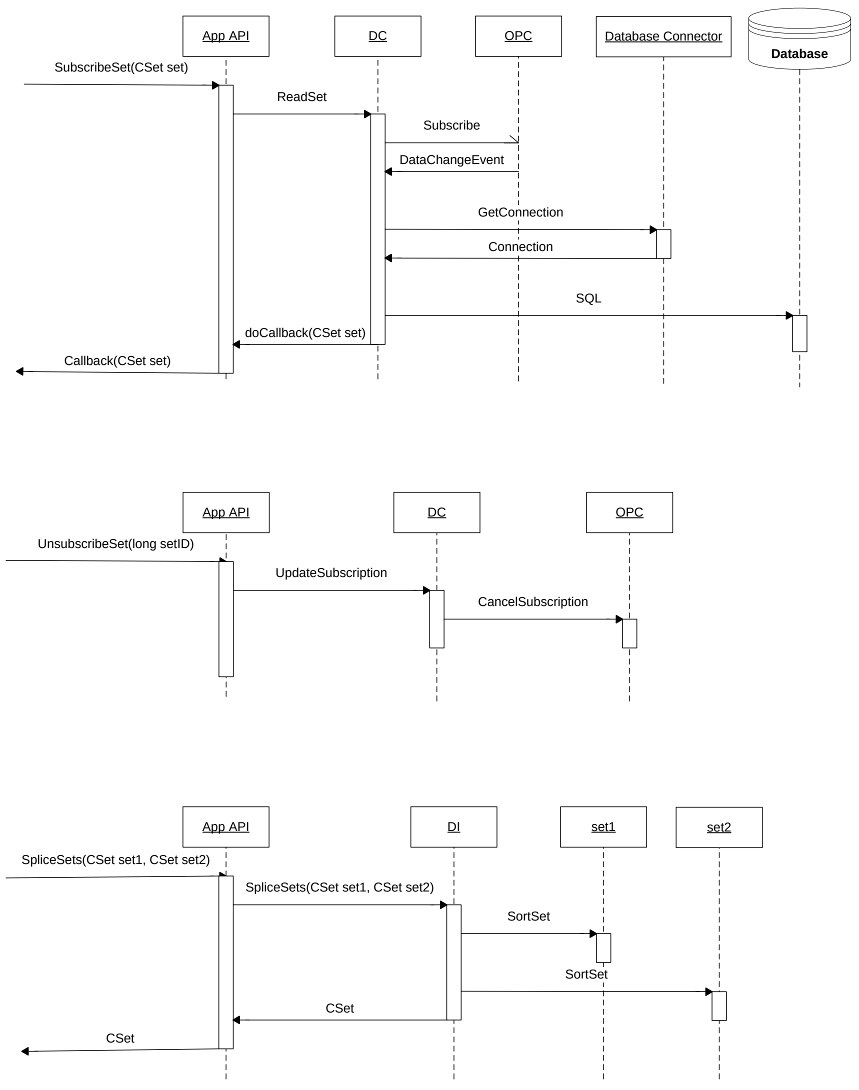


Figure 22.3: Subscribing or unsubscribing to set of tags

22.1.2 WEB - Web Interface

Interaction between the web modules can conceptually be divided into two parts. The modules make up the web user interface part of EasyIT, as shown in figure 21.1: The ServerConnector, TagBrowser, ConfigTool and AppMonitor.

The first type of interaction is the one concerning retrieving data from the database. A web page developer can run SELECT queries directly on the database. These data are typically series of tags the user has selected for viewing in the browser, or data web applications need for building their pages. Only two general examples are described with diagrams.

Figure 22.4 show a typical interaction between a web module and the database. The web module sends the request for data to the ServerConnector. The ServerConnector is responsible for calling the appropriate method in the Web API located at the server. The Web API gets the data from the database and returns it to the web module through the ServerConnector.

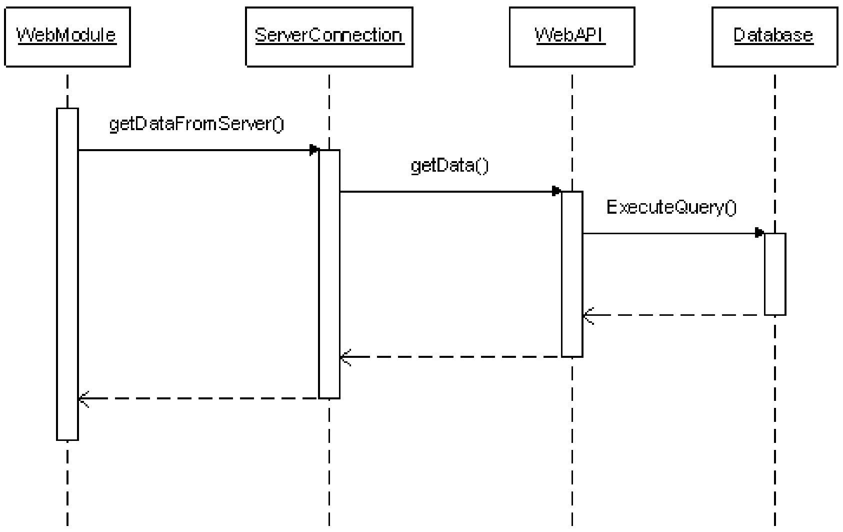


Figure 22.4: Typical interaction between a web module and the database

The other type of interaction is when a web module does not need anything directly from the database, for example the retrieval of application status for the AppMonitor web module. Here the DI has to run some logic to figure out whether the application is running, or not.

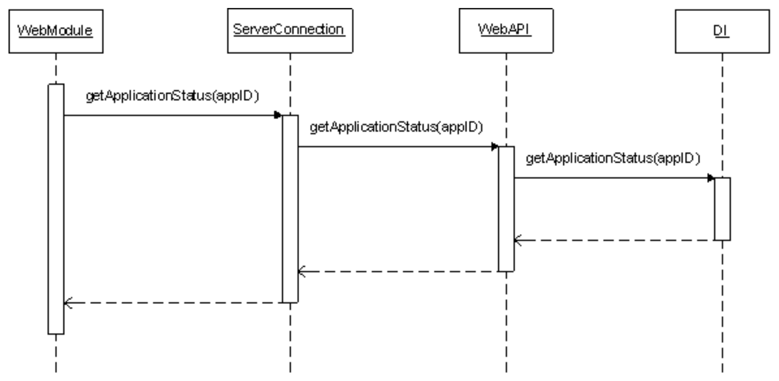

WebModule
ServerConnection
WebAPI
DI
getApplicationStatus(appID)
getApplicationStatus(appID)
getApplicationStatus(appID)

Figure 22.5: Typical interaction between a web module and the DI

The following tables give a detailed dependency description of the four WEB modules.

Attribute	**Description**
ID:	ServerConnector
Type:	WEB-module
Purpose:	This module exists because WEB needs a uniform way to read data from the database and communicate with the DI, possibly located at another server. The ServerConnector was created for requirement WEB-1.
Dependencies:	This module depends on the Web API for contacting the server.
Resources:	The ServerConnector relies on the .NET Remoting functionality. The server is responsible for registering an instance of the Web API as a well known service, which is then looked up by the ServerConnector.

Table 22.5: Dependencies for ServerConnector

22.1.3 External entities

Attribute	Description
ID:	ConfigTool and AppMonitor
Type:	WEB-modules
Purpose:	These modules exists to let the applications present their data on the web, and monitor application status. This is required by requirement WEB-2, WEB-5 and WEB-10.
Dependencies:	These modules depends on the ServerConnector for retrieving data from the database and communicating with the DI.

Table 22.6: Dependencies for the ConfigTool and AppMonitor

Attribute	Description
ID:	TagBrowser
Type:	WEB-module
Purpose:	These modules exists because users need to be able to view the data logged by DI. This is requrired by requirement WEB-8 and WEB-9.
Dependencies:	This module depends on the ServerConnector for retrieving data from the database.
Resources:	The TagBrowser relies on the ComponentOne [22] library for generating time plots of the data to be browsed. This library supports many different graphical depictions. This software library was recommended to us by ABB.

Table 22.7: Dependencies for the TagBrowser

Attribute	Description
ID:	Database
Type:	Data store
Purpose:	We need to store data in a structured way. We do this in a relational database. Related demands are API-11 to API-14 (18.12)
Dependencies:	Closely connected to the Database Connector.
Resources:	The database is a Microsoft SQL Server 2000 which in turn requires a Microsoft operating system (Windows 2000 or newer).

Table 22.8: Dependencies for Database

Attribute	Description
ID:	EasyIT Manager
Type:	Sub-program
Purpose:	We need a class to keep track of the instances of DC, DBC, APP API and WEBAPI, so they can communicate, i.e. launch subprograms at startup (DC) and applications. Related requirements are DI-8, 18.1
Dependencies:	none
Resources:	MS Windows OS with .NET

Table 22.9: Dependencies for EasyIT

CHAPTER 23
INTERFACE DESCRIPTION

This chapter describes the functionality of an entity and how other entities interact with the entity (i.e. exported methods). The UML diagram in Figure 23.1 shows the public methods of the main classes of the system. Also, each entity and its methods are discussed in more detail in tables.

23.1 MODULE INTERFACE

23.1.1 Class interfaces - DI

Attribute	Description
ID:	Data Collector
Function:	• un/subscribe to OPC-updates • minimize load on the OPC-servers • store received tag or set in the database • read instant value • pass instant value on to application • write tags to OPC
Public methods:	• **long ReadSet(CSet set)** Starts a subscription on the tags contained in the set. • **void StopReadSet(long setID)** Stops an existing subscription. • **void Startup()** Starts the Data Collector and its connections to OPC-servers. • **void Shutdown()** Stops the Data Collector and closes connections to OPC-servers. • **void WriteTagsToDB(CTag[] tags)** Writes a set of tags to the database. • **void WriteTagsToOPC(CTag[] tags)** Writes tags to OPC. • **CSet ReadTagsFromOPC(CTag[] tags)** Read tags from OPC.

Table 23.1: Interface for Data Collector

Attribute	**Description**
ID:	Database Connector
Function:	This entity facilitates reading and writing to the database.
Public methods:	• **SqlConnection GetSQLConnection()** Creates and returns a connection to the database. This connection is a *.Net*-object and provides methods querying the database. • **void ReleaseSqlConnection(SqlConnection sqlConnection)** Releases the connection after use in order to make it available for other applications. • **void Startup(void)** Starts the Database Connector and establishes the connection to the database. • **void Shutdown(void)** Shuts down the Database Connector and closes the connection to the database.

Table 23.2: Interface for Database Connector

Attribute	**Description**
ID:	Application API
Function:	The Application API is the interface between applications running on the system and the DI.
Public methods:	• **DataReceivedEventHandler DataArrival()** Event fired every time the application is supposed to receive OPC data. • **CSeries GetTags(DateTime dtStart, DateTime dtEnd, CTag[] tags, int gran)** This method creates a time series of logged tags. The granularity parameter specifies the desired number of samples. • **void WriteTagsToOPC(CTag[] tags)** Routes the call to the Data Collector. • **long SubscribeSet(CSet setm)** This method routes a subscription call to the Data Collector, thus starts a subscription. • **void UnsubscribeSet(long setID)** This method routes an end subscription call to the Data Collector, thus stops the subscription. • **CSet ReadTagsFromOPC(CTag[] tags)** Reads a set of tags from OPC. • **void WriteTagsToDB(CTag[] tags)** Routes a write call to the Data Collector. • **CSet SpliceSets(CSet set1, CSet set2)** Splices two sets. • **void WriteTagsToOPC(CTag[] tags)** Writes tags to OPC. • **void SetConfigValue(int id, string name, string valueString)** Writes a configuration value to the database. The *id* specifies the application and *name* specifies the entry to store the value. • **string GetConfigValue(int id, string name)** Reads a configuration value from the database.

Table 23.3: Interface for Application API

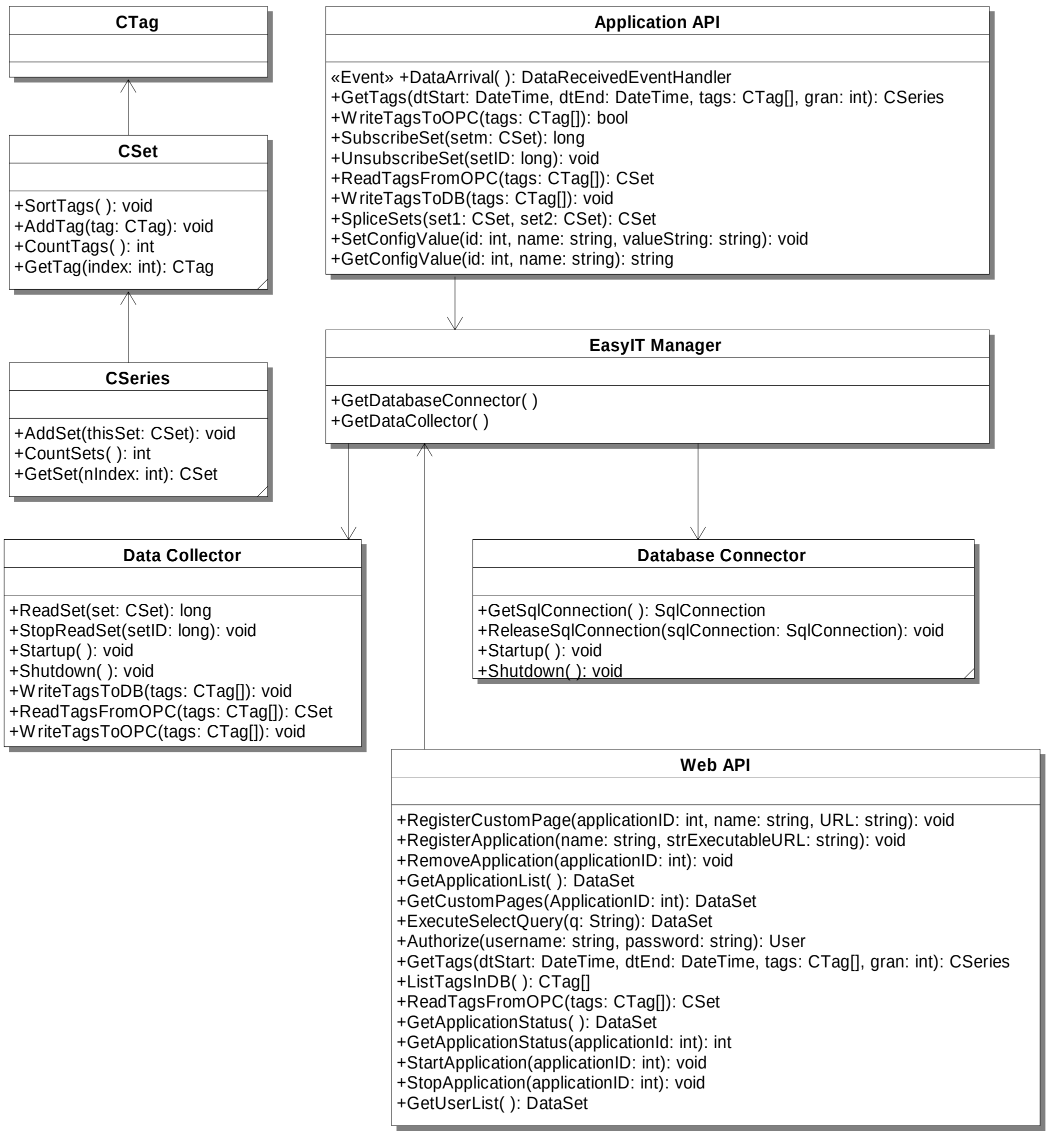

CTag
Application API
«Event» +DataArrival( ): DataReceivedEventHandler
+GetTags(dtStart: DateTime, dtEnd: DateTime, tags: CTag[], gran: int): CSeries
+WriteTagsToOPC(tags: CTag[]): bool
+SubscribeSet(setm: CSet): long
+UnsubscribeSet(setID: long): void
+ReadTagsFromOPC(tags: CTag[]): CSet
+WriteTagsToDB(tags: CTag[]): void
+SpliceSets(set1: CSet, set2: CSet): CSet
+SetConfigValue(id: int, name: string, valueString: string): void
+GetConfigValue(id: int, name: string): string
CSet
+SortTags( ): void
+AddTag(tag: CTag): void
+CountTags( ): int
+GetTag(index: int): CTag
EasyIT Manager
+GetDatabaseConnector( )
+GetDataCollector( )
CSeries
+AddSet(thisSet: CSet): void
+CountSets( ): int
+GetSet(nIndex: int): CSet
Data Collector
+ReadSet(set: CSet): long
+StopReadSet(setID: long): void
+Startup( ): void
+Shutdown( ): void
+WriteTagsToDB(tags: CTag[]): void
+ReadTagsFromOPC(tags: CTag[]): CSet
+WriteTagsToOPC(tags: CTag[]): void
Database Connector
+GetSqlConnection( ): SqlConnection
+ReleaseSqlConnection(sqlConnection: SqlConnection): void
+Startup( ): void
+Shutdown( ): void
Web API
+RegisterCustomPage(applicationID: int, name: string, URL: string): void
+RegisterApplication(name: string, strExecutableURL: string): void
+RemoveApplication(applicationID: int): void
+GetApplicationList( ): DataSet
+GetCustomPages(ApplicationID: int): DataSet
+ExecuteSelectQuery(q: String): DataSet
+Authorize(username: string, password: string): User
+GetTags(dtStart: DateTime, dtEnd: DateTime, tags: CTag[], gran: int): CSeries
+ListTagsInDB( ): CTag[]
+ReadTagsFromOPC(tags: CTag[]): CSet
+GetApplicationStatus( ): DataSet
+GetApplicationStatus(applicationId: int): int
+StartApplication(applicationID: int): void
+StopApplication(applicationID: int): void
+GetUserList( ): DataSet

Figure 23.1: Methods exported between entities

Attribute	**Description**
ID:	Web API
Function:	Web API provides an interface from DI to Web.
Public methods:	• **void RegisterCustomPage(int applicationID, string name, string URL)** Registerts a new custom page in the database. • **DataSet GetApplicationList()** Gets all applications registered in the database. • **DataSet GetCustomPages(int ApplicationID)** Gets all custom pages registered to an application. • **DataSet ExecuteSelectQuery(String q)** Executes a selectquery on the database. Other queries not allowed. • **CSeries GetTags(DateTime dtStart, DateTime dtEnd, CTag[] tags)** Gets tags in a given time interval. • **CTag[] ListTagsInDB()** Lists all the tags stored in the database. This method makes it possible to visualize what data is available in the tag browser. • **DataSet GetApplicationStatus()** Get application status for all applications.

Table 23.4: Interface for WEB API

23.1.2 Graphical user interface - Web

Attribute	**Description**
ID:	ServerConnector
Function:	This entity establishes a connection to the WEB API located at the server. This connection is then made available to all the web pages.
Interfaces:	The other modules contact the server through this module. The ServerConnector holds a reference to a server object the modules may use. The modules run methods on the referred object, and the ServerConnector forwards the request to the server, returning the data set to the module. Please se table 23.4 for a detailed description of the available interface methods.

Table 23.5: Interface for ServerConnector

Attribute	**Description**
ID:	TagBrowser
Function:	This module provides the users with at way to browse all dataseries logged by the DI. The data is made available either as a table or as a graphical time plot.
Interfaces:	• This is a user interface module, its purpose is to let the user browse tags logged by the DI. The browser consist of two web pages. In the first, the user selects which tags to browse, and chooses the time interval. Suitable user interface components will be chosen to accomplish this. The TagBrowser displays the data as a graphical timeplot and/or a table. Please see figure 23.2 and 23.3 for drafts of these pages. • Input ranges: For the first screen, where the user selects witch tags to browse, input needed is the time range. This has to be a valid date in a given format.

Table 23.6: Interface for TagBrowser

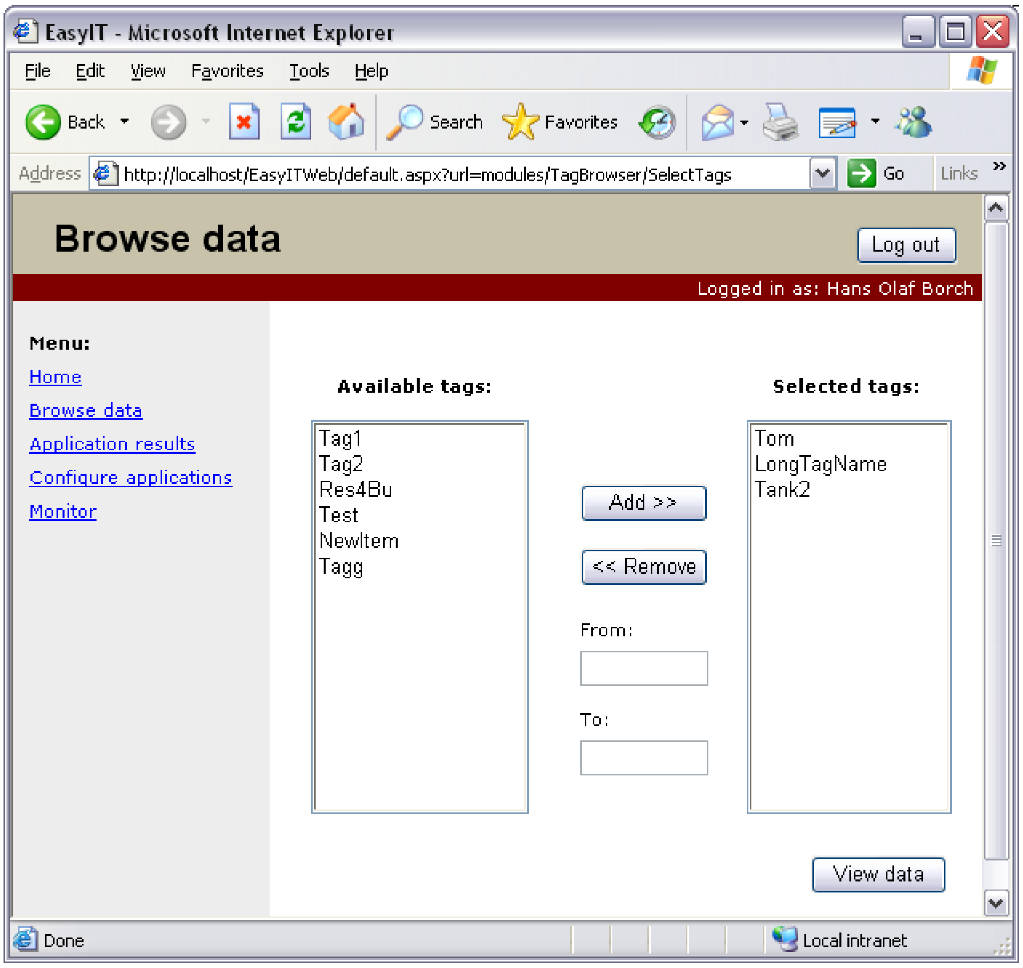


Figure 23.2: User interface 1 for TagBrowser. User selects tags for browsing

Attribute	**Description**
ID:	ConfigTool
Function:	This modules allows the administrator to set up whether an application shall present its data on the web portal. It should also allow the creation of empty skeletons for custom web pages associated with this application.
Interfaces:	• This is a user interface module. The ConfigTool lists the registered applications. The user chooses which application and logged data he wants to display. The ConfigTool generates a new page module, ready for the programmer to complete. This module needs interface needs an interface for adding and editing pages. A draft of this module is given in figure 23.4. • Input: The user has to specify a name and class when creating a new application. There might be an upper limit in these names, avoiding multiple lines in the menu.

Table 23.7: Interface for ConfigTool

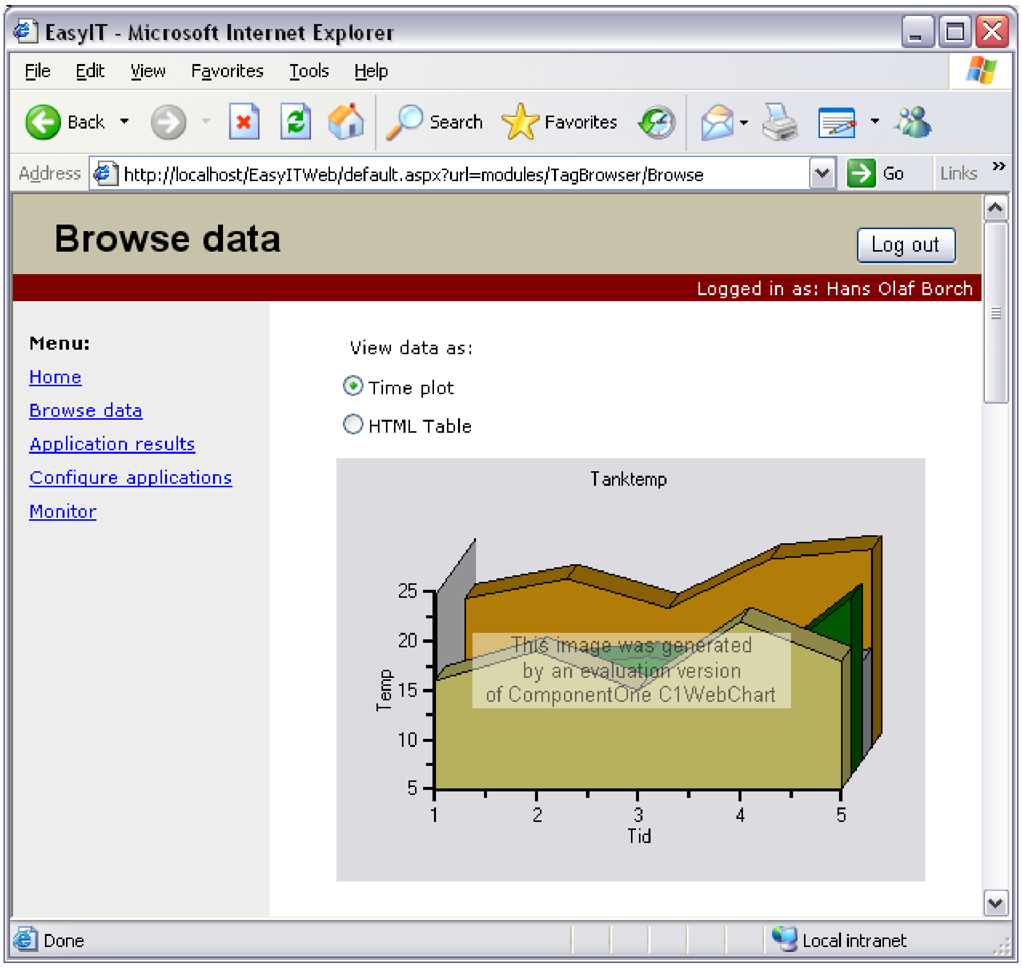
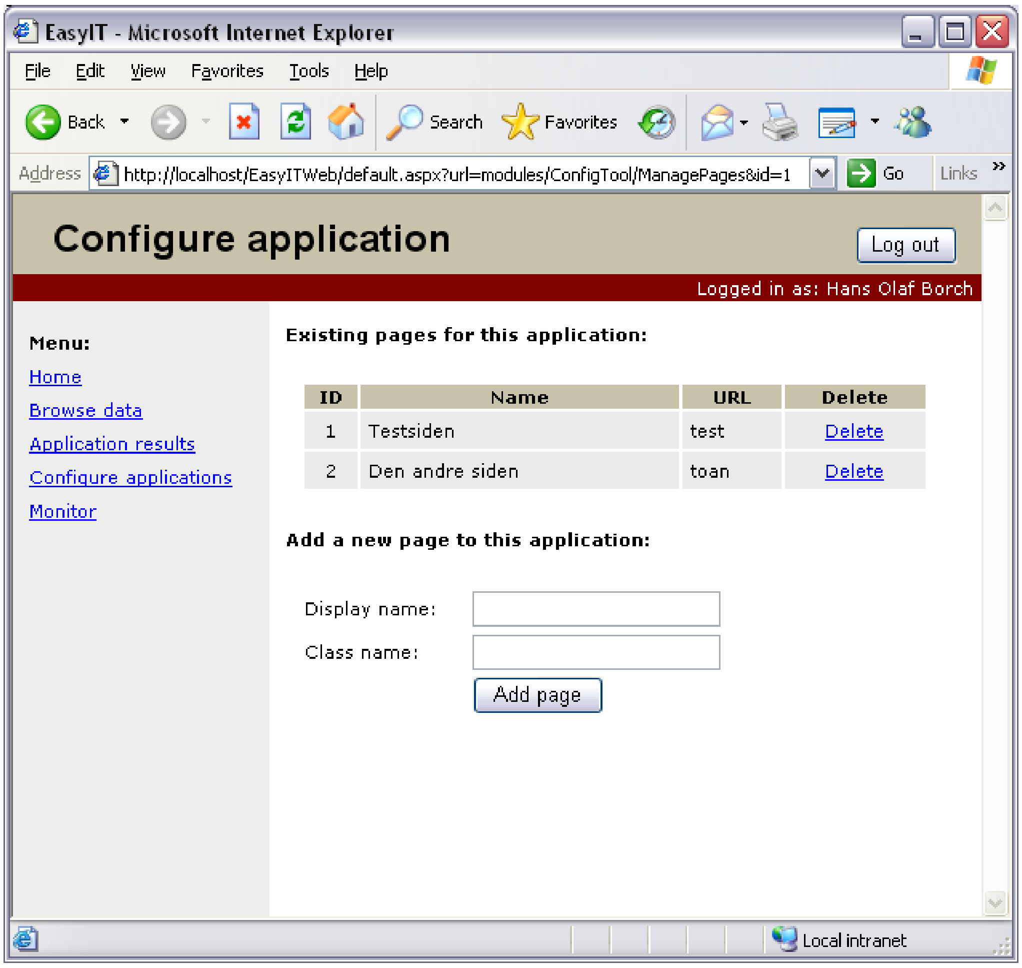

EasyIT - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/EasyITWeb/default.aspx?url=modules/TagBrowser/Browse Go Links
Browse data
Log out
Logged in as: Hans Olaf Borch
Menu:
Home
Browse data
Application results
Configure applications
Monitor
View data as:
Time plot
HTML Table
Tanktemp
Temp
Tid
This image was generated by an evaluation version of ComponentOne C1WebChart
Done
Local intranet

Figure 23.3: User interface 2 for TagBrowser. The chosen tags are displayed.

Attribute	**Description**
ID:	AppMonitor
Function:	The AppMonitor module queries the DI for status on all applications. These data are presented to the user.
Interfaces:	This is a user interface module. It lists the applications and their status. It does not have any user interaction, and a draft of this module can be seen in figure 23.6.

Table 23.8: Interface for AppMonitor


EasyIT - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/EasyITWeb/default.aspx?url=modules/ConfigTool/ManagePages&id=1
Go Links
Log out
Local intranet

Configure application

Logged in as: Hans Olaf Borch

Menu:
Home
Browse data
Application results
Configure applications
Monitor

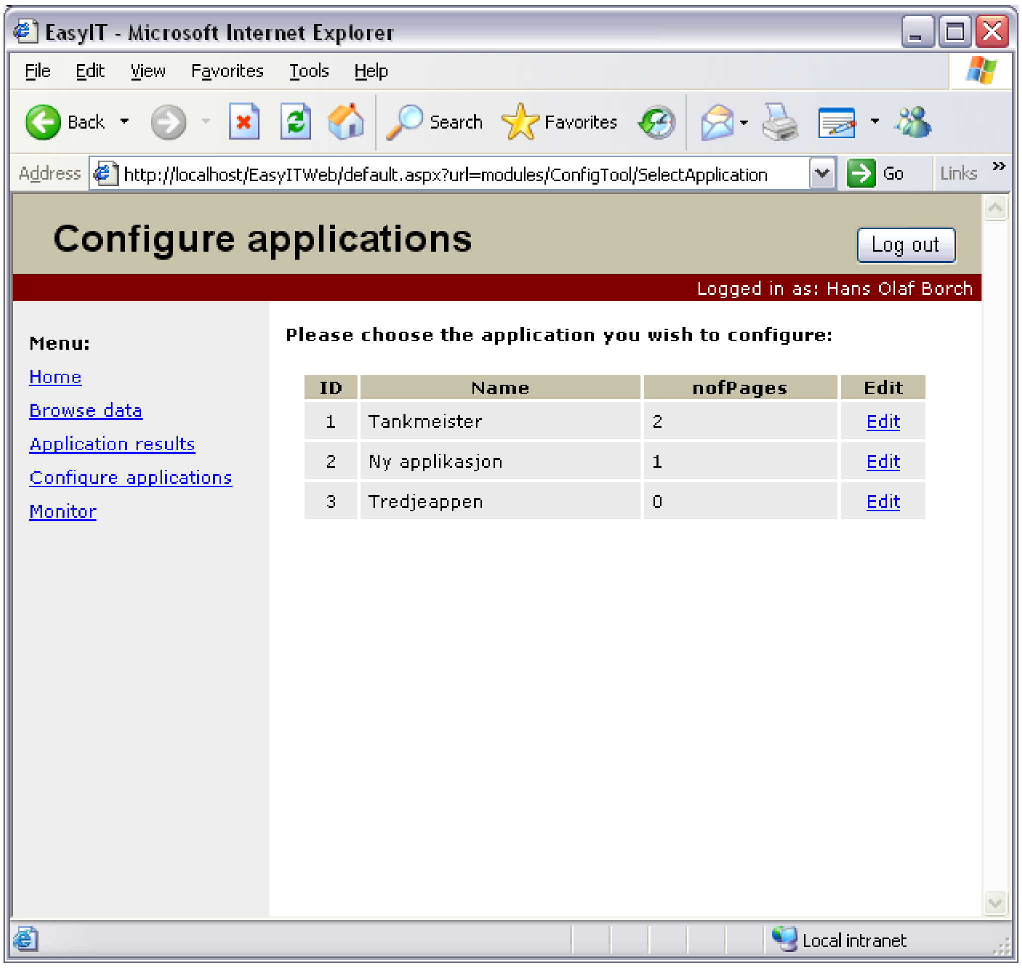
Existing pages for this application:

ID	Name	URL	Delete
1	Testsiden	test	Delete
2	Den andre siden	toan	Delete

Add a new page to this application:

Display name:	
Class name:	
	Add page

Figure 23.4: User interface for ConfigTool, configure new web application


EasyIT - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/EasyITWeb/default.aspx?url=modules/ConfigTool/SelectApplication
Go Links
Log out
Local intranet

Configure applications

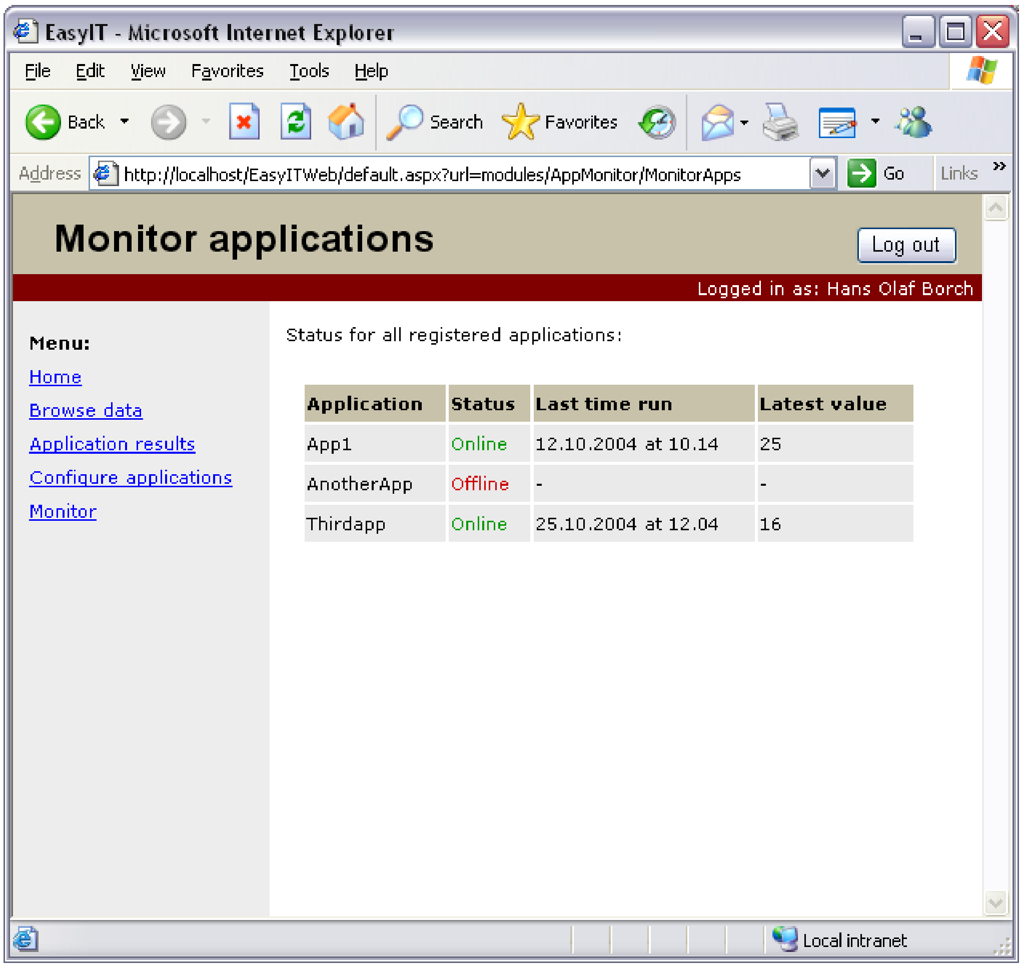
Logged in as: Hans Olaf Borch

Menu:
Home
Browse data
Application results
Configure applications
Monitor

Please choose the application you wish to configure:

ID	Name	nofPages	Edit
1	Tankmeister	2	Edit
2	Ny applikasjon	1	Edit
3	Tredjeappen	0	Edit

Figure 23.5: User interface for ConfigTool, select web application for configuration


EasyIT - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/EasyITWeb/default.aspx?url=modules/AppMonitor/MonitorApps
Go Links
Local intranet

Monitor applications

Log out

Logged in as: Hans Olaf Borch

Menu:
Home
Browse data
Application results
Configure applications
Monitor

Status for all registered applications:

Application	Status	Last time run	Latest value
App1	Online	12.10.2004 at 10.14	25
AnotherApp	Offline	-	-
Thirdapp	Online	25.10.2004 at 12.04	16

Figure 23.6: User interface for AppMonitor

23.1.3 External entities interface

Attribute	**Description**
ID:	Database
Function:	The entity gets data as input and stores them in relation tables. It can also fetch data from these tables and return this as output.
Interfaces:	We only interface the database through SQL.

Table 23.9: Interface for Database

Attribute	**Description**
ID:	EasyIT Manager
Function:	Start the DC, distribute references of modules.
Interfaces:	• **Database Connector GetDatabaseConnector()** returns the running instance of the database connector. • **Database Data Collector GetDataCollector()** returns the running instance of the data collector.

Table 23.10: Interface for EasyIT Manager

23.2 Process interface

Our design has proven to have very close connection between modules and processes. EasyIT Manager, Data Collector, Database Connector, Applications and Web Apps can be considered processes, leaving the interfaces as passive actors. Keeping this in mind, the dependency relations in Figure 22.1 illustrates what processes uses interface on other processes.

Chapter 24
Detailed design

This chapter describes what rules are governing an entity, and what algorithms are applied in specific processes. To describe this we have employed ns-charts and natural language. Internal data-entities critical to the operation is also to be listed. This relates to the previous chapter as a specification of how the methods are to be implemented.

24.1 Module detailed design

24.1.1 DI - Data Interchanger

Attribute	**Description**
ID:	Data Collector
Processing:	This is described in the implementation-document.
Data:	This is described in the implementation-document.

Table 24.1: Detailed description for Data Collector

Attribute	**Description**
ID:	Database Connector
Processing:	This is described in the implementation-document.
Data:	This is described in the implementation-document.

Table 24.2: Detailed description for Database Connector

Attribute	**Description**
ID:	Application API
Processing:	This is described in the implementation-document.
Data:	This is described in the implementation-document.

Table 24.3: Detailed description for Application API

Attribute	**Description**
ID:	WEB API
Processing:	This is described in the implementation-document.
Data:	This is described in the implementation-document.

Table 24.4: Detailed description for WEB API

24.1.2 WEB - Web Interface

Attribute	Description
ID:	ServerConnector
Processing:	The ServerConnector instanciates a .NET Remoting object called MarshalByRefObject. This object enables access to objects across application domain boundaries in applications that support remoting. This object holds a connection to the EasyIT DI, and enables WEB modules connect to the database to get data. The ServerConnector implements one method; getServerObject(), which returns the MarshalByRefObject.
Data:	This module uses an instance of the ServerObject class, which holds a referance to the server.

Table 24.5: Detailed description for ServerConnector

Attribute	Description
ID:	TagBrowser
Processing:	• The user is first presented with an overview of available tags to browse in some sort of tree view. The system is supposed to be able to collect tags from several OPC servers, and it's therefore natural to let the first level in the three be the server. The next level is the path, and the last, the tag's name. • The user enters the time span he wants to browse, and submits the form. The time span can be collected from two textboxes, represented as .net DateTime objects in the system. • The TagBrowser asks for the approperiate data from DI with the chosen tags and timespan. This request is sent through the ServerConnector, and the data are recieved as a CSeries object. • On a new page, the user is presented with the chosen data. He has the choice of viewing them as tabular data and/or as a graphical time plot. The graph component from Component1 [22] will be used, to save implementation time. • While viewing the results, the user can switch between the display modes for instance through radio buttons. The table can be realized as a .net DataTable, wich can be bound to a DataSet object. A conversion from our custom dataset cSeries will be needed. Please see figure 23.2 and 23.3 for a proposed layout.
Data:	The TagBrowser uses Tags and Sets of Tags as described in DI, retrieved through the ServerConnector.

Table 24.6: Detailed description for TagBrowser

Attribute	**Description**
ID:	ConfigTool
Processing:	• The ConfigTool initially retrieves the configuration of all applications through the ServerConnector represented as a DataSet. • The user is presented with a table containing status information from all appliactions. This information includes whether the application is set up to present data on the web. • If an application is not set up to publish data, the user can enable this, and set up one or more custom pages. A new empty file will be created locally on the web server, for the web page developer to fill in. Information about the new page(s) will also be added to the database through the ServerConnector.
Data:	The ServerConnector returns a DataSet containing the configuration for all applications.

Table 24.7: Detailed description for ConfigTool

Attribute	**Description**
ID:	AppMonitor
Processing:	• When the user selects "Monitor applications" in the menu, the AppMonitor control requests the status of all applications through the ServerConnector. • The user is presented with a table containing status information from all appliactions. This information includes whether the application is running or not, what the last value written by the application was etc.
Data:	The ServerConnector returns status for all applications, represented as integers.

Table 24.8: Detailed description for AppMonitor

24.1.3 External entities

Attribute	**Description**
ID:	Database
Processing:	The database has an underlying database managment system, which handles transactions and keeps the database consistent at all times.
Data:	Internal representation of data has no relevance.

Table 24.9: Detailed description for Database

Attribute	**Description**
ID:	EasyIT Manager
Processing:	This modules does no processing of data.
Data:	This modules does no processing of data.

Table 24.10: Detailed description for EasyIT Manager

24.2 DATABASE DETAILED DESIGN

This section shows how our database is organized. There are two separate uses that are covered: keeping OPC-data and keeping information about applications.

Attribute	Description
ID:	OPC datatables
Processing:	This modules does no processing of data.
Data:	The OPC datatables are three tables. The tblTag stores each tag logged by the system. Each tag has it's unique identifyer, an instanceID. The strName, strServer and strPath attributes are also considered as unique for a seires of a logged tag over time. Tags can also be collected and delivered to the system as sets, defined by the tblSet entity. A set is a collection of tags, where all tags have the same timestamp. Then, again, sets can be put together in setseries, defined in the tblSeries entity. Please see figure 24.1 for an ER-diagram of the three entities. Details tblTag: • **nInstanceID**: Unique identifyer for this tag • **nTagID**: This tag's ID • **strPath**: Path tag logged on the server • **strName**: Name of the tag • **dtTimestamp**: The time the tag was logged • **strValue**: The tag's value • **nDataType**: The tag's datatype • **bQuality**: The quality of the tag's value • **nDeadBand**: The tags deadband variable • **nServerID**: The ID of the server the tag was logged from • **nApplicationID**: If value written by an EasyIT application, the application's ID is set • **strServer**: The server the tag was logged from Details tblSet • **nSetID**: An ID identifying this series • **dtDate**: the time the set was logged • **nRate**: The set's rate, how often are values collected from OPC • **fDeadBand**: This set's deadband variable • **bActive**: Whether this set is an active set, or not Details tblSeries: • **nSeriesID**: An ID identifying this series • **nRate**: This seres rate, how often are the sets collected from OPC.

Table 24.11: Detailed description for the OPC datatables

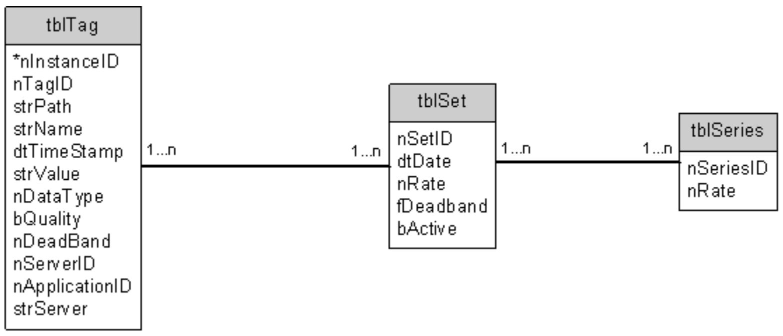


Figure 24.1: ER diagram of the OPC data table

Attribute	Description
ID:	WEB datatables
Processing:	This modules does no processing of data.
Data:	The WEB part of EasyIT needs to store which applications there are in the system, and the pages these applications have. This is necessary to let the WEB modules make the menu, and list information about registered web pages. Please see figure 24.2 for an ER diagram of the tables. Details tblCustomPages: • **PageID**: Unique identifyer for this page • **applicationID**: Which application this page belongs to • **PageURL**: The URL path to where this page is located on the WEB server • **Name**: The page's name Details tblApplication: • **nID**: Unique identifyer for this application • **Name**: The name of the application • **strExecutableURL**: The URL path to where this application can be launched on the WEB server

Table 24.12: Detailed description for the WEB datatable

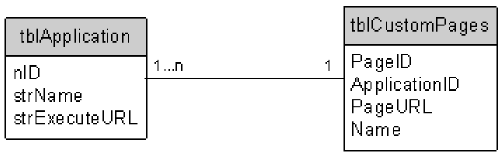


Figure 24.2: Diagram of the custom pages table

Part V

Implementation document

CHAPTER 25
INTRODUCTION

This chapter is a completion to the Software Design Description document. It contains all the details of the implementation work and a tries to describe the important choices we made and most importantly why we made them. As EasyIT is a prototype this document should prove itself as a very interesting study to the customer should he decide to continue the development of the product. Avoiding mistakes and utilizing the good ideas is what prototyping is all about.

This chapter is organized in three parts. First there is a class diagram(26.1), second an overview over how the system was built(26.2) and last there is a section with very detailed information with the experiences we had implementing the different parts(27).

Chapter 26
Mode of operation

26.1 Class diagrams

In Appendix H, a class diagram over the EasyIT system can be found. All the important classes are included there. The different seperate classes are presented in figure H.2 to figure H.7, while figure H.1 gives an overview over how the classes are connected. A much more detailed textual description of the system follow in the rest of this implementation document and these class diagrams should be used as a guide when reading through it.

26.2 Implementation partitioning

EasyIT was implemented as a Visual Studio .NET solution. The solution was separated into several projects. There were two reasons for this, the first reason was to group the files in smaller functionally connected groups and the second was to allow for easy distribution of classes needed in multiple locations. Our data items and the interface definitions fall in the second group. These are needed on the EasyIT Server, the client applications and the EasyIT Web. If these data items and interfaces are not in their own projects, we would need to distribute all the files in the entire solution to everybody who is interacting with EasyIT. This not desirable, the Application Library should only be dependant on the data items and the interfaces, not on the web portion of the project.

The solution was divided up in the following parts:

- ApplicationLibrary - all the files that makes up the Application Library.
- RemoteInterfaces - all the interfaces defined for use with .NET Remoting. The AppAPI and WebAPI interfaces resides here.
- DataItems - all the data items.
- EasyIt - all the files for the EasyIT Server, including all the files for the DataCollector, the DBConnector, the scheduling functionality and the implementation of AppAPI and WebAPI. This project also includes the EasyIT Manager. The entities earlier referred to as DI, APP and APP-API is also implemented here.
- EasyItStarter - The project with the executable that starts the EasyIT Manager.
- EasyITWeb - The entire web portion of EasyIT.

26.3 EasyIT Server / EasyIT Manager

26.3.1 Technical background

The EasyIT Server / Manager contains all parts from the above list except EasyITWeb. It is implemented in the C# language and uses .NET Remoteing for communication with both the client applications and EasyITWeb.

26.3.2 Framework

EasyIT Server and Manager is implemented around the EasyIT Manager. The manager is the glue that holds all the other parts together. It is the main class that gets started by the EasyITStarter executable. Its job is to start up the entire system and provide a central point of connection for all the other modules. It also provides a central location for logging error messages for the entire system. It is implemented in EasyIt.cs. Under the EasyIT Manager we also find the data items. The data items are classes that represent essential entities in the EasyIT system. We have a class representing an OPC tag, a class representing sets of these tags and a class representing series of such sets. These classes are implemented in the three following files: CTag.cs, CSet.cs and CSeries.cs. These classes are used to pass around information about these entities. If an application wants to read the value of a tag, it can just fill out just pass an instance of the CTag class as an argument. The manager is also the part that starts up the other modules of EasyIT, except for EasyITWeb. EasyIT Manager is also the entity that implements the AppAPI.

26.3.3 Accessing the APP API

The following list lists all the methods the application developers have access to through the AppAPI.

- public CSeries GetTags(DateTime dtStart, DateTime dtEnd, CTag[] tags, int gran)
- public CSeries GetAllTags(DateTime dtStart, DateTime dtEnd, CTag[] tags)
- public CSet ReadTagsFromOPC(CTag[] tags)
- public bool WriteTagsToOPC(CTag[] tags)
- public void WriteTagsToDB(CTag[] tags)
- public long SubscribeSet(CSet setm)
- public void UnsubscribeSet(long setID)
- public CSet SpliceSets(CSet set1, CSet set2)
- public void SetConfigValue(string name, string valueString)
- public string GetConfigValue(string name)

A detailed explanation on the use of each of them follows.

When CTag instances are expected as input in calls to these methods, it is vital that the Name, Server and Path properties of these objects match those of actual OPC tags. If these are not correct, EasyIT and all its subcomponents have no way of knowing which tag it should read or write.

GetTags(DateTime dtStart, DateTime dtEnd, CTag[] tags, int gran) is used to get the logged values of the specified tags in an interval between two points in time, start and stop, with a certain granularity. This is executed synchronously and a call to this method will block until it completes. The result is returned in a CSeries instance. For a detailed description of this important method, see 27.5

GetAllTags(DateTime dtStart, DateTime dtEnd, CTag[] tags) is used to get all the logged values of the specified tags in an interval between two points in time, start and stop. This is executed synchronously and a call to this method will block until it completes. The result is returned in a CSeries instance.

ReadTagsFromOPC(CTag[] tags) reads the values of the specified tags once directly from the OPC servers the tags reside on. This is executed synchronously and a call to this method will block until it completes. The result of this call, the current value of the tags, is returned as a CSet.

WriteTagsToOPC(CTag[] tags) writes the value of the specified tags once directly to the OPC servers the tags reside on. The value of each tag is specified in each of the tags given as input. This is executed synchronously and a call to this method will block until it completes. . The result of this call is true if there were no problems writing the tags, or false if there were.

WriteTagsToDB(CTag[] tags) writes the value of the specified tags once directly to the database. The value of each tag is specified in each of the tags given as input. This is executed synchronously and a call to this method will block until it completes.

SubscribeSet(CSet setm) starts an subscription of the tags in the specified set. Each tag can specify a deadband variable through the Deadband property, or leave it at the default 0.0. The set must specify the rate at which the client application wishes to get new data at by setting it at the creation of the set or by updating the Rate property. The method is executed synchronously and a call to this method will block until it EasyIT acknowledges the subscription. The data from the subscription however is returned asynchronously through the DataArrival event in the AppAPIClientBase class. If the client wishes to do something with this data is has to register an event-listener on this event. The result of this call is a long integer representing the unique id assigned to this subscription by the EasyIT DataCollector. This id must be saved for later if you wish to unsubscribe from the subscription.

UnsubscribeSet(long setID) stops a previously started subscription. An id previously returned from the SubscribeSet-method must be passed as the only argument. The method is executed synchronously and a call to this method will block until it EasyIT has unsubscribed the set completely.

SpliceSets(CSet set1, CSet set2) splice to sets. It does this by adding all the tags from both sets to a new set and removing all but the latest, the ones with the highest timestamp, values of each tag. The end result is a new set with the latest value of all the tags from set1 and set2 in it. The method is executed synchronously and a call to this method will block until it completes.

SetConfigValue(string name, string valueString) stores name/value-pair in the database. This name/value-pair is unique for this application. Storing a value with the same name as previously stored pair overwrites the old value. Any value can be stored as long as it can be represented as a string. It is up to the application developers to handle the conversion from and to a string. The method is executed synchronously and a call to this method will block until it completes.

GetConfigValue(string name) retrives the value of the name/value-pair with the specified name for this application from the database. The method is executed synchronously and a call to this method will block until it completes.

The two sample applications in figure G.1 (PressureChecker) and G.3(Averager) show full code examples for most of the methods mentioned above. The source code in these two figures can also be found in the two files PressureChecker.cs and Averager.c in the "application-samples" sub-folder where EasyIT was installed.

26.4 WEB

26.4.1 Technical background

EasyITWeb is written in ASP.NET, using C# for server-side data manipulation. ASP.NET supports two kinds of pages: stand-alone pages (extension .aspx) and so called UserControls (extension .ascx). Each .aspx/.ascx page has a corresponding C# file connected to it. The C# files are called the Codebehind files for the .aspx/.ascx files, and are named .aspx.cs or .ascx.cs respectively.

26.4.2 Framework

The entire web interface consists of one aspx file, called default.aspx, which includes several ascx UserControls. The default.aspx page loads the menu as a UserControl, and the content of the page as another UserControl. For each request the url to the UserControl to be loaded as main content of the page is given in the querystring. For instance: default.aspx?url=modules/TagBrowser/SelectTags would load the SelectTags.ascx UserControl.
The reason for choosing this approach is that is simplifies managing the page layout considerably. A problem with all web sites is having to include header and footer files on each page in order to make them look alike. We've used ASP.NET 1.1, which has no built-in support for doing this. On the contrary, ASP.NET 2.0 supports so called 'Master-pages' for creating common layout for all pages.
Our solution uses ASP.NET PlaceHolders, which can be loaded with UserControls at run-time. This gives us the additional advantage of being able to let for instance the 'Create application' button call a refresh method on the menu UserControl directly, because references to these object can be passed around freely.

26.4.3 Accessing the WebAPI

The first time the EasyITWeb is accessed the class ServerObject.cs is used to connect to the EasyIT server. The hostname and port number for contacting the server is configured in the xml file EasyITWeb.config in the EasyITWeb root folder. The reference to the server is stored as an instance of the WebAPI in a so called 'application variable' available to all ASP.NET pages. To use the WebAPI the line

```
WebAPI webapi = (WebAPI)Application["webapi"];
```

is used. Having obtained this reference, the programmer can access the methods of the WebAPI as if it was a local object. The following code gives an example, by obtaining a set of data from the database:

```
DataSet myData = webapi.ExecuteSelectQuery("SELECT Name from tblApplications");
```

26.4.4 Developing custom pages using the WebAPI

This section will go through the methods available to support custom page developing in our system. These mehods are all available through the WebAPI (see previous section for instructions on how to access it).

CTag[] ListTagsInDB()
Gets a list of all tags stored in the database. The tags returned have "na" set as value, and are therefore only useful for identifying a tag. Subsets of the returned CTag[] array can be used as parametres for the GetTags and GetAllTags methods.

CSeries GetTags(DateTime dtStart, DateTime dtEnd, CTag[] tags, int gran):
Gets a series of tags in the given time interval, averaging values in each timeslot. Each timeslot length is defined as the dtStart to dtEnd interval, divided by granulatity.

Details:

- **dtStart:** The start date for plotting
- **dtEnd:** The end date for plotting
- **tags:** The tags wanted to be plotted, need to have valid Name, Path, Server. To get a list of available tags stored in the database, use the `CTag[] ListTagsInDB()` method.
- **gran:** Granularity of the data, give as the max number of points to be returned. If data is missing in a timeslot, a tag with value "na" is returned.

- **Return value CSeries:** A series of tags logged in the specified time interval, passed as a DataItems.CSeries object.

CSeries GetAllTags(DateTime dtStart, DateTime dtEnd, CTag[] tags)
Gets all tags logged in the database in the given time interval. Does not do any calculations on tag values. This method must be used with caution; a single call can return all data stored in the database.

Details:

- **dtStart:** The start date for plotting
- **dtEnd:** The end date for plotting
- **tags:** The tags wanted to be plotted, need to have valid Name, Path, Server. To get a list of available tags stored in the database, use the `CTag[] ListTagsInDB()` method. returned. If data is missing in a timeslot, a tag with value "na" is returned.
- **Return value CSeries:** A series of tags logged in the specified time interval, passed as a DataItems.CSeries object.

CTag GetFirstOccurence(CTag tag)
Get the first occourence of a tag stored in the database.

Details:

- **tag:** The tag wanted the first occurence of.
- **Return value CTag:** The first occurence of the specified tag.

CTag GetLatestOccurence(CTag tag)
Get the latest occourence of a tag stored in the database.

Details:

- **tag:** The tag wanted the latest occurence of.
- **Return value CTag:** The latest occurence of the specified tag.

Chapter 27 Specific implementation information

27.1 Implementing the DBConnector

The DBConnector is responsible for all communication with the database and is implemented in the DBConnector.cs file.

Early in the implementation phase it became apparent that the system would access the database very often. The typical query against the database would be short, but it would happen often and concurrently with other access. An example is the need of the DC to log incoming data from OPC servers. Only one row in the table called tblTag is written, but the operation will happen once for each time an OPC server issues a callback to the system. From earlier experience we know that opening new connections to the database can be rather time consuming because of the need to go through a TCP/IP handshake every time. If we had just opened a new connection to the database each time we needed to run a query, the time needed to open the connection would by far exceed the time needed by the database to execute the actual query. This is not an acceptable solution. To solve this we implemented a connection pooling system. This system starts up when EasyIT Manager is started. When it starts, the pooling system opens n connections, where n can be tweaked for optimal performance, to the database. These connections are then made available to the rest of the system through the two methods GetSqlConnection() and ReleaseSqlConnection(SqlConnection sqlConnection). When some part of the system needs to access the database it gets an already open connection to the database by calling the GetSqlConnection-method. When it is finished, it releases the connection back into the pool by calling the ReleaseSqlConnection-method. The connections are never closed until EasyIT Manager closes down. If all the open connections in the pool are in use, the connection pool is dynamically expanded with new connections as needed. This way, the time penalty for opening the connections is taken exactly once, instead of each and every time we need to run a query. We feel that this kind of functionality is needed for any system doing more than extremely small amounts of database access.

27.2 Implementing the scheduling mechanism of EasyIT Manager

The requirement specification, see Figure 18.1: DI-7, required EasyIT Manager to have the possibility to schedule applications. This is implemented in the following fashion. Each application registered with the system is started when EasyIT Manager is started. The applications themselves are responsible for requesting the data they need at the interval they want it at. If an application wants to read a specific tag once every 10 minutes, it subscribes through the APP-API to a set including the tag with rate 600 seconds. It then goes to sleep until the EasyIT Manager fires a callback every 600 seconds with new data. Another possibility for implementing scheduling is to start and stop the application each time they want data. Considering the fact that the applications are stand alone applications, we don't think this is a good idea. Stand alone applications are external entities and we have little control over how much resources and time is used to start an application. Windows does quite a lot of housekeeping when it starts an application and we wanted to avoid having to do this over and over. It is much more preferable to start all applications at startup and let them sleep in the background. This way of thinking can also be observed in the section describing the implementation of the DBConnector. The connection pooling solution for SQL-connections and this way of solving the application scheduling are both instances of using a "create once, then use multiple times" way of thinking that we feel greatly improves the efficiency and scalability abilities of the system. It is also very easy for the end user to do the "scheduling", all he has to do is to subscribe to a set with the rate it wants to receive data at. This is important, considering that ease of use was one of the main success criteria ABB had for this project.

The actual implementation of scheduling in EasyIT Manager is implemented in the file Application-Launcher.cs. The StartAllApplication-method in this file is called when EasyIT Manager starts. This method reads from the database all the different applications and the path to their executables. It then spawns all the applications as separate processes.

27.3 Implementing the DataCollector

The DataCollector is one of the most crucial parts of the system. It's the part that has all the logic that is needed to communicate with the OPC servers. The DataCollector has two main methods for communicating with the world, called ReadSet(CSet set) and StopReadSet(int setID). The first method takes in a set of tags and adds these tags to the pool of tags that EasyIT is currently subscribing to from the OPC servers.

The job of the DataCollector is divided into two parts: getting data from the OPC servers and getting this data back to the clients(the applications that utilize the system). The two parts can be very closely connected or they can be separates from each other. We have chosen to separate them; the reason for this is explained later in this section. First we explain how we have implemented the two different parts.

The first part is to solve the problem "what do I do when some application wants to subscribe to a set of tags?" The easiest way to do this is to simply create a new subscription to a OPC server for each of the tags in the set. This sounds like a good solution until you think of what happens when you have more than a few tags in the pool. (And this is extremely likely in a production environment!) If the pool contains 500 tags to one server we would have 500 individual subscriptions. If these tags have a rate of 1 second, we would get 500 callbacks from this one server alone each second! Doing things this way gives us enormous scalability issues. A much better solution is to try to build optimal subscriptions on each server. An optimal subscription in the above scenario is just one subscription, with all 500 tags in it. It would result in exactly one callback each second with new data for all 500 tags. This results in a huge saving in overhead on both our system and on the OPC server in question. We opted for the optimal set solution. What we actually do when ReadSet(CSet set) is called is the following:

1. Check if we already have a subscription for this tag at the rate indicated in the set, or at a rate that is a divider of the rate indicated in the set. For example; if tag A is requested at rate 6, meaning once every six seconds, and we are already subscribing to tag A at rate 3, we don't need to start a new subscriptions to OPC server of the tag. The application that requested the tag at rate 6 is just given back every second value returned from the already running subscription at rate 3.

2. If the tag was not covered by running subscriptions in 1) we check if we have an open connection to the tags OPC server. If we don't, we open one.

3. We check if there are subscriptions on the tags server at the rate we want the tag at. If there isn't, we add a new subscription to the tags OPC server. We set the rate for the subscription to the rate we want the tag at and tell the OPC server to call us back at the given interval.

4. After 3) we know we have a live subscription, either an old subscription or a new subscription created in 3), with the rate we want on the tags OPC server. We then add the tag to this subscription.

The OPC server will now issue callbacks to the DataCollector with the current value of the tags in the subscription at the interval we told it. When the DataCollector gets a callback, it writes the value of all the tags in the callback to the database through the DBConnector.

The StopReadSet method does this in reverse.

1. Are we subscribing to this tag and are there no other sets also using this subscription? This is the reverse of the previous list.
2. If 1) is true, remove the tag from the subscription.
3. If the subscription in 2) becomes empty, remove the subscription.
4. If the server of the description in 3) has no more subscriptions, close the connection to the server.

The second part of the DataCollector deals with getting the data back to the client that actually requested it in the first place. One obvious way to do this would be to just call the clients back with new data when we have a callback from the OPC servers. The problem with this method is what happens when the client requests from multiple OPC servers in the same set. To mimimize the traffic between the EasyIT system and the client applications, we want to deliver whole sets of values. If client C tells EasyIT that it wants to subscribe to set S with tag A on server 1, tag B on server 2 and tag C on server 3 at rate 4, it should get a callback from the DataCollector with a new set with the value of tag A, the value of tag B and the value of tag C every fourth second. How can we accomplish this when we can't guarantee the condition of the servers the tags are at? The answer is: we can't. We have no way of knowing if the three OPC servers actually will give back data at the given interval. A OPC server can crash, can be disconnected or just have a high load and respond very slowly. OPC makes a promise of "best effort". When our data sources are "best effort", this is the best we can promise EasyIT to be too. This means we can't guarantee that we have new data for all, or any, of the tags the client applications have subscribed to at the given rate. The fact that we are "best effort" means that the callback from the OPC servers can seem quite random at times. Given the success criterion that EasyIT should be very easy to use for application developers, we think this should be hidden from the client applications. They should be separated as much as possible from the irregularities of the OPC servers.

Our solution to this, and our reason for choosing to separate the two parts of the DataCollector, is to have no direct connection between the callbacks from the OPC server and the clients that requested the data. When OPC calls us back with new data, we simply write it to the database with the DB-Connector. A separate thread, running in the background, checks all the sets of the current running applications every second. Has rate seconds passed since the last time we checked? If it has, get the newest data for the tags in the set directly from the database and give this data to the client application that subscribed to the set. Subscribing to a set of tags means you get back the latest values of the tags that OPC has put in the database. Doing things this way means we can guarantee that applications get a callback at the rate it requested, we just can't guarantee that we have new values for all the tags in callback. We think this solution is easier for the application programmer, as we at least can guarantee timely callbacks. In an ideal world we would want to guarantee that the callbacks to the client applications came at the requested rate and that we had new values for each of tags. As discussed earlier, we can not make this guarantee as long as OPC is "best effort" instead of "guaranteed real-time".

The DataCollector and all its accompanying functionality is implemented in the file DC.cs. An instance of this class is created when EasyIT Manager starts.

27.4 Implementing the application library and the application API

The application library is the framework that the client applications have at their disposal for communication with EasyIT. It is divided into two different parts. The application library is the code needed to do the actual communication with EasyIT and the API is the methods exposed on top of this for use by the application developers. The library is hidden for the application developers. All they have to do is include the ApplicationLibrary.dll as a reference to their applications.

When implementing the library we tried to always have as the main priority that developing applications for EasyIT should be easy and uncomplicated, while at the same time is should be fairly flexible. For communicating with the EasyIT Manager, we chose to use .NET Remoting. The reason for this is that we knew that applications would run as separate processes from the EasyIT Server. There are limited possibilities for separate processes to communicate with each other; the most common is to use either a shared pipe or some kind of socket programming. Both of these methods are fairly low level and would require large amounts of work built on top of them to be of use for us. That kind of work has already been implemented by many people before, so there is no need to re-invent the wheel over and over. Choosing to use names pipes to communicate between the processes would also restrict the applications to having to be run on the same machine as the EasyIT Manager. We don't need to know how exactly the data is transferred between the processes; we only need to know that it IS transferred without unreasonable amounts of overhead. .NET Remoting was added to the .NET framework for this exact purpose. It makes it possible to call methods on applications running on a remote computer as if it the applications were running locally. The idea is to implement whatever methods you want to execute on a server, declare them in an interface and publish on the server that you accept remote applications to call the methods in the interface. The client then asks the server for an instance of the interface and can call methods on this instance directly. All the underlying work like marshaling the arguments, communication over the network and getting the result back to the caller is done automatically by .NET.

For the application library the interface on the server is called IAppAPI and is declared in the file IAppAPI.cs. This file declares all the methods the application library can call on the EasyIT Manager. The code used by the application library to connect to EasyIT Manager is shown in figure 27.1. It is taken from the file AppAPIClientBase.cs in the ApplicationLibrary project. What actually happens is that we open a TcpChannel to the EasyIT Manager on a specified port and request an instance of an object that implements the interface called IAppAPI. For this to work the EasyIT Manager must be listening on the specified port and it must publish objects that implement the interface. The code for the server side part of the Remoting channel is shown in figure 27.2

```
BinaryClientFormatterSinkProvider clientProvider =
new BinaryClientFormatterSinkProvider();
BinaryServerFormatterSinkProvider serverProvider =
new BinaryServerFormatterSinkProvider();
serverProvider.TypeFilterLevel =
System.Runtime.Serialization.Formatters.TypeFilterLevel.Full;
TcpChannel chan = new TcpChannel(props, clientProvider, serverProvider);
ChannelServices.RegisterChannel(chan);
appAPI =(IAppAPI)Activator.GetObject(
typeof(IAppAPI),
"tcp://"+easyItURL+":"+easyItPort+"/AppAPI");
```

Figure 27.1: Code for connection to EasyIT through an application

We can see that the server publishes objects of type AppAPI, which implement the IAppAPI, on a new TcpChannel. This object, AppAPI, is the actual implementation of the application API. It can be found in the file AppAPI.cs.

A small detail that is very important to notice is that the TcpChannel needs to be based a value of "Full" for its typeFilterLever. This must be present if you plan to pass anything else then pure value types over .NET Remoting. As we intended to pass whole sets of tags, which are implemented in our own custom classes, we need to enable this feature. We chose to utilize the built in binary encoded tcp

```
IDictionary props = new Hashtable();
props["typeFilterLevel"] = "Full";
BinaryClientFormatterSinkProvider clientProvider =
new BinaryClientFormatterSinkProvider(null, null);
props["port"] = port;
BinaryServerFormatterSinkProvider formatterProvider =
new BinaryServerFormatterSinkProvider(props, null);
TcpChannel chan = new TcpChannel(props, clientProvider, formatterProvider);
ChannelServices.RegisterChannel(chan);
RemotingConfiguration.RegisterWellKnownServiceType(
Type.GetType("EasyIt.AppAPI",true),
"AppAPI",
WellKnownObjectMode.Singleton);
```

Figure 27.2: Code that listens for connections on the EasyIT Server

channels to do the actual communication. This is the most efficient setup, but one can easily switch to different types communication encoding. XML encoding over the HTTP protocol is one possibility that is extremely flexible when it comes to navigating through firewalls. This can be very useful if the application clients run on machines that are physically separated from the EasyIT Manager through one or more restrictive firewalls. It should however be noted that this will add extra overhead to the communication, a thorough analysis of the required throughput needed should be performed before taking such a step.

Now that the client application has gotten an instance of an object that implements IAppAPI, it can call methods on this object to communicate with the EasyIT Manager. The client calls the Register-method in IAppAPI to register itself. It sends with as a parameter another remote object that the EasyIT Manager can use to call the client back with new data. This object is implemented in the AppClient.cs and this class implements the IAppClient interface. At this time the client application can call EasyIT Manager and vice versa.

If we look at the IAppAPI interface, we see that all the methods needs to have the id of the application as an argument to the call. This is needed for EasyIT Manager to know which object called the method. In all likelihood multiple clients will call the EasyIT Manager through the remote interface at the same time, so we need to keep track of which one is which. We cannot keep state over a .NET Remoting channel, this is a design choice by the .NET Remoting team, so instead we must demand that the client identifies itself in every call to the server. This, however, is not very user friendly for the application developers. There should not be a need for them to pass the application id in every call they make to the server. To fix this we wrote wrappers for each of the methods in IAppAPI in the AppAPIClientBase-class. These wrappers are the methods the application developers calls. The wrapper then adds the application id and sends the call to the EasyIT Manager for processing.

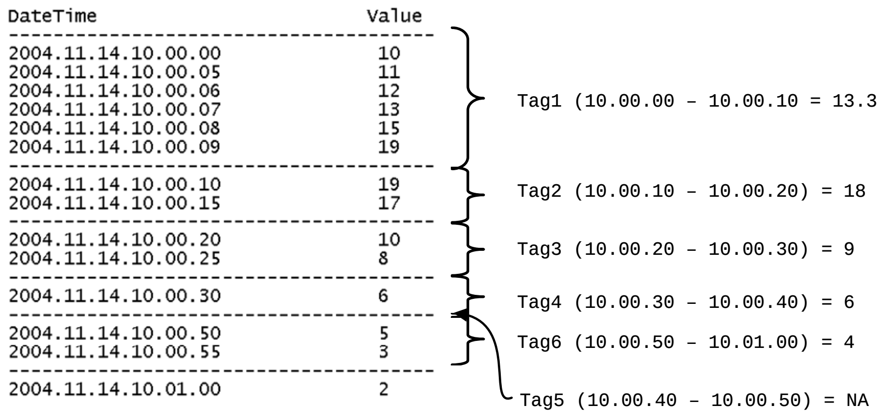
27.5 Implementing the GetTags method

The purpose of the `GetTags`-method is to create time trends based on logged data stored in the database. Obviously, the *Tag Browser* in the WEB part depends heavily on this method, but also applications might use it. Tags stored in the database may have been logged at different rates and the rate may even vary from time to time for one single tag. `GetTags` hides such variance and the users of the method can set the granularity of the requested time series to any value. The C# definition of the method is as follows:

```
GetTags(DateTime dtStart, DateTime dtEnd, CTag[] tags, int gran)
```

As the definition indicates, the method returns a `CSeries`-object representing the time series requested. The `dtStart` and `dtEnd` parameters defines the upper and lower limits of the time interval and the `tags` parameters declare what to include in the series. Please note that the `tags` parameter is an array, so it is possible to create trends for more than one tag in the same time interval. Finally, the `gran` parameter sets the level of granularity. This variable specifies the number of samples contained in the returned `CSeries` object. If it is set to 50, the series contains 50 samples for every tag in the series, set to 400 the series contains 400 samples. The samples are distributed evenly in the interval. So, if the interval spans 60 seconds and the granularity level is set to 30, the sample rate in this series is 2 seconds. Certainly, the desired number of samples may not match the actual number of samples in the database. The following paragraph explains how `GetTags` calculates the value of the samples in the series.

Assume a tag is logged in the database at different rates in the time interval spanning from 10.00.00 to 10.01.00 on November 14th 2004. The distribution is shown in figure 27.3. Certainly, this is no real logging, but it will illustrate how the `GetTag`-algorithm works quite well.



```
DateTime                          Value
--------------------------------------
2004.11.14.10.00.00               10
2004.11.14.10.00.05               11
2004.11.14.10.00.06               12
2004.11.14.10.00.07               13
2004.11.14.10.00.08               15
2004.11.14.10.00.09               19
--------------------------------------
2004.11.14.10.00.10               19
2004.11.14.10.00.15               17
--------------------------------------
2004.11.14.10.00.20               10
2004.11.14.10.00.25               8
--------------------------------------
2004.11.14.10.00.30               6
--------------------------------------
2004.11.14.10.00.50               5
2004.11.14.10.00.55               3
--------------------------------------
2004.11.14.10.01.00               2
```

Tag1 (10.00.00 – 10.00.10 = 13.3
Tag2 (10.00.10 – 10.00.20) = 18
Tag3 (10.00.20 – 10.00.30) = 9
Tag4 (10.00.30 – 10.00.40) = 6
Tag6 (10.00.50 – 10.01.00) = 4
Tag5 (10.00.40 – 10.00.50) = NA

Figure 27.3: Logged data

As we can see, the tag is first logged every second, then every fifth second. Suppose `GetTags` is called on this distribution with the time spanning from 10.00.00 to 10.01.00 and granularity set to 6. Thus, the six samples are separated by ten seconds. On the rightmost part of figure 27.3, the six samples are shown. The first sample, *Tag1* averages the values from 10.00.00 to 10.00.10. *Tag2* averages from 10.00.10 to 10.00.20 and so on. There are no samples available in 10.00.30 to 10.00.40 so *Tag5* is set to the value `NA` (not available).

So, in general the algorithm separates the time span in equaled sized intervals. Every sample returned from `GetTags` is the average of the logged values within each interval. If there are no tags within the interval, the sample is set to `NA`. `GetTags` uses an SQL *SELECT*-query to extract the logged values in the time span. As this operation is performed by the SQL-server, the complexity is unknown. The `GetTags`-algorithm then iterates through the resultset only once, yielding a linear complexity.

CHAPTER 28 EXTRA FUNCTIONALITY

Having started the implementation phase, we realized that the implementation was comming along quicker than expected. Our carefullness in planning the time had payed off, and we desided to add a couple of extra features to the system. This includes functionality not described in the requirements specification, but we think it adds to the quality and usefullness of the prototype. The following chapter describes these extra features.

28.1 MANAGE USER ACCOUNTS

This requirement was originally excluded from the implementation because of time considerations. The 'Configure users' option in the web menu was implemented as a part of the existing ConfigTool web-module. It simply lists users, and allows creating, editing and deleting users. This is only available for administrators.

28.2 SUBSCRIPTIONS

Users might often want to browse through data just as they arrive from OPC. In the system described in the requirements specification, you would have to set up a new application, and then write the code to make it subscribe to the tag to be able to browse them with the TagBrowser. To solve this inconvenience, we allow users to set up so called web subscriptions.

These subscriptions are set up by administrators, which can manage the sets of data currently being subscribed to. The underlying functionality is exactly that of the AppAPI's subscribe methods, only that these subscriptions don't need an application associated with them. They run in the background, and are startet each time the server starts.

28.3 "REAL-TIME" DATA BROWSING

In addition to browsing tags with the TagBrowser, we've added the opportunity to view the data in "real-time". This is accessed by selecting the tags and setting the desired granularity, and then clicking the 'Auto refresh' button in the TagBrowser. This is implemented simply by using a html meta-refresh tag, indicating how often the browser is to refresh the page. The same component as in the regular TagBrowser is used, and it shows the latest data in the time span given before hitting the 'Auto refresh' button.

28.4 SCHEDULING APPLICATIONS THROUGH THE WEB INTERFACE

Originally, the application monitor was designed only for viewing the state of the applications. We decided to add the possibility to start and stop the applications through the web interface. Start/Stop buttons appear in the web interface if an application is run locally. This assumes that the .exe file for the application has been copied to the correct location (see the user guide), and that the application has been configured with the correct name of the .exe file through the 'Configure applications' option in the web interface.

CHAPTER 29
FURTHER DEVELOPEMENT

This chapter is provided to gather all our ideas on how to improve the system if it was to be used in a real world application.

29.1 SUGGESTED IMPROVEMENTS

XML-parser for registering subscriptions
If you want to subscribe to tags in todays system, you need to enter them one by one through the 'Subscriptions' page in the web interface. This obviously is rather time consuming if you want to subscribe to more than a few tags. A soulution would be to use a tool like Matrikon's 'OPC Browser' to select all the tags you need, and output their names to an XML file. The EasyIT system could then just parse this XML file and start subscribing all the tags.

Tree structure for selecting tags for browsing
When selecting tags to browse you have to first select a server, then select a tag from another list box. This could be made easier by using a tree structure with servers as root nodes, containing all tags from that server as leaves. To simplify the process further a drag-and-drop interface could be used to allow users to drag the tags directly from the tree into the 'Selected tags' list box. Components like ComponentArt [6] have incorporated such drag-and-drop functionality in .NET components.

Algorithms concerning granularity of data
The algorithm used for adjusting granularity of the data to be browsed is quite straight foreward. If the granularity of the data is less than required, it simply leaves out the points not available. If the granularity is higher than required, the algorithm calculates the mean value of the available data in the time interval. In real life however, one might be interested in seeing for instance the spikes that we smooth out. Our system can easily be extended with customized algorithms for this in the EasyIT.getTags method, although the algorithms themselves might be very complex.

Built-in computations for the TagBrowser
Since all the data are readily available in the TagBrowser, it can be extended with some built-in functionality. This could include calulating average, minimum or maximum values, or performing regression analysis on the data being displayed.

Separate WebAPI for web programmers
Our system includes a common API to be used for both internal functionality, and for the functionality to be used by programmers developing custom web pages. Ideally this would be a separate WebAPI containing a subset of the current WebAPI functionality. Since this is a prototype, it's gathered in one class for simplicity.

Customize WebAPI functionality
Since we have limited information on exactly how programmers at ABB want to use the WebAPI, the available methods are very general. If many custom pages will require similar functionality, this might be included in the WebAPI to tailor it to their specific needs.

Validating user input
Our project was focused towards exploring the technology, rather than creating a robust solution. This explains our choice of not spending time on validating user input. In a real world application, a comprehensive scheme for validation should of course be implemented. Visual Studio and ASP.NET include built-in components for making sure forms are filled out correctly.

Improving security
The system uses a simple scheme for implementing security. This should be extended for instance to encrypt passwords. For any serious production use of the system where the users may access access EasyIT WEB from outside the local intranet, the entire web solution should be ported to fully encrypted communication channel. Technologies like HTTPS and SSL should be very easy it integrate with EasyIT Web for this purpose.

Error handling
The system should have systematic ways of catching errors, writing them to a log and notifying the user. Our system displays an interal error message not necessarily easily understood by a user. EasyIT Server works under the assumption that the .NET OPC framework we utilize does everything it should. It should handle errors from the OPC servers, lost connections and validate erroneous data. This is a luxury we can afford as a prototype, but in a production system substantial effort will have to be put in to making the system more robust when faced with external failure sources.

29.2 Estimating time consumption

This section will give a rough estimate as to how long time will be needed to implements the improvements suggested in the last section. Because many of the improvements are waguely described, an accurate estimation is difficult. Instead of estimating the exact number of hours spent, we have classified the improvements in three categories. Simple improvements are estimated to take under 25 hours to implement. Medium improvements are estimated between 25 and 50 hours. Complex improvements are estimated to take more than 50 hours. Table 29.1 categorizes the suggested improvements using this scheme.

Improvement	**Complexity**
XML-parser for registering subscriptions	Complex
Tree structure for selecting tags for browsing	Medium
Algorithms concerning granularity of data	Complex
Built-in computations for the TagBrowser	Simple
Separate WebAPI for web programmers	Simple
Customize WebAPI functionality	Medium
Validating user input	Medium
Improving security	Medium
Error handling	Simple

Table 29.1: Estimation of the complexity of suggested improvements

APPENDIX G
CODE SAMPLES

G.1 INTRODUCTION

This chapter shows how the AppAPI and WebAPI can be used to create simple applications.

G.2 EXAMPLE APPLICATIONS

Figure G.1 and figure G.3 are code listings for two sample applications.

G.3 SHELLS FOR CUSTOM WEB PAGES

The following code listings show how the files outputted by the EasyITWeb look like. These are taken from a page named 'MyPage' with file name 'mypage' under applcation 24. See listings G.6 and G.7

```
using System;
using EasyIt.DataItems;
using EasyIt.Application;

namespace Application
{
   /// <summary>
   /// This application checks if a specifed tag was over
   /// a certain limit and writes
   /// another tag both to the database and to a OPC server based on this.
   /// </summary>
   class PressureChecker
   {
     public PressureChecker()
      {
        const double PRESSURE_LIMIT = 150.0;

        //connect to EasyIT
        AppAPIClientBase b = new AppAPIClientBase(55, 6666, "localhost");

        //Define tag we want
        CTag tag = new CTag("opcda://testserver", "", "Pressure");
        CTag[] tags = new CTag[1];

        //Read tag directly from database.
        CSet result = b.ReadTagsFromOPC(tags);

        //figure out if pressure was over limit
        double val = (double) result[0].Value;
        bool overLimit;
        if(val > PRESSURE_LIMIT)
        {
           overLimit = true;
        }
        else
        {
           overLimit = false;
        }
```

Continues on next page

Figure G.1: The pressure checker sample application page 1

Continued from previous page

```
        //Define new tag to write
        CTag writeTag = new CTag("opcda://anothertestserver", "",
                                 "PressureStatus");
        writeTag.Timestamp = DateTime.Now;
        writeTag.Value = overLimit;
        writeTag.DataType = EasyItDataType.boolType;
        writeTag.Quality = 1;
        tags = new CTag[1];
        tags[0] = writeTag;

        //write tag to database
        b.WriteTagsToDB(tags);
        //write tag to opc
        b.WriteTagsToOPC(tags);
      }

      [STAThread]
      static void Main(string[] args)
      {
        new PressureChecker();
      }
   }
}
```

Figure G.2: The pressure checker sample application page 2

```
using System;
using System.Collections;
using System.Threading;
using EasyIt.DataItems;
using EasyIt.Application;
using EasyIt.RemoteInterfaces;

namespace Applications
{
   /// <summary>
   /// This sample application subscribes to two different tags at a rate of
   /// 3 seconds.
   /// It reads a config value from the database "averagesRequested". It then
   /// calculates as many averages from the data it gets through the callback
   /// as the config value says. After that it sleeps for one hour and repeats
   /// everything from the start.
   /// </summary>
   public class Averager
  {
    int averagesRequested;
    int averagesCalculated= 0;
    EasyIt.Application.AppAPIClientBase b;
    bool subscribe = true;
    long setID;

    public Averager()
    {
      //connect to EasyIT
      b = new AppAPIClientBase(23, 6666, "localhost");
       //define event listener, we want to handle data
      b.DataArrival += new DataReceivedEventHandler(b_DataArrival);
       //read config pair from database
      averagesRequested = Convert.ToInt32(b.GetConfigValue("averagesRequested"));

      //write config pair to database. Write startup time.
      b.SetConfigValue("lastStartupDateTime", DateTime.Now.ToString());

      //Define tags we want
      CTag tag = new CTag("opcda://localhost/DSxPOpcSimulator.TSxOpcSimulator.1",
                          "", "Simulation Items.Real.Real_01");
      CTag tag2 = new CTag("opcda://localhost/DSxPOpcSimulator.TSxOpcSimulator.1"
                           , "", "Simulation Items.Real.Real_02");

      //Create a subscrition set with rate 3
      CSet setm = new CSet("testSet", 3);
      setm.AddTag(tag);
      setm.AddTag(tag2);
```

Continues on next page

Figure G.3: The averager sample application page 1

Continued from previous page

```
    //run forever, all further data handling is done in the DataArrival handler.
    while(true)
    {
      //subscribe to set
      subscribe = true;
      averagesCalculated = 0;
      setID = b.SubscribeSet(setm);
      //sleep this thread until we have calculated enough averages
      //in the callback method
      while(subscribe)
      {
        Thread.Sleep(5000);
      }
      //we have calculated the requested number of averages, let’s unsubscribe,
      //wait an hour and repeat
      b.DataArrival -= new DataReceivedEventHandler(b_DataArrival);
      b.UnsubscribeSet(setID);
      Thread.Sleep(1000 * 60 * 60);
      }
  }
```

Continues on next page

Figure G.4: The averager sample application page 2

Continued from previous page

```
    private void b_DataArrival(EasyIt.DataItems.CSet setm)
    {
       CTag tag1 = setm.Tags[0] as CTag;
       CTag tag2 = setm.Tags[1] as CTag;
       //get values
       double d1 = (double)tag1.Value;
       double d2 = (double)tag2.Value;

       //calculate average
       object average = (double) (d1 + d2)/2;

       //Define tag to write to database
       CTag[] tags = new CTag[1];

       CTag tag = new CTag();
       tag.Path = "";
       tag.Name = "Average";
       tag.Timestamp = DateTime.Now;
       tag.Value = average;
       tag.DataType = EasyItDataType.doubleType;
       tag.Quality = 1;
       tag.Deadband = 0.0f;
       tag.Server = "Average application";
       tags[0] = tag;

       //write new tag to database.
       b.WriteTagsToDB(tags);

       //check if we have calculated enough averages.
       if(averagesCalculated > averagesRequested)
       {
         subscribe = false;
       }
     }


     [STAThread]
     static void Main()
     {
       new Averager();
     }
 }
```

Figure G.5: The averager sample application page 3

```
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="mypage.ascx.cs"
Inherits="WEB.modules.Custom.App24.mypage" Src="mypage.ascx.cs"
TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
<h1>Under construction!</h1><br>
Please edit the file modules/Custom/App24/mypage.ascx in order to
customize this page.
```

Figure G.6: Code shell for mypage.ascx 1

```
namespace WEB.modules.Custom.App24
{
  using System;
  using System.Data;
  using System.Web;

  public class mypage : EasyITControl
  {

    public mypage()
    {
      PageTitle = "MyPage";
    }

    private void Page_Load(object sender, System.EventArgs e)
    {
      WebAPI webapi = (WebAPI)Application["webapi"];
    }

    #region Web Form Designer generated code
    override protected void OnInit(EventArgs e)
    {
      InitializeComponent();
      base.OnInit(e);
    }

    private void InitializeComponent()
    {
      this.Load += new System.EventHandler(this.Page_Load);
    }
    #endregion
  }
}
```

Figure G.7: Code shell for mypage.ascx.cs 2

Appendix H
Class diagram

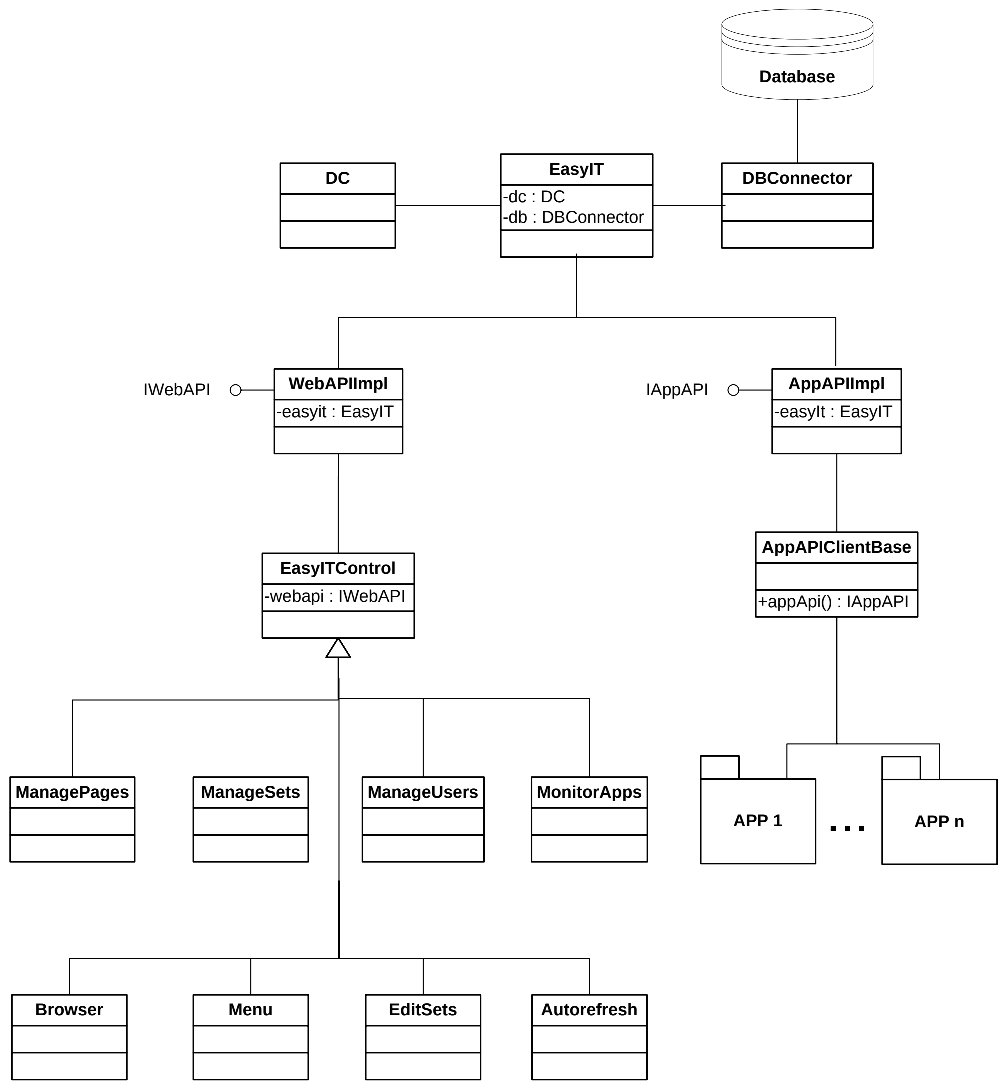
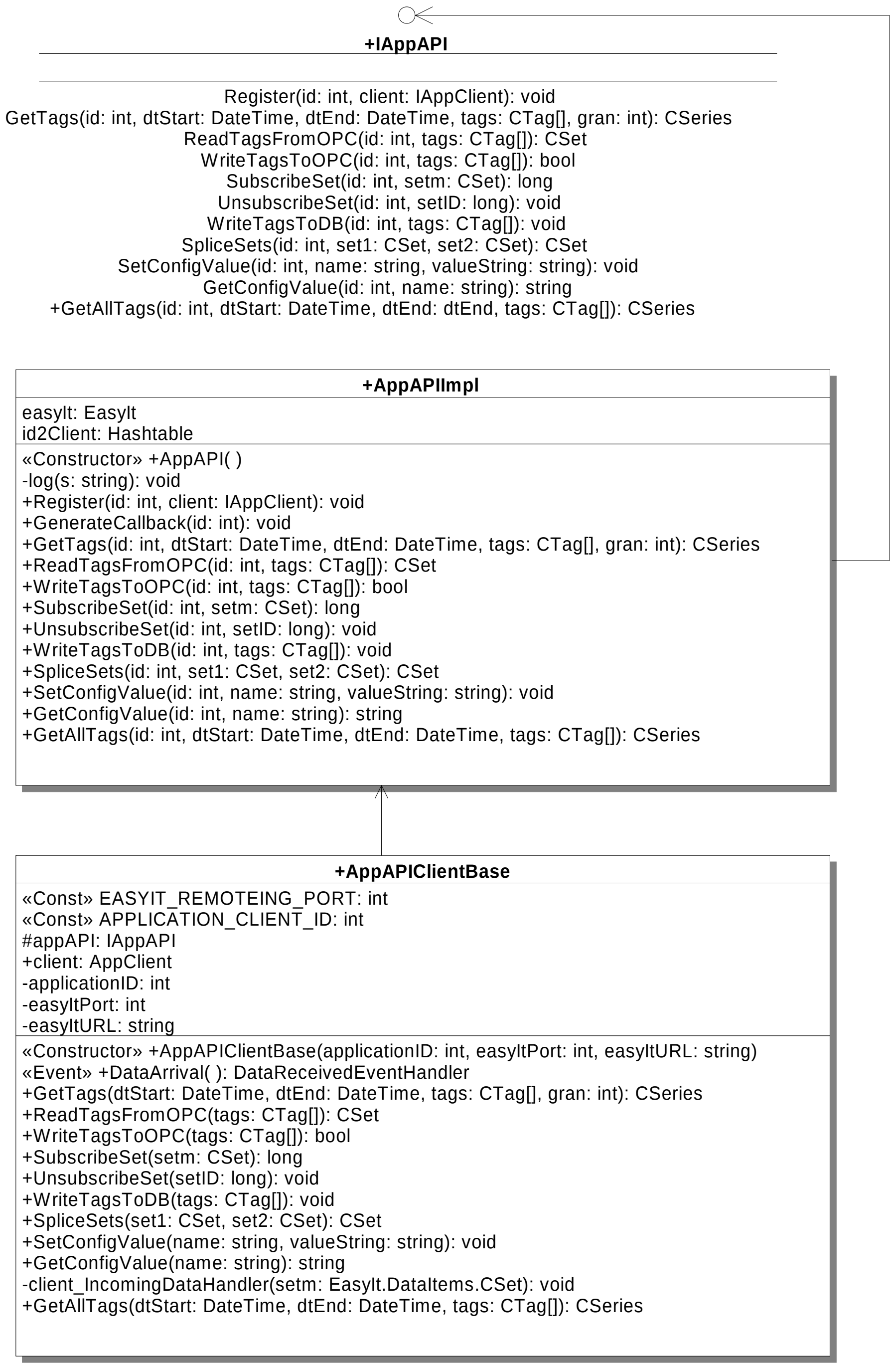


Figure H.1: Overall class diagram



```
+IAppAPI

Register(id: int, client: IAppClient): void
GetTags(id: int, dtStart: DateTime, dtEnd: DateTime, tags: CTag[], gran: int): CSeries
ReadTagsFromOPC(id: int, tags: CTag[]): CSet
WriteTagsToOPC(id: int, tags: CTag[]): bool
SubscribeSet(id: int, setm: CSet): long
UnsubscribeSet(id: int, setID: long): void
WriteTagsToDB(id: int, tags: CTag[]): void
SpliceSets(id: int, set1: CSet, set2: CSet): CSet
SetConfigValue(id: int, name: string, valueString: string): void
GetConfigValue(id: int, name: string): string
+GetAllTags(id: int, dtStart: DateTime, dtEnd: dtEnd, tags: CTag[]): CSeries
```

```
+AppAPIImpl
easyIt: EasyIt
id2Client: Hashtable
«Constructor» +AppAPI( )
-log(s: string): void
+Register(id: int, client: IAppClient): void
+GenerateCallback(id: int): void
+GetTags(id: int, dtStart: DateTime, dtEnd: DateTime, tags: CTag[], gran: int): CSeries
+ReadTagsFromOPC(id: int, tags: CTag[]): CSet
+WriteTagsToOPC(id: int, tags: CTag[]): bool
+SubscribeSet(id: int, setm: CSet): long
+UnsubscribeSet(id: int, setID: long): void
+WriteTagsToDB(id: int, tags: CTag[]): void
+SpliceSets(id: int, set1: CSet, set2: CSet): CSet
+SetConfigValue(id: int, name: string, valueString: string): void
+GetConfigValue(id: int, name: string): string
+GetAllTags(id: int, dtStart: DateTime, dtEnd: DateTime, tags: CTag[]): CSeries
```

```
+AppAPIClientBase
«Const» EASYIT_REMOTEING_PORT: int
«Const» APPLICATION_CLIENT_ID: int
#appAPI: IAppAPI
+client: AppClient
-applicationID: int
-easyItPort: int
-easyItURL: string
«Constructor» +AppAPIClientBase(applicationID: int, easyItPort: int, easyItURL: string)
«Event» +DataArrival( ): DataReceivedEventHandler
+GetTags(dtStart: DateTime, dtEnd: DateTime, tags: CTag[], gran: int): CSeries
+ReadTagsFromOPC(tags: CTag[]): CSet
+WriteTagsToOPC(tags: CTag[]): bool
+SubscribeSet(setm: CSet): long
+UnsubscribeSet(setID: long): void
+WriteTagsToDB(tags: CTag[]): void
+SpliceSets(set1: CSet, set2: CSet): CSet
+SetConfigValue(name: string, valueString: string): void
+GetConfigValue(name: string): string
-client_IncomingDataHandler(setm: EasyIt.DataItems.CSet): void
+GetAllTags(dtStart: DateTime, dtEnd: DateTime, tags: CTag[]): CSeries
```

Figure H.2: Application related classes and interfaces

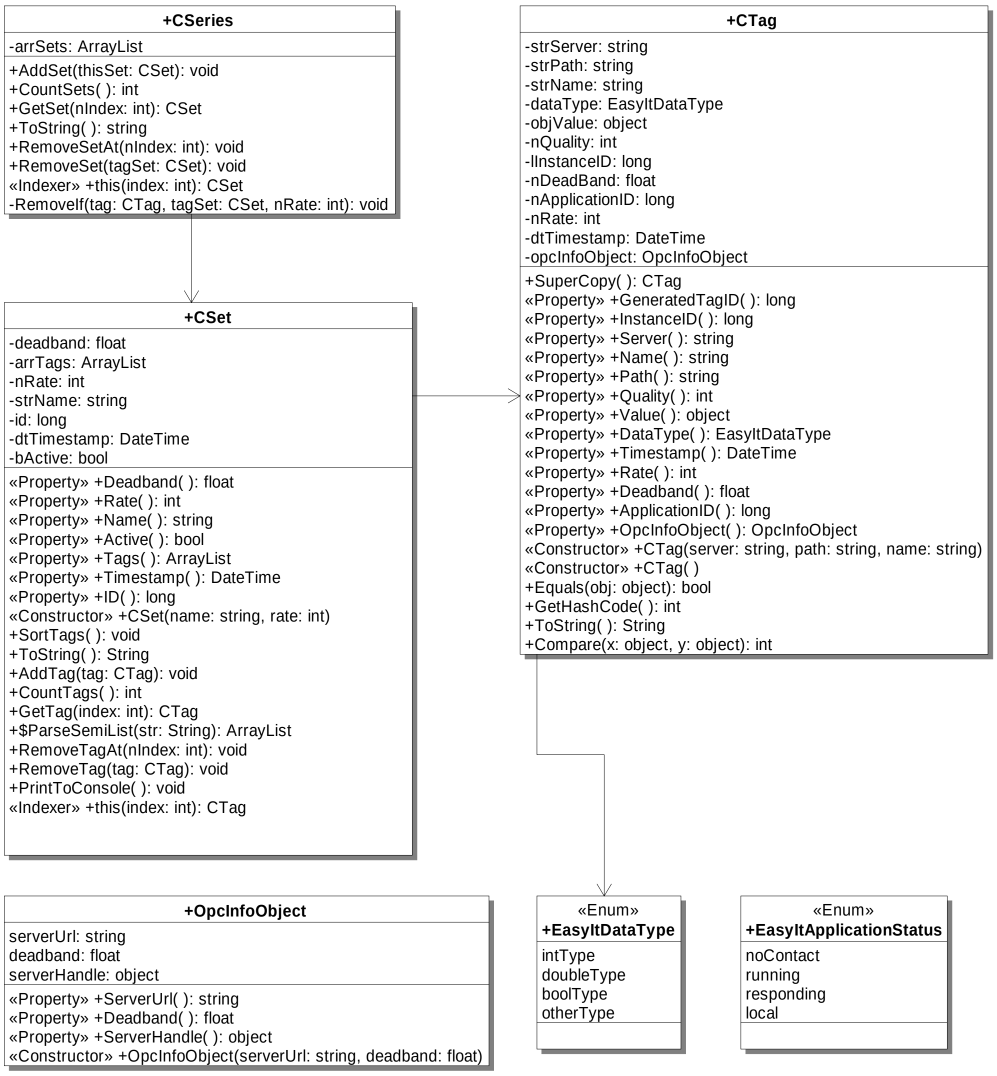
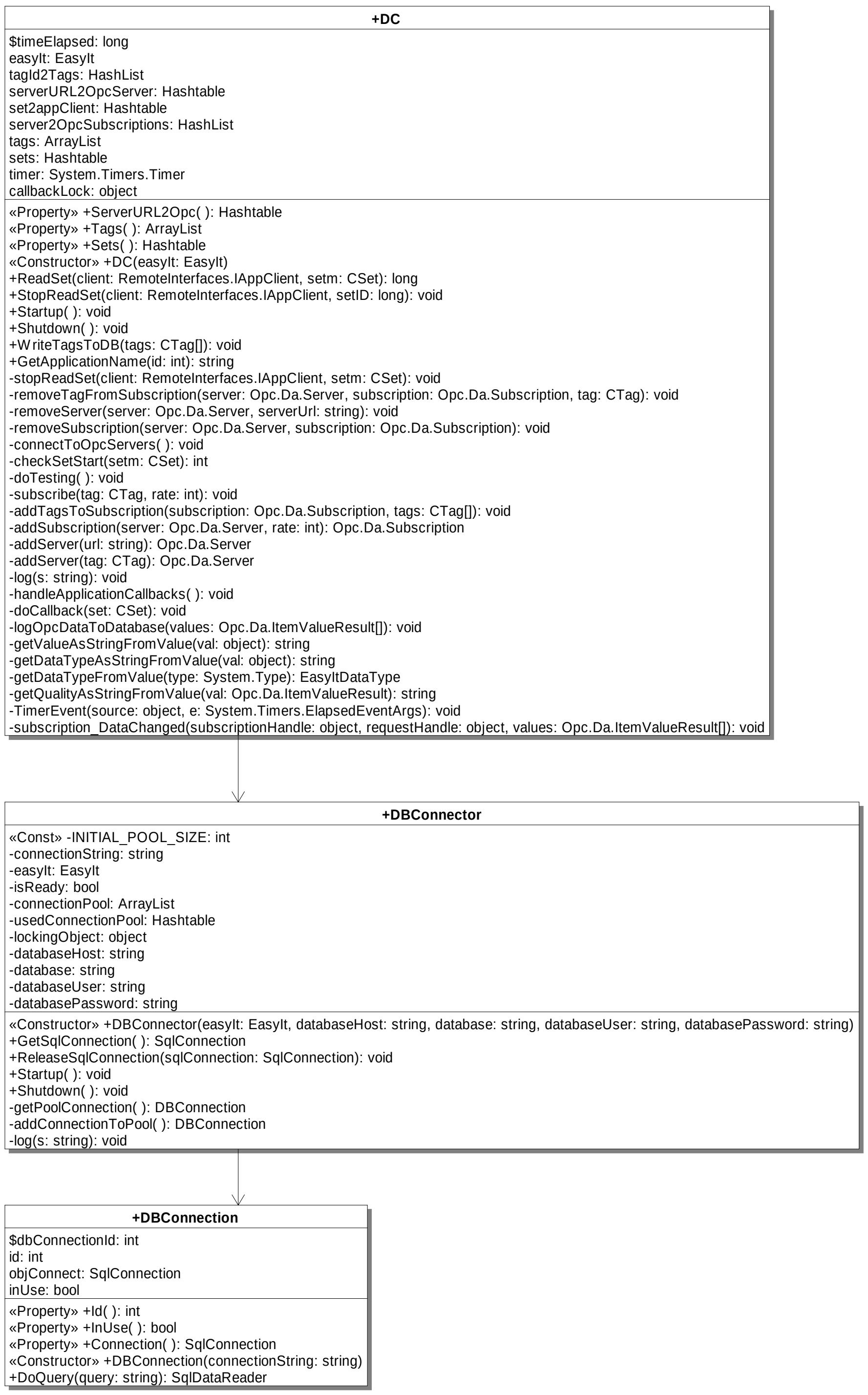

+CSeries
-arrSets: ArrayList
+AddSet(thisSet: CSet): void
+CountSets( ): int
+GetSet(nIndex: int): CSet
+ToString( ): string
+RemoveSetAt(nIndex: int): void
+RemoveSet(tagSet: CSet): void
«Indexer» +this(index: int): CSet
-RemoveIf(tag: CTag, tagSet: CSet, nRate: int): void
+CTag
-strServer: string
-strPath: string
-strName: string
-dataType: EasyItDataType
-objValue: object
-nQuality: int
-lInstanceID: long
-nDeadBand: float
-nApplicationID: long
-nRate: int
-dtTimestamp: DateTime
-opcInfoObject: OpcInfoObject
+SuperCopy( ): CTag
«Property» +GeneratedTagID( ): long
«Property» +InstanceID( ): long
«Property» +Server( ): string
«Property» +Name( ): string
«Property» +Path( ): string
«Property» +Quality( ): int
«Property» +Value( ): object
«Property» +DataType( ): EasyItDataType
«Property» +Timestamp( ): DateTime
«Property» +Rate( ): int
«Property» +Deadband( ): float
«Property» +ApplicationID( ): long
«Property» +OpcInfoObject( ): OpcInfoObject
«Constructor» +CTag(server: string, path: string, name: string)
«Constructor» +CTag( )
+Equals(obj: object): bool
+GetHashCode( ): int
+ToString( ): String
+Compare(x: object, y: object): int
+CSet
-deadband: float
-arrTags: ArrayList
-nRate: int
-strName: string
-id: long
-dtTimestamp: DateTime
-bActive: bool
«Property» +Deadband( ): float
«Property» +Rate( ): int
«Property» +Name( ): string
«Property» +Active( ): bool
«Property» +Tags( ): ArrayList
«Property» +Timestamp( ): DateTime
«Property» +ID( ): long
«Constructor» +CSet(name: string, rate: int)
+SortTags( ): void
+ToString( ): String
+AddTag(tag: CTag): void
+CountTags( ): int
+GetTag(index: int): CTag
+$ParseSemiList(str: String): ArrayList
+RemoveTagAt(nIndex: int): void
+RemoveTag(tag: CTag): void
+PrintToConsole( ): void
«Indexer» +this(index: int): CTag
+OpcInfoObject
serverUrl: string
deadband: float
serverHandle: object
«Property» +ServerUrl( ): string
«Property» +Deadband( ): float
«Property» +ServerHandle( ): object
«Constructor» +OpcInfoObject(serverUrl: string, deadband: float)
«Enum»
+EasyItDataType
intType
doubleType
boolType
otherType
«Enum»
+EasyItApplicationStatus
noContact
running
responding
local

Figure H.3: Data item classes

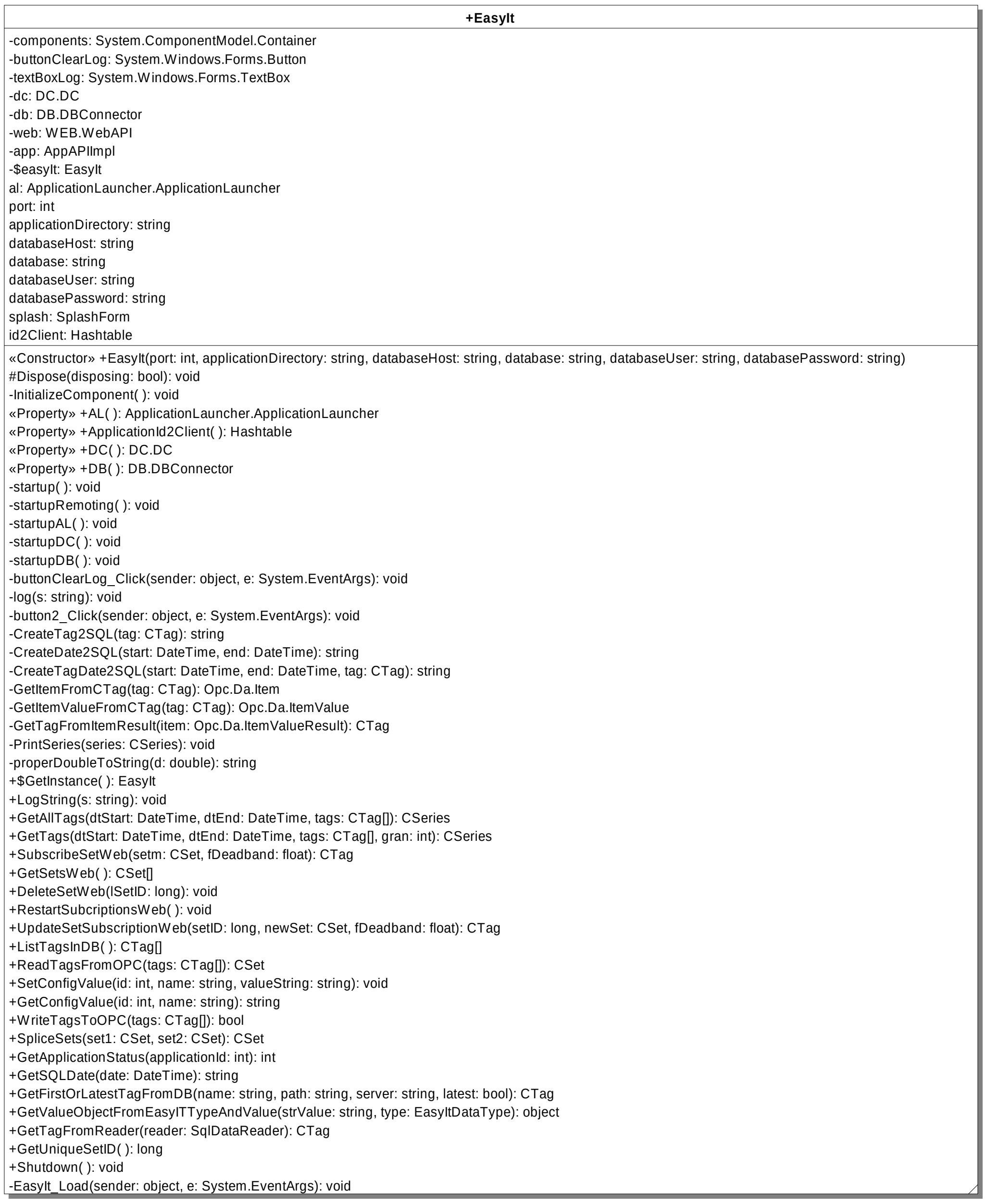


```
+DC
$timeElapsed: long
easyIt: EasyIt
tagId2Tags: HashList
serverURL2OpcServer: Hashtable
set2appClient: Hashtable
server2OpcSubscriptions: HashList
tags: ArrayList
sets: Hashtable
timer: System.Timers.Timer
callbackLock: object
«Property» +ServerURL2Opc( ): Hashtable
«Property» +Tags( ): ArrayList
«Property» +Sets( ): Hashtable
«Constructor» +DC(easyIt: EasyIt)
+ReadSet(client: RemoteInterfaces.IAppClient, setm: CSet): long
+StopReadSet(client: RemoteInterfaces.IAppClient, setID: long): void
+Startup( ): void
+Shutdown( ): void
+WriteTagsToDB(tags: CTag[]): void
+GetApplicationName(id: int): string
-stopReadSet(client: RemoteInterfaces.IAppClient, setm: CSet): void
-removeTagFromSubscription(server: Opc.Da.Server, subscription: Opc.Da.Subscription, tag: CTag): void
-removeServer(server: Opc.Da.Server, serverUrl: string): void
-removeSubscription(server: Opc.Da.Server, subscription: Opc.Da.Subscription): void
-connectToOpcServers( ): void
-checkSetStart(setm: CSet): int
-doTesting( ): void
-subscribe(tag: CTag, rate: int): void
-addTagsToSubscription(subscription: Opc.Da.Subscription, tags: CTag[]): void
-addSubscription(server: Opc.Da.Server, rate: int): Opc.Da.Subscription
-addServer(url: string): Opc.Da.Server
-addServer(tag: CTag): Opc.Da.Server
-log(s: string): void
-handleApplicationCallbacks( ): void
-doCallback(set: CSet): void
-logOpcDataToDatabase(values: Opc.Da.ItemValueResult[]): void
-getValueAsStringFromValue(val: object): string
-getDataTypeAsStringFromValue(val: object): string
-getDataTypeFromValue(type: System.Type): EasyItDataType
-getQualityAsStringFromValue(val: Opc.Da.ItemValueResult): string
-TimerEvent(source: object, e: System.Timers.ElapsedEventArgs): void
-subscription_DataChanged(subscriptionHandle: object, requestHandle: object, values: Opc.Da.ItemValueResult[]): void
```

```
+DBConnector
«Const» -INITIAL_POOL_SIZE: int
-connectionString: string
-easyIt: EasyIt
-isReady: bool
-connectionPool: ArrayList
-usedConnectionPool: Hashtable
-lockingObject: object
-databaseHost: string
-database: string
-databaseUser: string
-databasePassword: string
«Constructor» +DBConnector(easyIt: EasyIt, databaseHost: string, database: string, databaseUser: string, databasePassword: string)
+GetSqlConnection( ): SqlConnection
+ReleaseSqlConnection(sqlConnection: SqlConnection): void
+Startup( ): void
+Shutdown( ): void
-getPoolConnection( ): DBConnection
-addConnectionToPool( ): DBConnection
-log(s: string): void
```

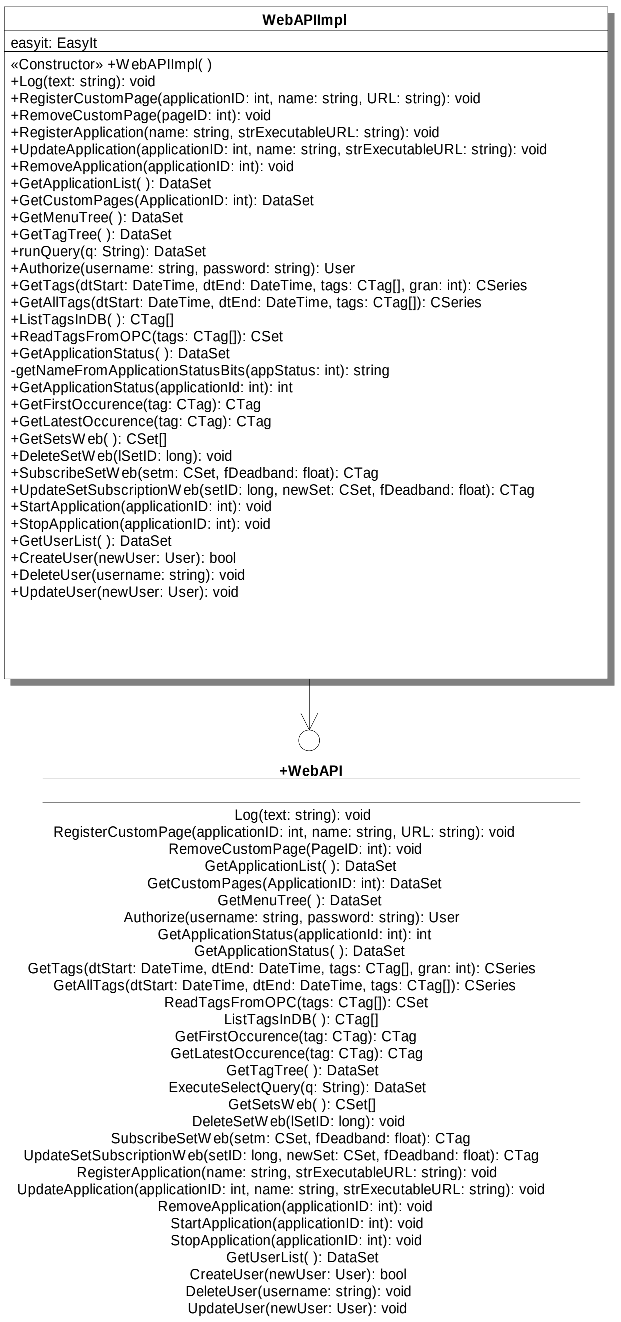
```
+DBConnection
$dbConnectionId: int
id: int
objConnect: SqlConnection
inUse: bool
«Property» +Id( ): int
«Property» +InUse( ): bool
«Property» +Connection( ): SqlConnection
«Constructor» +DBConnection(connectionString: string)
+DoQuery(query: string): SqlDataReader
```

Figure H.4: DBConnector and DataCollector(DC) classes



```
+EasyIt
-components: System.ComponentModel.Container
-buttonClearLog: System.Windows.Forms.Button
-textBoxLog: System.Windows.Forms.TextBox
-dc: DC.DC
-db: DB.DBConnector
-web: WEB.WebAPI
-app: AppAPIImpl
-$easyIt: EasyIt
al: ApplicationLauncher.ApplicationLauncher
port: int
applicationDirectory: string
databaseHost: string
database: string
databaseUser: string
databasePassword: string
splash: SplashForm
id2Client: Hashtable
«Constructor» +EasyIt(port: int, applicationDirectory: string, databaseHost: string, database: string, databaseUser: string, databasePassword: string)
#Dispose(disposing: bool): void
-InitializeComponent( ): void
«Property» +AL( ): ApplicationLauncher.ApplicationLauncher
«Property» +ApplicationId2Client( ): Hashtable
«Property» +DC( ): DC.DC
«Property» +DB( ): DB.DBConnector
-startup( ): void
-startupRemoting( ): void
-startupAL( ): void
-startupDC( ): void
-startupDB( ): void
-buttonClearLog_Click(sender: object, e: System.EventArgs): void
-log(s: string): void
-button2_Click(sender: object, e: System.EventArgs): void
-CreateTag2SQL(tag: CTag): string
-CreateDate2SQL(start: DateTime, end: DateTime): string
-CreateTagDate2SQL(start: DateTime, end: DateTime, tag: CTag): string
-GetItemFromCTag(tag: CTag): Opc.Da.Item
-GetItemValueFromCTag(tag: CTag): Opc.Da.ItemValue
-GetTagFromItemResult(item: Opc.Da.ItemValueResult): CTag
-PrintSeries(series: CSeries): void
-properDoubleToString(d: double): string
+$GetInstance( ): EasyIt
+LogString(s: string): void
+GetAllTags(dtStart: DateTime, dtEnd: DateTime, tags: CTag[]): CSeries
+GetTags(dtStart: DateTime, dtEnd: DateTime, tags: CTag[], gran: int): CSeries
+SubscribeSetWeb(setm: CSet, fDeadband: float): CTag
+GetSetsWeb( ): CSet[]
+DeleteSetWeb(lSetID: long): void
+RestartSubcriptionsWeb( ): void
+UpdateSetSubscriptionWeb(setID: long, newSet: CSet, fDeadband: float): CTag
+ListTagsInDB( ): CTag[]
+ReadTagsFromOPC(tags: CTag[]): CSet
+SetConfigValue(id: int, name: string, valueString: string): void
+GetConfigValue(id: int, name: string): string
+WriteTagsToOPC(tags: CTag[]): bool
+SpliceSets(set1: CSet, set2: CSet): CSet
+GetApplicationStatus(applicationId: int): int
+GetSQLDate(date: DateTime): string
+GetFirstOrLatestTagFromDB(name: string, path: string, server: string, latest: bool): CTag
+GetValueObjectFromEasyITTypeAndValue(strValue: string, type: EasyItDataType): object
+GetTagFromReader(reader: SqlDataReader): CTag
+GetUniqueSetID( ): long
+Shutdown( ): void
-EasyIt_Load(sender: object, e: System.EventArgs): void
```

Figure H.5: EasyIT Server and Manager classes



```
WebAPIImpl
easyit: EasyIt
«Constructor» +WebAPIImpl( )
+Log(text: string): void
+RegisterCustomPage(applicationID: int, name: string, URL: string): void
+RemoveCustomPage(pageID: int): void
+RegisterApplication(name: string, strExecutableURL: string): void
+UpdateApplication(applicationID: int, name: string, strExecutableURL: string): void
+RemoveApplication(applicationID: int): void
+GetApplicationList( ): DataSet
+GetCustomPages(ApplicationID: int): DataSet
+GetMenuTree( ): DataSet
+GetTagTree( ): DataSet
+runQuery(q: String): DataSet
+Authorize(username: string, password: string): User
+GetTags(dtStart: DateTime, dtEnd: DateTime, tags: CTag[], gran: int): CSeries
+GetAllTags(dtStart: DateTime, dtEnd: DateTime, tags: CTag[]): CSeries
+ListTagsInDB( ): CTag[]
+ReadTagsFromOPC(tags: CTag[]): CSet
+GetApplicationStatus( ): DataSet
-getNameFromApplicationStatusBits(appStatus: int): string
+GetApplicationStatus(applicationId: int): int
+GetFirstOccurence(tag: CTag): CTag
+GetLatestOccurence(tag: CTag): CTag
+GetSetsWeb( ): CSet[]
+DeleteSetWeb(lSetID: long): void
+SubscribeSetWeb(setm: CSet, fDeadband: float): CTag
+UpdateSetSubscriptionWeb(setID: long, newSet: CSet, fDeadband: float): CTag
+StartApplication(applicationID: int): void
+StopApplication(applicationID: int): void
+GetUserList( ): DataSet
+CreateUser(newUser: User): bool
+DeleteUser(username: string): void
+UpdateUser(newUser: User): void
```

```
+WebAPI
Log(text: string): void
RegisterCustomPage(applicationID: int, name: string, URL: string): void
RemoveCustomPage(PageID: int): void
GetApplicationList( ): DataSet
GetCustomPages(ApplicationID: int): DataSet
GetMenuTree( ): DataSet
Authorize(username: string, password: string): User
GetApplicationStatus(applicationId: int): int
GetApplicationStatus( ): DataSet
GetTags(dtStart: DateTime, dtEnd: DateTime, tags: CTag[], gran: int): CSeries
GetAllTags(dtStart: DateTime, dtEnd: DateTime, tags: CTag[]): CSeries
ReadTagsFromOPC(tags: CTag[]): CSet
ListTagsInDB( ): CTag[]
GetFirstOccurence(tag: CTag): CTag
GetLatestOccurence(tag: CTag): CTag
GetTagTree( ): DataSet
ExecuteSelectQuery(q: String): DataSet
GetSetsWeb( ): CSet[]
DeleteSetWeb(lSetID: long): void
SubscribeSetWeb(setm: CSet, fDeadband: float): CTag
UpdateSetSubscriptionWeb(setID: long, newSet: CSet, fDeadband: float): CTag
RegisterApplication(name: string, strExecutableURL: string): void
UpdateApplication(applicationID: int, name: string, strExecutableURL: string): void
RemoveApplication(applicationID: int): void
StartApplication(applicationID: int): void
StopApplication(applicationID: int): void
GetUserList( ): DataSet
CreateUser(newUser: User): bool
DeleteUser(username: string): void
UpdateUser(newUser: User): void
```

Figure H.6: Web related classes and interfaces on the EasyIt Manager

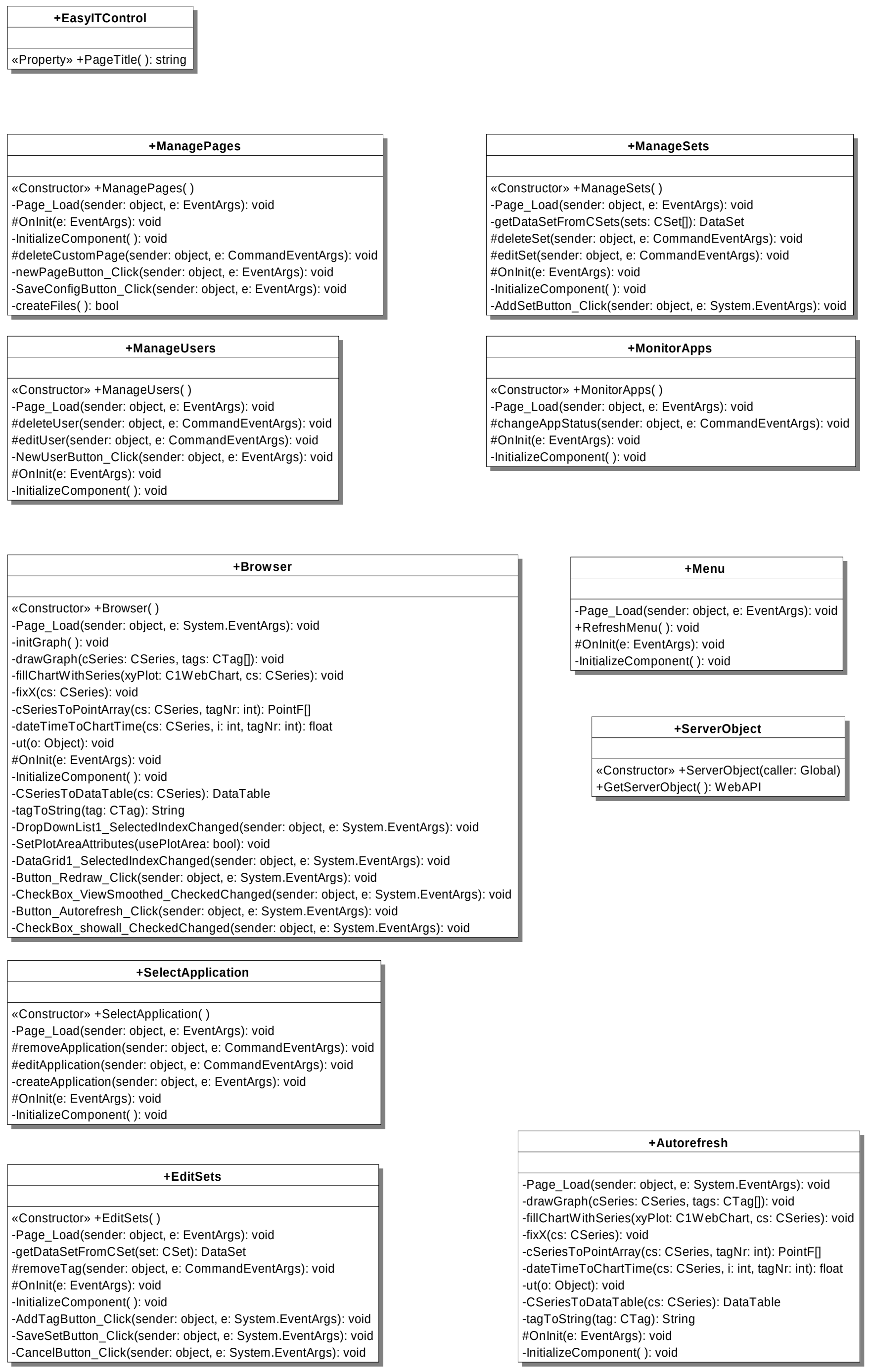

+EasyITControl
«Property» +PageTitle( ): string
+ManagePages
«Constructor» +ManagePages( )
-Page_Load(sender: object, e: EventArgs): void
#OnInit(e: EventArgs): void
-InitializeComponent( ): void
#deleteCustomPage(sender: object, e: CommandEventArgs): void
-newPageButton_Click(sender: object, e: EventArgs): void
-SaveConfigButton_Click(sender: object, e: EventArgs): void
-createFiles( ): bool
+ManageSets
«Constructor» +ManageSets( )
-Page_Load(sender: object, e: EventArgs): void
-getDataSetFromCSets(sets: CSet[]): DataSet
#deleteSet(sender: object, e: CommandEventArgs): void
#editSet(sender: object, e: CommandEventArgs): void
#OnInit(e: EventArgs): void
-InitializeComponent( ): void
-AddSetButton_Click(sender: object, e: System.EventArgs): void
+ManageUsers
«Constructor» +ManageUsers( )
-Page_Load(sender: object, e: EventArgs): void
#deleteUser(sender: object, e: CommandEventArgs): void
#editUser(sender: object, e: CommandEventArgs): void
-NewUserButton_Click(sender: object, e: EventArgs): void
#OnInit(e: EventArgs): void
-InitializeComponent( ): void
+MonitorApps
«Constructor» +MonitorApps( )
-Page_Load(sender: object, e: EventArgs): void
#changeAppStatus(sender: object, e: CommandEventArgs): void
#OnInit(e: EventArgs): void
-InitializeComponent( ): void
+Browser
«Constructor» +Browser( )
-Page_Load(sender: object, e: System.EventArgs): void
-initGraph( ): void
-drawGraph(cSeries: CSeries, tags: CTag[]): void
-fillChartWithSeries(xyPlot: C1WebChart, cs: CSeries): void
-fixX(cs: CSeries): void
-cSeriesToPointArray(cs: CSeries, tagNr: int): PointF[]
-dateTimeToChartTime(cs: CSeries, i: int, tagNr: int): float
-ut(o: Object): void
#OnInit(e: EventArgs): void
-InitializeComponent( ): void
-CSeriesToDataTable(cs: CSeries): DataTable
-tagToString(tag: CTag): String
-DropDownList1_SelectedIndexChanged(sender: object, e: System.EventArgs): void
-SetPlotAreaAttributes(usePlotArea: bool): void
-DataGrid1_SelectedIndexChanged(sender: object, e: System.EventArgs): void
-Button_Redraw_Click(sender: object, e: System.EventArgs): void
-CheckBox_ViewSmoothed_CheckedChanged(sender: object, e: System.EventArgs): void
-Button_Autorefresh_Click(sender: object, e: System.EventArgs): void
-CheckBox_showall_CheckedChanged(sender: object, e: System.EventArgs): void
+Menu
-Page_Load(sender: object, e: EventArgs): void
+RefreshMenu( ): void
#OnInit(e: EventArgs): void
-InitializeComponent( ): void
+ServerObject
«Constructor» +ServerObject(caller: Global)
+GetServerObject( ): WebAPI
+SelectApplication
«Constructor» +SelectApplication( )
-Page_Load(sender: object, e: EventArgs): void
#removeApplication(sender: object, e: CommandEventArgs): void
#editApplication(sender: object, e: CommandEventArgs): void
-createApplication(sender: object, e: EventArgs): void
#OnInit(e: EventArgs): void
-InitializeComponent( ): void
+Autorefresh
-Page_Load(sender: object, e: System.EventArgs): void
-drawGraph(cSeries: CSeries, tags: CTag[]): void
-fillChartWithSeries(xyPlot: C1WebChart, cs: CSeries): void
-fixX(cs: CSeries): void
-cSeriesToPointArray(cs: CSeries, tagNr: int): PointF[]
-dateTimeToChartTime(cs: CSeries, i: int, tagNr: int): float
-ut(o: Object): void
-CSeriesToDataTable(cs: CSeries): DataTable
-tagToString(tag: CTag): String
#OnInit(e: EventArgs): void
-InitializeComponent( ): void
+EditSets
«Constructor» +EditSets( )
-Page_Load(sender: object, e: EventArgs): void
-getDataSetFromCSet(set: CSet): DataSet
#removeTag(sender: object, e: CommandEventArgs): void
#OnInit(e: EventArgs): void
-InitializeComponent( ): void
-AddTagButton_Click(sender: object, e: System.EventArgs): void
-SaveSetButton_Click(sender: object, e: System.EventArgs): void
-CancelButton_Click(sender: object, e: System.EventArgs): void

Figure H.7: Web related classes on the EasyIT Web

Part VI

Test Document

Chapter 30 Introduction

30.1 Purpose

The purpose of this part is to document how we have planned and performed testing of the requirements stated in the requirements specification.

This document was commenced in the Requirements Specification phase, and has evolved throughout the Software Design Description and Implementation phases. When the latter phase was finished, we performed the tests described in Appendix I. The overall testing objectives are:

- Find irregularities in the behavior of the system, as early as possible.
- Verify that specified requirements are met.
- Verify that existing and new processes can be performed as intended.
- Verify that all modules in the system work as an integrated system.
- Show the customer that the product has the agreed functionality and quality.

30.2 Overview

This document is divided into 4 chapters. The following chapter describes the overall test plan for the system. This chapter forms the foundation for all testing procedures performed on the system. Chapter 32 is a collection of templates used in the test process. In the summary, chapter 33, we will summarize how the different testing procedures were performed, including the results from these tests, before the summary provides and conclusion of the tests performed and the system's performance in general.

This document includes two appendixes: Appendix I contains the test report, and Appendix J contains the test log. The aforementioned summary (chapter 33) draws its conclusion from these two appendices,

Chapter 31
Overall Test plan

31.1 Introduction

This chapter describes the overall test plan for EasyIT. It includes explanation of different test methods, the objective of testing, and a description of our selected test methods. The test plan documents how we intend to verify that the implementation is according to the requirements specification.

EasyIT is a system that relays heavily on the performance of other systems in its environment. However, during the development of EasyIT, there will be no possibility to carry out a test in context with a real control system. This limitation forms a challenge in finding good and realistic tests. The situation will mainly affect module and stress testing of DI, but will also affect the rest of the testing processes. Detailed plan for testing of the separate modules will follow later in this chapter, in 31.8

31.2 Common methods

There exists two distinct common approaches for testing, Black Box and White Box. The two methods give us different knowledge about the system, and both will be used in the overall test plan for EasyIT.

31.2.1 Black Box testing

In Black Box testing the system is treated without considering its internal structures. All test input is given and output is collected on the outside of the object being tested. This method is often used to test the implementation of functional requirements. We intend to use this method for this purpose.

31.2.2 White Box testing

In White Box testing, knowledge about the internal structures is necessary. The data used for input is specifically selected for testing of the different areas of the system.

31.3 Test stages

This section describes the different test stages defined in the project. These stages cover the whole implementation phase from design of the first classes, to the final acceptance test.

31.3.1 Unit tests

This is tests of the smallest defined units in the system. These tests forms the base of the base of the continuous verification of the systems performance. All unit tests in this project will be executed by the designer, which also generates the test cases based on the design document.Unit tests are performed as White Box tests.

31.3.2 Module test

When units are joined together, they form separate software modules. Module testing is performed to certify that interaction between the separate units is according to specifications. Errors occurring at this stage, should in general not be related to malfunction in single units. Those errors should be discovered in unit testing. In the requirements specification we have identified three separate modules in EasyIT:

- Data Collector, Database Connector and Database
- APP-API
- WEB - Web portal, and WEB-API

We will perform Black Box testing in the module test. The test phase for each module will not occur at the same time, because of dependencies in function and implementation. Execution plan for the module test will be described later in this chapter.

31.3.3 Integration test

At this stage, the different software modules are joined together. Testing will be executed on selected parts of the system to discover possible errors in interaction between modules. Malfunction in previously tested modules should have been discovered in earlier module testing. In these tests, the system will be treated as a Black Box. The functions tested, are according to those implemented in the current increment. In our test plan we will indirectly execute the integration test stage while performing module testing, since we will use previously tested modules as a test environment during the module testing.

31.3.4 System test

As in general integration tests, we will test the system in a simulated environment, to certify that its interface to other subsystems and users is according to requirements specification.

We intend to use Black Box testing at this final integration test. EasyIT has three major interfaces to the outside environment.

- the control sub-system through OPC.
- the human user through the web interface
- the application interface

Testing of the system will be executed simultaneously from, and on, all three interfaces. OPC test data that will be produced by an OPC simulation server. Test data will be described later in this document.

31.3.5 Acceptance test

The acceptance test serve the purpose of a final system test. The former tests should have discovered all technical errors in the system before this stage. This test will be performed by the customer alone, based on the general system documentation. On customers request, more in-depth information may be provided. All additional information will be registered as a documentation or system flaw. This final test will conclude whether the product satisfies its initial requirements or not.

31.4 Test types

This section describes the two test types we will focus on in the project, namely functional testing and testing of non-functional requirements. We could have extended the number of types to include independent tests for process testing, interface testing and integrity testing, but we will include these types while testing the non-functional requirements. These requirements will for the most part be tested in module testing of WEB and at the system test stage.

31.4.1 Functional testing

Testing of functional requirements, as specified in the requirements specification. This test type is included at all testing stages.

31.4.2 Testing of non-functional requirements

Performed both in module and system test stage. For most parts of the system this test is equivalent to interface, performance and stress testing.

31.5 Test approaches

There are numerous ways of carrying out a test stage. Our criteria for choosing approach depends on the nature of the system we are developing. This is also discussed in chapter 8 in the Project Directive.

31.5.1 Requirements based testing

We have selected to follow a requirements based testing approach, an approach we find applicable. Which requirements to be tested is presented later in this chapter. In general we aim at testing as many as possible of the requirements defined in the requirements specification. Requirements tested are both functional and non functional requirements.

31.6 Test data resource

OPC test data will be provided by the DSxPOpcSimulator, an OPC Server for simulation and testing. The server generates random, ramped, and stepped values in the OPC protocol, and is fully configurable by the user.

31.7 Error handling

This section describes classification of errors and incidents in the project. This includes description of error handling procedures.

31.7.1 Error and incident classification

The definitions in this section is taken from *''Test Management Plan Guidelines''*, presented in lecture Oct-01-04.

Critical: The defect causes system crash, loss or corruption of data. The defect also stops the testing completely, both for the test being run and for other tests. Correcting this error or implementing this change request is crucial for the system to work properly.

High: The defect stops the current test, and might also have an impact on testing concerning interfacing units.

Medium: The defect affects this test only and has limited impact for the test progress. Correcting this error may add value to the business in terms of time and/or cost.

Low: The defect does not stop the current test and has no impact on the test progress.

31.7.2 Error handling procedures

All errors detected during the tests will be reported using the error reporting form (see 32.2). However, minor errors (class Low) that are corrected during the test stage do not need to be reported. This rule applies mainly to unit testing. Errors in class medium and above initiates a re-execution of the test, after the error has been corrected.

31.8 Test plan

This section describes the general test process, that is the different activities and the general sequencing of activities to take place as part of the test effort for each individual test stage. The tests described will be performed at different stages in the implementation phase. Responsible for all tests is Test Manager, but tests may and will sometimes be executed by other roles in the group.

31.8.1 Unit Test

The smallest and first test to take place is the unit test. We have identified the following units that will be included in the unit test:

Units in DI

- APP-API
- WEB-API
- Data Collector
- Database Connector
- Database

Units in WEB

- Server Connector
- Tag Browser
- Config Tool
- APP Monitor

Testing of these units will be by executed by developer during the implementation phase. Requirements tested are those identified in the design document. We will not document these tests, due to the nature of the system we are developing. As previously discussed the final product will take the form of a prototype, and we have chosen at this stage to emphasize implementation rather than documentation.

31.8.2 Module and integration test

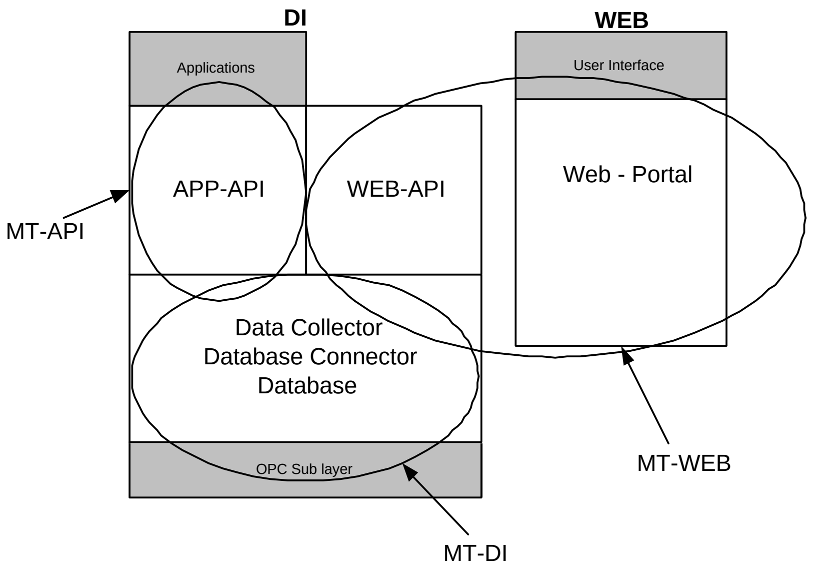


Figure 31.1: Module and Integration Test

The module tests will be executed incrementally. By doing so, we will minimize the amount of time used to prepare each test. Instead of designing numerous specific test scripts, we intend to use a previously tested module to provide data and environment for the tests. This approach creates a gradual transition from module to integration test phase, since interaction between modules will be tested indirectly while each module is tested. The tests will be executed by the developer group for each module in company with the test manager. Figure 31.1 illustrates the three module tests and the interfaces they are affected by. The three circles representing the tests also illustrates how integration of the three modules will be tested.

1. The first module to be tested is the module containing the three units; Data Collector, Database Connector and Database module. This module forms the base of the DI. This module will be tested with the test cases MT-DI-1 - MT-DI-12 (see I.1), which covers all functional requirements for DI.
2. (a) The second module test will test the APP-API. The test can be classified as both a module and an integration test, since it aims to test APP-API as a module, while also testing the

integration with the previously tested sub layers of the DI. This explains why the entity APP-API is treated independently as a unit in the former test stage and now as an independent module.

(b) WEB: The third module and integration test is testing the WEB module and the interface between WEB and DI (WEB-API).

In addition to the general module tests, a stress test (Figure I.25) on DI and the database is also planned to be executed. This test will be performed according to the quality criteria proposed in chapter 8.

31.8.3 System test

The system test will be executed by the end of the implementation period. In this phase the entire system will be tested with the Black Box approach. The test will be executed in a simulated environment, where the system receives input from all of it three interfaces.

Requirements tested from test application

- Set handling
- Deadband handling
- Read/Write data
- Historical data
- Configuration

Requirements tested from web user

- General functional requirements
- Data presentation
- Configuration
- WEB-API

Requirements tested from the OPC Sub layer

- Data Read/Write
- Scheduling
- Handling Tags

Several of the requirement tests for the different modules overlap. Testing of requirements for the APP-API will also indirectly test most of the requirements for DI. The system tests will be executed as followed:

ST-1 Configure DI to read and store tags.

ST-2 Creation of a test application. We will by doing this test, secure the attainment of the overall quality goals proposed in the project directive, chapter 8. The APP should use functions described in API requirements.

ST-3 Test execution of the application. Testing both APP-API and DI requirements.

ST-4 Create web page for the test application.

ST-5 Test Web portal with the predefined MT-WEB tests, and test application running.

31.8.4 Acceptance Test

The final acceptance test will be a modified version of the system test, with a time limit for execution set to two hours. Instead of having to develop code for the test application (Figure: I.27), the customer will be given a code example ready to use. Our conceptual end-user is familiar with the tools and procedures for writing applications for EasyIT. We do not expect the tester, our customer, to be that comfortable with these features of EasyIT in the few minutes reserved for this test. This is why we alter ST-2.

Table 31.8.4 summarize the different test stages used in the project.

	Unit Test	Module and Integration Test	System Test	Acceptance Test
Planned by	TM	TM	TM	TM
Executed by	Designer	Project/TM	Project/TM	Customer
Checklists created by	Designer	TM	TM	TM
Test cases created by	Designer	TM	TM	TM
Test data provided by	Project	Project	Project	Project
Acceptance criteria	Checklist	No High Errors	No High Errors	Customer approval
Schedule	Week 43/44	Week 45	Week 46	Week 46

Table 31.1: Test Scope

31.9 Requirements to be tested

Following requirements will be tested:

- Table 19.1: DI requirement list
- Table 19.13: API requirement list
- Table 19.29: Functional requirements for WEB

31.10 Requirements not to be tested

WEB-1 *All data series logged from OPC servers and data series written by applications shall be available through the web portal.* The WEB portal does not explicit distinct these two types of data. The requirement will be indirectly tested while testing other functions which utilize data from database.

WEB-3 *When viewing data from applications operating on real time data, the user shall be presented with the newest data available at the time of his request.* The requirement has low priority, and will be implemented before the last increment. If implemented, then a unit test will be performed by designer.

DB-4 *The database must support the Structured Query Language (SQL).* Since Microsoft SQL Server is chosen as a database server, we do not see the need for testing this requirement.

DB-5 *The database must be able to run physically separated from the rest of our application.* Since the project has only one PC at disposal we do not have the ability to test this requirement.

Chapter 32
Templates for Testing

32.1 Introduction

All tests executed will be documented by the following templates.

- Test description table, written by TM - Including; detailed step by step description written independent for each test.
- Error reporting form

32.2 Templates

Test ID	*Unique Test ID*		
Title			
Test Stage	*Module/System/Acceptance test*		
Responsible			
Executed by			
Date			
Pre Conditions			
Test Spec.	**Task**	**Expected response**	**Result**
Passed/Failed			
Comments			
Ref. to error log			

Figure 32.1: Test description table

Error Report ID	
Classification	*L / M / H / C*
Reported By	
Date	
Description	
Responsible for correction	
Correction description	

Figure 32.2: Error Report Table

Chapter 33 Summary

All the tests (module tests, system test and the acceptance test) were executed with a satisfactory result. We revealed several issues to be improved, but we did not experience any incidents which could be classified as errors in our system. One reason for this is maybe the way the implementation and unit testing process was done. The top down approach with defined requirements for each phase made us put an effort in getting rid of many errors in the early stages. Important matters, such as integration of modules and connection to interfaces, were prototyped before the actual implementation phase began. In that way we had stable versions of the system when the final requirements were implemented and the official module test and system test were executed.

One test case, STRESS-1, could not be executed due to limitations in our test environment. To execute this test we would have needed more than the one test machine provided by the course and other test servers.

Even though the tests did not reveal any gap between our system and the requirements specification, there are many issues to be solved before it could be classified as stable and applicable for an end user. Our ideas and plans for further development of EasyIT are found in the Implementation Document. A summary of further development and cost estimation is also found in the evaluation document. The requirements specification and hence the tests were defined for a prototype. That has allowed us to overlook important matters for ready-to-ship software, for example security matters such as encryption of passwords, and validation of user input. While these matters are indeed important for software ready to ship, these matters can be overlooked for prototyp software such as EasyIT.

We are happy to conclude that we have documented the fulfillment of all functional requirements for DI, APP-API and WEB.

Appendix I
Test Report

This chapter describes the results from executing the test stages described in the overall test plan (see chapter 31). The tests in this section was made during the Design / Test Design phase. The environment for all tests have been an ordinary computer - 1 GHz - 512 MB Ram. Additional description of data used, and results from the tests are provided in the Test Log, Appendix B J.

I.1 Unit tests

As stated in the overall test plan, unit tests will not be documented. However the identified units have been continuously tested during the implementation phase. The DSxPOpcSimulator has been used to provide test data during implementation. The same test server will be used for the documented module and system tests. The developers report that unit testing has been executed, and all units perform as expected before the module tests takes place.

I.2 Module Test Increment 1 - DI

The first documented test to be executed is the DI module test. This section displays the test cases from MT-DI-1 to MT-DI-5.

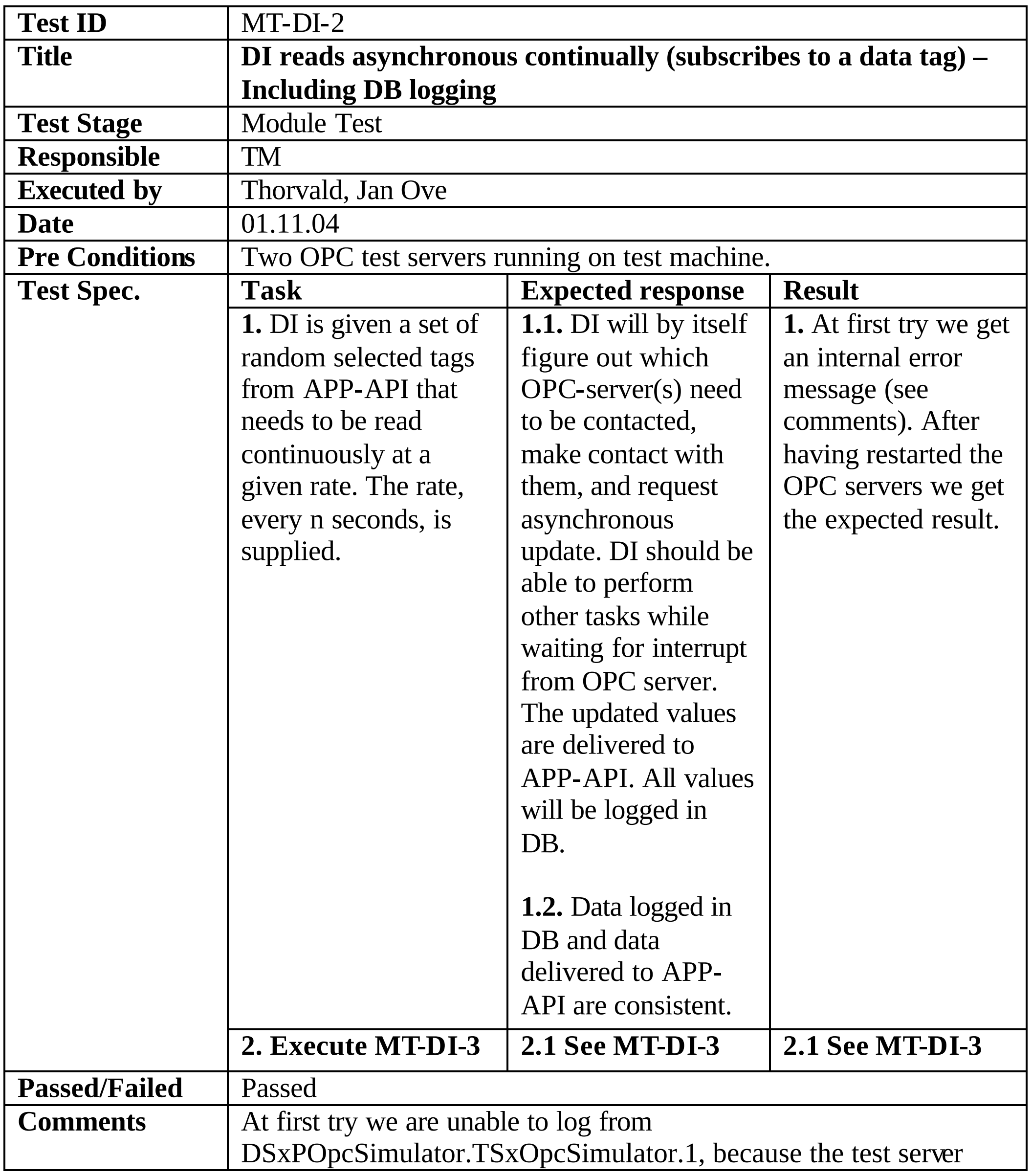
I.2.1 MT-DI-1

Test ID	MT-DI-1		
Title	**DI reads synchronous from OPC-server**		
Test Stage	Module Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	01.11.04		
Pre Conditions	Two OPC test servers running on test machine.		
Test Spec.	**Task**	**Expected response**	**Result**
	1. DI is given two random selected tags from the APP-API that needs an update.	**1.** DI will by itself figure out which OPC-server(s) need to be contacted, make contact with them - and request synchronous update. In this state all other connections will be blocked. The updated values are delivered to APP-API **2.** Data delivered to APP-API are consistent.	**1.** Both OPC servers are contacted, and synchronous tag update is done. Tag values, and OPC-server addresses are shown in log table.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.1: MT-DI-1

I.2.2 MT-DI-2 and MT-DI-3

MT-DI-2 and MT-DI-3 are executed as one test since MT-DI-3 is depending on having a subscription to stop.



Test ID	MT-DI-2		
Title	**DI reads asynchronous continually (subscribes to a data tag) – Including DB logging**		
Test Stage	Module Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	01.11.04		
Pre Conditions	Two OPC test servers running on test machine.		
Test Spec.	**Task**	**Expected response**	**Result**
	1. DI is given a set of random selected tags from APP-API that needs to be read continuously at a given rate. The rate, every n seconds, is supplied.	**1.1.** DI will by itself figure out which OPC-server(s) need to be contacted, make contact with them, and request asynchronous update. DI should be able to perform other tasks while waiting for interrupt from OPC server. The updated values are delivered to APP-API. All values will be logged in DB. **1.2.** Data logged in DB and data delivered to APP-API are consistent.	**1.** At first try we get an internal error message (see comments). After having restarted the OPC servers we get the expected result.
	2. Execute MT-DI-3	**2.1 See MT-DI-3**	**2.1 See MT-DI-3**
Passed/Failed	Passed		
Comments	At first try we are unable to log from DSxPOpcSimulator.TSxOpcSimulator.1, because the test server		

Figure I.2: MT-DI-2

Test ID	MT-DI-3		
Title	**DI stops a continually asynchronous read.(A subscribed tag)**		
Test Stage	Module Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	01.11.04		
Pre Conditions	1. Two OPC test servers running on test machine. 2. MT-DI-2 is executed and subscription not stopped		
Test Spec.	**Task**	**Expected response**	**Result**
	1. DI is given a command from APP-API to stop subscribing a group of tags.	**1.** DI will by itself figure out which OPC-server(s) need to be contacted, make contact with them, and cancel further subscription. DI should be able to perform other tasks while waiting for acknowledgement from OPC server. **2.** Data logged in DB and data delivered to APP-API are consistent.	**1.** Subscription was stopped according to the given signal.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.3: MT-DI-3

I.2.3 MT-DI-4

Test ID	MT-DI-4		
Title	**DI starts applications**		
Test Stage	Module Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	01.11.04		
Pre Conditions	Schedule table is not empty.		
Test Spec.	**Task**	**Expected response**	**Result**
	1. DI starts up and checks database to figure out which applications it needs to start. Starts all needed applications.	**1.** Applications start up.	**1.** Application start up according to specifications from database.
Passed/Failed	Passed		
Comments	The following was written to EasyIt-log: AL: application C:\work\EasyIt\EasyIt\EasyItStarter\bin\Debug\apps\HackerTest.exe started.		
Ref. to error log	No errors occurred		

Figure I.4: MT-DI-4

I.2.4 MT-DI-5

Test ID	MT-DI-5		
Title	**Creation and removal of tag groups.**		
Test Stage	Module Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	01.11.04		
Pre Conditions	Two OPC servers are running.		
Test Spec.	**Task**	**Expected response**	**Result**
	1. DI is told directly by APP-API to start subscription on a tag from one of the two servers. **1.2.** Step one is repeated 8 times.	*For each new requested tag:* **1.1.** DI checks the tags and finds out which tags belong on the same servers. **1.2.** DI creates sub-groups with all the tags with the same rate, on the same servers in 1). **1.3** DI subscribes to the needed subgroups on the different OPC-servers.	**1.1** Subscription from to servers is as expected. New tags are added to their respective sub-groups.
	2.1 The subscriptions of tags created in 1), is requested to be deleted from APP-API.	**2.1.** DI checks the tag and finds out which sub-groups it is in. **2.2.** DI deletes the tags from the groups. **2.3.** DI stops all subscriptions for empty sub-groups. **2.4** DI disconnects from all servers without any subscriptions.	**2.1** Tags are deleted from sub-groups **2.2** Subscription of the empty sub-groups is stopped. **2.4** DI has disconnected from the two test servers.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.5: MT-DI-5

I.3 Module Test Increment 2 - API

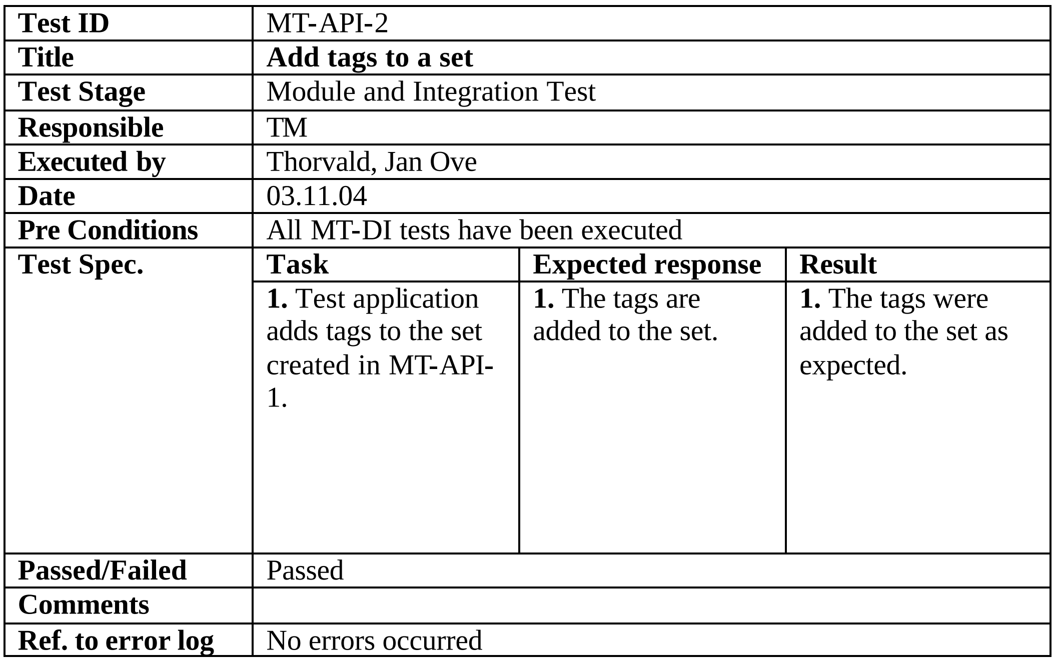
The second test increment is testing of the APP-API. This section describes the test cases from MT-API-1 to MT-API-13.

I.3.1 MT-API-1

Test ID	MT-API-1		
Title	**Define set of tags**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	All MT-DI tests have been executed		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Test application creates a set-object.	**1.** A new set was created.	**1.** New set was created as expected.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.6: MT-API-1

I.3.2 MT-API-2



Test ID	MT-API-2		
Title	**Add tags to a set**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	All MT-DI tests have been executed		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Test application adds tags to the set created in MT-API-1.	**1.** The tags are added to the set.	**1.** The tags were added to the set as expected.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.7: MT-API-2

I.3.3 MT-API-3

Test ID	MT-API-3		
Title	**Remove tags from a set**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	All MT-DI tests have been executed		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Test application remove tags to the set created in MT-API-1.	**1.** The tags are removed from the set.	**1.** The tags were removed from the set.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.8: MT-API-3

I.3.4 MT-API-4

Test ID	MT-API-4		
Title	**Splice sets**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. OPC test server is running 3. Functions used in sub layers of DI are implemented		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Test application calls the splice function in the APP-API with to sets as arguments.	**1.** The API returns a new set with the following properties: • The new set is an union of the two sets • If tags intersect, only the most up to date are members of the new set.	**1.** The result is as expected. The new set is a union of the two sets.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.9: MT-API-4

I.3.5 MT-API-5

Test ID	MT-API-5		
Title	**Delete a set**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	All MT-DI tests have been executed.		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Test application deletes the set object created in MT-API-1.	**1.** The set is deleted.	**1.** The set was deleted.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.10: MT-API-5

I.3.6 MT-API-6

Test ID	MT-API-6		
Title	**Setting the deadband variable**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. OPC test server is running 3. Functions used in sub layers of DI are implemented		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Test application set the deadband variable on a tag. **2.** Test application adds tag to set. **3.** Test application subscribes to set.	**1.** APP-API asks DI to subscribe to the set. **2.** New data is only received from OPC for this tag if the new data is changed more than the deadband indicates.	**1.** New data were only received if the value varied more than the deadband variable.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.11: MT-API-6

I.3.7 MT-API-7

Test ID	MT-API-7		
Title	**Subscribe to a set of tags**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. OPC test server is running 3. Functions used in sub layers of DI are implemented		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Test application calls the API. The call provides the set and the rate at which the set is to be subscribed at.	**1.** The APP-API subscribes to the set in DI and returns a set ID immediately to the application. **2.** In accordance to the rate, the APP-API calls the application back regularly. The callback contains a new set with the latest values of the tags in the requested set.	**1.** Subscription was established, as expected.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.12: MT-API-7

I.3.8 MT-API-8

Test ID	MT-API-8		
Title	**Terminate subscription of tags**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. OPC test server is running 3. Functions used in sub layers of DI are implemented		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Test application calls the APP-API. The application provides the ID of the set to be terminated.	**1.** APP-API tells DI to terminate the set requested by test application.	**1.** Termination of subscription executed as expected.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.13: MT-API-8

I.3.9 MT-API-9

Test ID	MT-API-9		
Title	**Synchronous poll of a set of tags**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. OPC test server is running 3. Functions used in sub layers of DI are implemented		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Test application calls the API. The call provides the ID of the set.	**1.** APP-API gathers data and returns the set. The control of the caller is not released until the set is returned. **2.** The newest data set is returned to the application.	**1.** The synchronous poll executed as expected.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.14: MT-API-9

I.3.10 MT-API-10

Test ID	MT-API-10		
Title	**Write data back to the DB**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. OPC test server is running 3. Functions used in sub layers of DI are implemented		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Test application calls the API with a set of tags as argument.	**1.** The set is stored in the database.	**1.** The set is stored in the database, according to the expected response.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.15: MT-API-10

I.3.11 MT-API-11

Test ID	MT-API-11		
Title	**Read data from DB**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. OPC test server is running 3. Functions used in sub layers of DI are implemented		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Test application calls the APP-API with a tag, a from-date and a to-date.	**1.** The APP-API gets all logged tag values of the given tag in the given interval and returns this as a series of tags to the application.	**1.** APP-API gets all tags in the specified interval.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.16: MT-API-11

I.3.12 MT-API-12

Test ID	MT-API-12		
Title	**Write configuration (And read configuration)**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. OPC test server is running 3. Functions used in sub layers of DI are implemented		
Test Spec.	**Task**	**Expected response**	**Result**
	1.1 Test application calls the API, requesting a configuration write. The application identifies itself to the API, so that any old configuration will be updated.	**1.1** If the configuration identification is unknown, the API creates a new configuration record for the specified configuration. If the configuration id is know, the configurations are stored directly.	**1.1** The test application writes "testvalue" to the config field "TestField"; The database shows that config field TestField for application with the id of the test application is stored with value "testvalue".
	2.1 Execute MT-API-15	**2.1 See MT-API-15**	**2.1 See MT-API-15**
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.17: MT-API-12

I.3.13 MT-API-13

Test ID	MT-API-13		
Title	**Read configuration**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	03.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. OPC test server is running 3. Functions used in sub layers of DI are implemented		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Test application calls the API, requesting configurations. The application has to identify itself to the API, so that the right configuration can be returned.	**1.** The API returns the configuration to the application.	**1.** A test application reads the config. field "TestField" from MT-API-14. It gets the correct value, "testvalue", returned.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.18: MT-API-13

I.7 Acceptance test

The acceptance test was executed successfully on the 11.11.04. The test cases used are identical to the ones covering the system tests (except ST-2). Since no errors occurred, the test cases will not be repeated here. The exception is, that instead of having to develop code for the test application (Figure: I.27), the customer was given a code example ready to use. The code example was the test application written by tester in ST-2. The customer was given a code walkthrough of the test application, as a presentation of the feature.
Since our customer is located in Oslo, the test was done via Microsoft's Remote Desktop. The test executors were able to observe the tester on the shared computer desktop. For communication an ordinary telephone connection was used.

I.4 Module Test Increment 3 - WEB

The third test increment is testing of the WEB. This section describes the test cases from MT-WEB-1 to MT-WEB-6.

I.4.1 MT-WEB-1

Test ID	MT-WEB-1		
Title	**Authorize user**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Hans Olaf		
Date	04.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. DI is running		
Test Spec.	**Task**	**Expected response**	**Result**
	1. User enters the URL for EasyIT in the web browser.	**1.1.** The web portal redirects the user to a login page, prompting the user for his username and password	**1.1** Ok. Webpage is displayed in browser
	2.1 The tester supplies a regular username and password. **2.2.** User press "Log in" in the web interface.	**2.1.** The web portal accesses the user database through the web API.	**2.1** User name is first wrong spelled. (see comments) **2.2** Correct username is given.
	3.1 Repeat from **2.1.** This time supplying an administrator username.	**3.1.** The web portal receives a notice that the user has been logged in, and additio nal information about the user. **3.2.** The web portal displays a start page giving an overview of the available applications. **3.3**. If the user is an administrator. The web portal displays a control panel allowing the user to choose between the available administrator tasks.	**3.1.** Start page is displayed, both for user and administrator logon.
Passed/Failed	Passed		
Comments	In 2.1, the username was first wrong spelled. The web portal gave an error message that it did not recognize the user. The test continued without restarting.		
Ref. to error log	No errors occurred		

Figure I.19: MT-WEB-1

I.4.2 MT-WEB-2

Test ID	MT-WEB-2		
Title	**View results outputted by applications**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Hans Olaf		
Date	04.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. User is logged in		
Test Spec.	**Task**	**Expected response**	**Result**
	1.1 The tester selects an application to investigate further	**1.1** The web portal displays a list of the available web pages for the selected application.	**1.1** The correct list is presented
	2.1 The tester selects which results to view.	**2.1** The web portal displays the data in the way specified by application the programmers that have customized the web page	**2.1.** The correct results are presented.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.20: MT-WEB-2

I.4.3 MT-WEB-3

Test ID	MT-WEB-3		
Title	**Browse data series logged by DI**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Hans Olaf		
Date	04.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. User is logged in		
Test Spec.	**Task**	**Expected response**	**Result**
	1.1 The tester selects which data series to be browse, and sets granularity and time span.	**1.1.** The web portal gets the selected data series from the DI	**1.1.** The selected tags appear in the list box.
	2.1 The tester selects how the data series should be displayed (test for both table and time plot)	**2.1.** The web portal displays the data in the way specified by the user	**2.1** Data series is displayed in both table and time plot.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.21: MT-WEB-3

I.4.4 MT-WEB-4

Test ID	MT-WEB-4		
Title	**Configure applications**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Hans Olaf		
Date	04.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. Tester is logged in as Administrator 3. The tester has selected “Configure applications” in the menu		
Test Spec.	**Task**	**Expected response**	**Result**
	1.1 The tester selects an application to configure.	**1.1**. The web portal displays the existing configuration of the chosen application	**1.1** Configur ation is displayed
	2.1. The tester: - adds a web page -deletes a web page -changes application name	**2.1** The web portal allows changes in configurations to be made.	**2.1** Ok.
	3.1 The tester submits the changes made in 2.1.	**3.1.** The web portal reads the user input, and makes the appropriate changes to the configurations **3.2.** The web portal receives a notice when the changes are saved **3.3.** The web portal displays a confirmation to the user	**3.1** A confirma tion is displayed . Configur ation is saved.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.22: MT-WEB-4

I.4.5 MT-WEB-5

Test ID	MT-WEB-5		
Title	**Manage user accounts**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Hans Olaf		
Date	04.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. Tester is logged in as Administrator 3. Tester has selected “Manage users” in the menu		
Test Spec.	**Task**	**Expected response**	**Result**
	1. The tester selects a user to configure	**1.1.** The web portal displays the existing configuration for the chosen user **1.2**. The administrator is allowed to edit properties of the selected user	**1.1.** Existing configurations are displayed.
	2.1. The tester alters configuration data for the user and which applications the user is permitted to access, and submits the changes.	**2.1.** The web portal reads the user input, and makes the appropriate changes to the configurations.	**2.1. (**See comments) **2.2.** Changes are submitted and stored.
Passed/Failed	Passed		
Comments	By accident the tab key was pressed when entering configurations. The server responded with an error message. Since user validation is not required for our system, this is not classified as an error. The test continued with Task 2.1, the next result was Result 2.2.		
Ref. to error log	No errors occurred		

Figure I.23: MT-WEB-5

I.4.6 MT-WEB-6

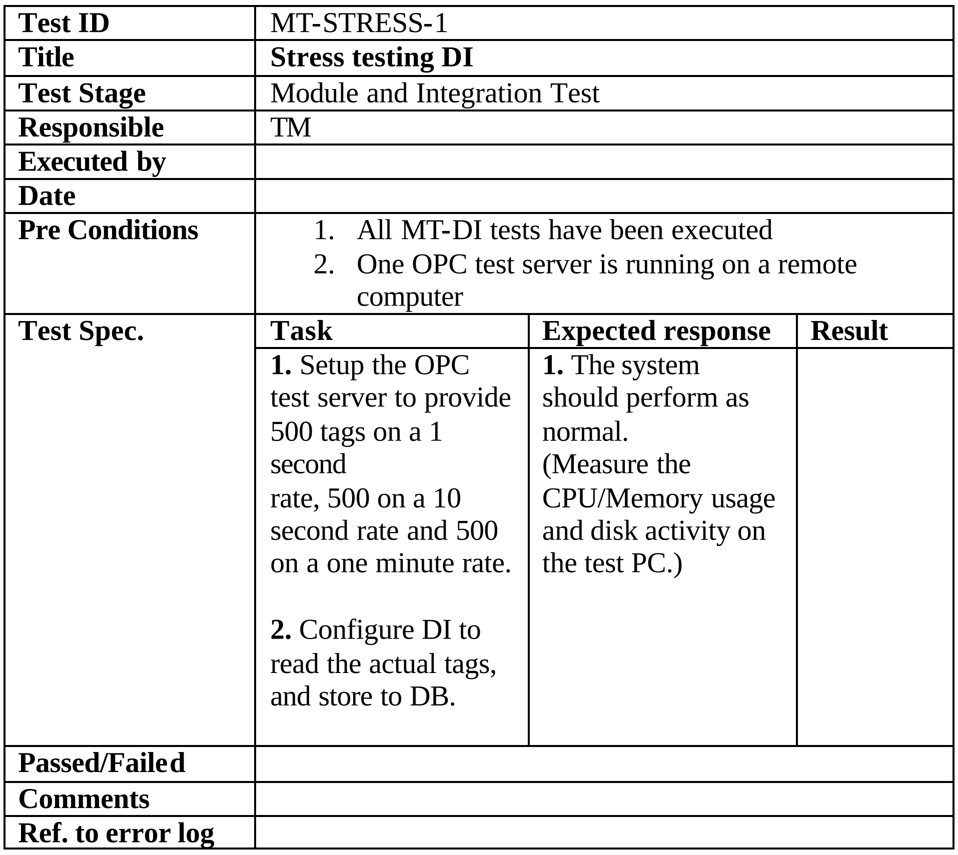
Test ID	MT-WEB-6		
Title	**Monitor application status**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by	Thorvald, Hans Olaf		
Date	04.11.04		
Pre Conditions	1. All MT-DI tests have been executed 2. Tester is logged in as Administrator		
Test Spec.	**Task**	**Expected response**	**Result**
	1. The tester selects to view the application status	**1**. The web portal requests status of all the applications from the DI. **2**. The administrator is presented with a simple list of status for each of the applications **3**. The administrator can start and stop registered applications	**1.** The correct application status is displayed on the web portal.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.24: MT-WEB-6

I.5 Stress testing DI

Because limitations in the test environment, we were not able to perform this test. To execute this test we would have needed more than the one test machine provided by the course and other test servers.

I.5.1 MT-STRESS-1



Test ID	MT-STRESS-1		
Title	**Stress testing DI**		
Test Stage	Module and Integration Test		
Responsible	TM		
Executed by			
Date			
Pre Conditions	1. All MT-DI tests have been executed 2. One OPC test server is running on a remote computer		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Setup the OPC test server to provide 500 tags on a 1 second rate, 500 on a 10 second rate and 500 on a one minute rate. **2.** Configure DI to read the actual tags, and store to DB.	**1.** The system should perform as normal. (Measure the CPU/Memory usage and disk activity on the test PC.)	
Passed/Failed			
Comments			
Ref. to error log			

Figure I.25: MT-STRESS-1

I.6 System Test

The final documented test is the system test. The system test was executed according to the overall test plan. This section describes the test cases from ST-1 to ST-5.

I.6.1 ST-1

Test ID	ST-1		
Title	**Configuration of DI**		
Test Stage	System Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	08.11.04		
Pre Conditions	1. All module tests have been executed. 2. OPC Test server is running 3. The system is installed and running on a machine (s) fulfilling the soft and hardware requirements specified in the requirements specification.		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Configure the DI to connect to OPC test server, and by this be able to provide tags to application developer.	**1.** Configuration should be easy, and not exceed 15 minutes.	**1.** Configuration was finished in 3 minutes.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.26: ST-1

I.6.2 ST-2

Test ID	ST-2		
Title	**Create application**		
Test Stage	System Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	08.11.04		
Pre Conditions	1. All module tests have been executed. 2. OPC Test server is running 3. The system is installed and running on a machine(s) fulfilling the soft and hardware requirements specified in the requirements specification.		
Test Spec.	**Task**	**Expected response**	**Result**
	1. The test executor will create a test application which utilizes the implemented functions in APP-API, according to specified requirements.	**1.** The time used for development of the test application should not exceed 30 min.	**1.** Test application was made in 10 minutes.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.27: ST-2

I.6.3 ST-3

Test ID	ST-3		
Title	**Execute application**		
Test Stage	System Test		
Responsible	TM		
Executed by	Thorvald, Jan Ove		
Date	08.11.04		
Pre Conditions	1. All module tests have been executed. 2. OPC Test server is running 3. The system is installed and running on a machine(s) fulfilling the soft and hardware requirements specified in the requirements specification.		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Configure DI to start scheduling of the test application created in ST-2. **2.** Let the test application run while performing ST-4 and ST-5.	**1.** The result from execution of the test application should be according to requirements and design specifications, and not cause the DI to restart.	**1.** The test application is started up, without causing a restart on DI.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.28: ST-3

I.6.4 ST-4

Test ID	ST-4		
Title	**Create web page for the test application created in ST-2**		
Test Stage	System Test		
Responsible	TM		
Executed by	Thorvald, Hans Olaf		
Date	08.11.04		
Pre Conditions	1. All module tests have been executed. 2. OPC Test server is running 3. The system is installed and running on a machine(s) fulfilling the soft and hardware requirements specified in the requirements specification. 4. Test application is running.		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Create a web page for the test application created in ST-2.	**1.** The time used for creation of web page, should not exceed 30 minutes, and be easily included in the web portal without having to restart.	**1.** The webpage is created in 3 minutes, and is displayed through the web portal.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.29: ST-4

I.6.5 ST-5

Test ID	ST-5		
Title	**Re-execute web portal module tests**		
Test Stage	System Test		
Responsible	TM		
Executed by	Thorvald, Hans Olaf		
Date	08.11.04		
Pre Conditions	1. All module tests have been executed. 2. OPC Test server is running 3. The system is installed and running on a machine(s) fulfilling the soft and hardware requirements specified in the requirements specification. 4. Test application is running.		
Test Spec.	**Task**	**Expected response**	**Result**
	1. Re-execute the web portal module tests (MT-WEB-1 to MT-WEB-6) with the current test application running	**1.** Response should be according to those defined in MT-WEB-1 to MT-WEB-6.	**1.** All tests give expected results. No changes since module test.
Passed/Failed	Passed		
Comments			
Ref. to error log	No errors occurred		

Figure I.30: ST-5

APPENDIX J
TEST LOG

This chapter describes the different data used in all module and system tests, and their result.

J.1 MT-DI

This section includes the test data, and results from the module tests done on DI.

J.1.1 MT-DI-1

The test tags selected:

- 1: DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Real.Real 01"
- 2: ICONICS.SimulatorOPCDA.2","","Numeric.Step"

Test values delivered to APP-API:

- S1: 0.00000000023283064365386963
- S2: 1.0

J.1.2 MT-DI-2 and MT-DI-3

MT-DI-2 and MT-DI-3 are executed as one test since MT-DI-3 is depending on having a subscription to stop.

The test tags selected:

- 1: DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Real.Real 01"
- 2: ICONICS.SimulatorOPCDA.2","","Numeric.Step"

Set update rate: 2 sec

The result from logging is shown by the out print from the database:

J.1.3 MT-DI-4

This test required no input data, nor produced one.

Tag:	Timestamp:	Value
Numeric.Step	01.11.2004 10:28:55	0.5
Numeric.Step	01.11.2004 10:28:55	0.5
Simulation Items.Real.Real 01	01.11.2004 10:28:55	214.161479697796
Numeric.Step	01.11.2004 10:28:57	1
Simulation Items.Real.Real 01	01.11.2004 10:28:59	216.325453507481
Simulation Items.Real.Real 01	01.11.2004 10:29:01	216.810264586471
Numeric.Step	01.11.2004 10:29:01	0

Table J.1: Tags logged in database

J.1.4 MT-DI-5

The test application started subscription on the provided 2 second rate, on each of these collection of tags.

```
DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Real.Real_01"
ICONICS.SimulatorOPCDA.2","","Numeric.Step");

DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Real.Real_02"
ICONICS.SimulatorOPCDA.2","","Numeric.Step"

DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Real.Real_03"
ICONICS.SimulatorOPCDA.2","","Numeric.Step"

DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Real.Real_04"
ICONICS.SimulatorOPCDA.2","","Numeric.Step"

DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Integer.Int_01"
ICONICS.SimulatorOPCDA.2","","Numeric.Step"

DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Integer.Int_02"
ICONICS.SimulatorOPCDA.2","","Numeric.Step"

DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Integer.Int_03"
ICONICS.SimulatorOPCDA.2","","Numeric.Step"

DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Integer.Int_04"
ICONICS.SimulatorOPCDA.2","","Numeric.Step"
```

The following optimal sets were created:

Set a: DSxPOpcSimulator.TSxOpcSimulator.1, rate 2

- Simulation Items.Real.Real 01
- Simulation Items.Real.Real 02
- Simulation Items.Real.Real 03
- Simulation Items.Real.Real 04
- Simulation Items.Integer.Int 01
- Simulation Items.Integer.Int 02
- Simulation Items.Integer.Int 03
- Simulation Items.Integer.Int 04

Set b: CONICS.SimulatorOPCDA.2, rate 2

- Numeric.Step
- Simulation Items.Real.Real 02
- Simulation Items.Real.Real 03
- Simulation Items.Real.Real 04
- Simulation Items.Integer.Int 01
- Simulation Items.Integer.Int 02
- Simulation Items.Integer.Int 03
- Simulation Items.Integer.Int 04

Then the application requested subscription for the following tags with a rate on 3 seconds.

```
DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Real.Real_05"
ICONICS.SimulatorOPCDA.2","","Numeric.Step"
```

The following new correct set were made:

Set c: DSxPOpcSimulator.TSxOpcSimulator.1, rate 3

- Simulation Items.Real.Real 05

Set d: CONICS.SimulatorOPCDA.2, rate 3

- Numeric.Step

J.2 MT-API

This section includes the test data, and results from the module tests done on APP-API.

J.2.1 MT-API-1,MT-API-2,MT-API-3 and MT-API-5

All three test used the same test data winch were provided in MT-DI-5. The result from these tests are also identical with the ones produced in MT-DI-5.

J.2.2 MT-API-4

Two set of the following tags were spliced:

```
DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Real.Real_01"
ICONICS.SimulatorOPCDA.2","","Numeric.Step"

Set 1:
Numeric.Step                        03.11.2004 11:26:54         0.5
Simulation Items.Real.Real_02       03.11.2004 11:26:55       111.791614697796

Set 2:
Numeric.Step                       03.11.2004 11:26:57        1
Simulation Items.Real.Real_01       03.11.2004 10:27:01       193.331682227481
```

The result set with three tags, after splice:

```
Numeric.Step                        03.11.2004 11:26:57        1
Simulation Items.Real.Real_01        03.11.2004 10:27:01        193.331682227481
Simulation Items.Real.Real_02        03.11.2004 11:26:55        111.791614697796
```

J.2.3 MT-API-6

A set with rate 3 and deadband 1.0 on all tags were requested via an application. At the same time, DSxPOpcSimulator and ICONICS were set to provide a constant simulation value on these tags. These tags were requested:

```
DSxPOpcSimulator.TSxOpcSimulator.1", "", "Simulation Items.Real.Real_05" = 0.5
ICONICS.SimulatorOPCDA.2","","Numeric.Step" = 1.0
```

The result was as expected: No callback was made from OPC, since there was no change in value.

J.2.4 MT-API-7 and MT-API-8

Both tests used the same test data provided in MT-DI-3. The result from these tests are also identical with the ones produced in MT-DI-3.

J.2.5 MT-API-9

This test used the same test data provided in MT-DI-1. The results from the test are also identical with the ones produced in MT-DI-1.

J.2.6 MT-API-10

The following set of tags where requested to be written to database:

```
TestDataTag1 = 1.343
TestDataTag2 = 34.343
TestDataTag3 = 76.333
```

The following rows where added to the database:

```
TestDataTag1        03.11.2004 11:50:57        1.343
TestDataTag2        03.11.2004 11:50:57        34.343
TestDataTag3        03.11.2004 11:50:57        76.333
```

J.2.7 MT-API-11

The tag Simulation Items.Real.Real 01 on OPC-simulator DSxPOpcSimulator.TSxOpcSimulator.1 is requested from 03.11.2004 10:45:00 to 03.11.2004 10:47:00. The following tag values are returned:

```
03.11.2004 10:45:33        120.916258137207
03.11.2004 10:45:37        122.469736196101
03.11.2004 10:45:39        123.548466004198
03.11.2004 10:46:10        135.612150835572
03.11.2004 10:46:12        136.250593777746
```

J.2.8 MT-API-14 and MT-API-15

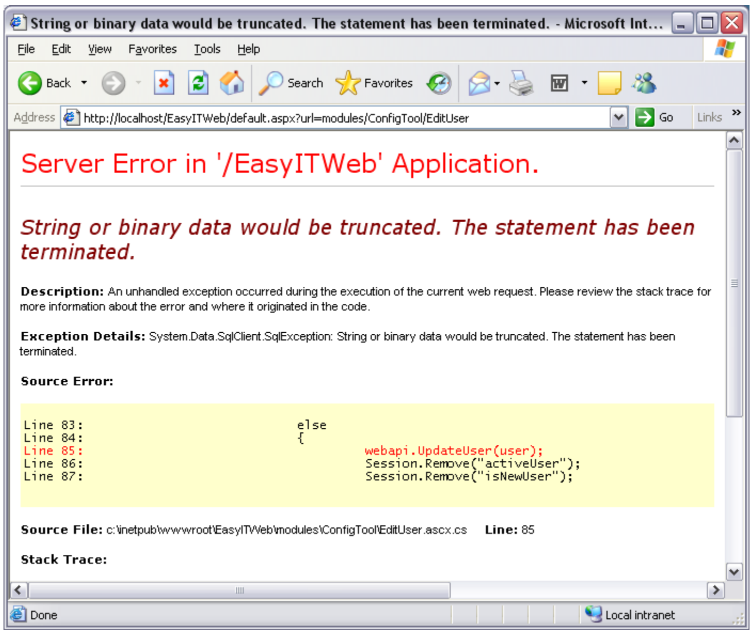
Only specified in the Test Report.

J.3 MT WEB

These tests were based on user input through the web portal. The results from these tests were a confirmation given back to the user in the form of an alternation in the display of the web page. These tests are described further in the test cases found in the Test Report. The only test result documented here is an exception screen shot from MT-WEB-5.

J.4 MT-WEB-5

By accident the tab key was pressed when entering configurations. The server responded with an error message. Since user validation is not required for our system, this is not classified as an error. The test continued with Task 2.1 without restarting. Figure J.1, displays a screen shot documenting the incident.


String or binary data would be truncated. The statement has been terminated. - Microsoft Int...
File Edit View Favorites Tools Help
Back
Search
Favorites
Address http://localhost/EasyITWeb/default.aspx?url=modules/ConfigTool/EditUser
Go
Links
Done
Local intranet

Server Error in '/EasyITWeb' Application.

String or binary data would be truncated. The statement has been terminated.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: String or binary data would be truncated. The statement has been terminated.

Source Error:

```
Line 83:                                 else
Line 84:                                 {
Line 85:                                         webapi.UpdateUser(user);
Line 86:                                         Session.Remove("activeUser");
Line 87:                                         Session.Remove("isNewUser");
```

Source File: c:\inetpub\wwwroot\EasyITWeb\modules\ConfigTool\EditUser.ascx.cs **Line:** 85

Stack Trace:

Figure J.1: Screen shot MT-WEB-5

J.5 ST

J.5.1 ST-1

The configuration file created in ST-1:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
        <appSettings>
                <add key="serverPort" value="6666" />
                <add key="applicationDirectory" value="apps" />
                <add key="database" value="fest" />
                <add key="databaseHost" value="doc10.idipc.idi.ntnu.no" />
                <add key="databaseUser" value="easyit" />
                <add key="databasePassword" value="easyit" />
        </appSettings>
</configuration>
```

J.5.2 ST-2

Code for the test application created in ST-2:

```
using System;
using EasyIt.Application;
using EasyIt.DataItems;
namespace TestApp1
{
/// <summary>
 /// Summary description for Class1.
/// </summary>
class Class1
{
 /// <summary>
 /// The main entry point for the application.
 /// </summary>
[STAThread]
 static void Main(string[] args)
 {
AppAPIClientBase clientBase = new AppAPIClientBase(4, 6666, "localhost");
 clientBase.DataArrival +=new
 EasyIt.RemoteInterfaces.DataReceivedEventHandler (clientBase_DataArrival);
CTag tag = new CTag("opcda://localhost/ICONICS.SimulatorOPCDA.2", "", "Numeric.Step");
CSet set = new CSet("", 3);
set.AddTag(tag);
 clientBase.SubscribeSet(set);
                }
private static void clientBase_DataArrival(CSet setm)
                {
Console.WriteLine("got data from EasyIt");
                }
        }
}
```

J.6 Acceptance test

The test cases used for the acceptance test are identical to the ones covering the system tests, except ST-2 (for further description see the overall test plan 31). Since no errors occurred, all test data and results provided in this log for system tests also apply to the acceptance test.

Part VII

Project Evaluation

Chapter 34 Introduction

This part evaluates the project, the course TDT420 Kundestyrt Prosjekt and the other components required to accomplish the project.

34.1 Purpose

The goal of this project evaluation is to critcal examine the project and the way it has been carried out. We will look at the results as well as the organization of the project, and point out positive and negative matters with regards to the accomplishment of the project. Hopefully, all involved parties (the project group especially, but perhaps also the customer and the advisors) can learn from the matters pointed out in this part in later projects.

34.2 Overview

The following views of the the project are evaluated in the succeeding chapters:

Chapter 35 Provides an evaluation of the outcome of the project.

Chapter 36 Provides an evaluation of different forms of cooperation we have experienced during the project.

Chapter 37 Provides an evaluation of how we solved practical obstacles during the project.

Chapter 38 Provides an evaluation of the tools we used for documentation, implementation og versioning control.

Chapter 39 Provides an evaluation on how we carried out the different phases of the project

Chapter 40 Provides an evaluation of the progression and hours spent on the project

Chapter 41 Concludes the project evaluation.

Chapter 35
The project outcome

This chapter deals with evaluations of the outcome. First, we examine how we carried out the aminis-traive parts of the project, then we evaluate the outcome of the course TDT4290 itself.

35.1 The phases of the project mapped to parts of the document

The project has been divided into 8 phases. Easch phase has been accompanied by a phase document. The different phases of the project are:

1. Preliminary planning - as presented in 3.1.1
2. Pre study - as presented in 3.1.2
3. Requirements specification /test plan creation- as presented in 3.1.3
4. Design and test plan construction - as presented in 3.1.4
5. Implementation and testing - as presented in 3.1.5
6. Project evaluation - as presented in 3.1.6
7. Presentation and demonstration - see 3.1.7

The phase documents, divided by parts, are mapped to the phases as follows:

Part I - *Project directive* maps phase 1

Part II - *Pre study* maps phase 2

Part III - *Requirements specification* maps phase 3

Part IV - *Software Design Description* maps phase 4

Part V - *Implementation document* maps phase 5

Part VI - *Test Document* maps the test contents of phase 3, 4 and 5

Part VII - *Project evaluation* maps phase 6

Part VIII - *User guide to EasyIT* maps partially phase 7

Part I is mainly about managing the project, whereas we're approaching the EasyIT system with gradually increasing precision during the phases II, III and IV. We have written Part III and IV to comply with IEEE standards. Part V seeks to cover the main aspects of the implementation, with important code snipplets and description of essential algorithms. Part VI covers the overall test plan and test reports for all official tests of the system. Bear in mind, though, that the EasyIT system is regarded as a prototype, and that exhaustive testing has not been prioritzed on a high level. Part VII is this part, it seeks to evaluate the project. In part VIII a short user guide to EasyIT is provided. This serves as some of the background material for the presentation.

We see several advantages of the mapping of phases to document parts. Doing so makes an even more obvious separation between the phases, and thereby makes natural mile stones for the project. We have one exception for this practice - the test document, which doesn't map to a specific phase. The reason for this it that the work with the testing has been carried out in parallel with the other phases as stated in the first list above.

Overall, we're satisfied with this approach.

35.2 The TDT4290 course

The group agrees on the high relevance of the course - this is perhaps the most relevant course yet with regards to experience what a real-life job situation will be like. We have put a lot of effort in this project, and we have learned a lot. This section provides an evaluaton of how we experienced the following aspects of the course.

35.2.1 Organization of the course

The organization of the course has been adequate. The course started on time, and the course staff succeeded to motivate us. In short, the first impression of the organization was very good. We experienced however something that could be done better:

Every group was promised the exclusive use of one computer. We needed a computer to run an OPC dummy server, an MS SQL server, MS Virtual Source Safe and Visual Studio - in short, we needed a computer. We had to ask for it, then wait, then ask for it again, then wait, then ask for it again before we could finally use it. We can't see any reason for why the computers weren't ready from day one. Also, we suggest that the room in which the computers are placed are marked as unavailable to everyone not involved in the course. Before we unplugged the monitor,keyboard and mouse from the computer, and put a note over the monitor that said: ”KPRO10 only”, we experienced several times that someone had rebooted the computer and used it for something else.

The course compendium handed out in the beginning course appears poorly organized. We suggest a rearranging of the compendium, where the the administrative matters are placed at the beginning, followed by the content of the each phase document, and the different projects placed at the end.

35.2.2 Lectures

The lectures given in the course were supposed to support the work with the project, and they were. We have nothing adversely to point to with regard to the lectures.

35.2.3 Mandatory activity I: Six Thinking Hats

Jens Aarup, the lecturer this day, was outstanding. We learned a lot from this session. We tried to use the Six Thinking Hats method in the sucessive days after the session, but we did not use it later, although it could have been useful. We see Six Thinking Hats as a useful resource, but it presumes considerable effort from each group member. We didn't use enough time to establish the proper setting to make this method work.

The organization of the day could have been better. 7 hours straight with only short breaks in between made us unfocused at the end of the day. Perhaps this could be avoided with a split of the day, with one hour in between.

35.2.4 Mandatory activity II: Team building in Estenstadmarka with Luftkrigsskolen

The purpose of this day was to make a stronger team of the project group. The organization of this day seemed very thought through. After one hour of briefing on campus, buses transported us to Estenstadmarka where each were assigned two people from from Luftkrigsskolen as guides for the day.

The arena for the team building is a completely different arena than is the usual scene for us working together. The good thing with moving the arena like this, is that we learned to collaborate without regard to the context we're in. The tasks given were varied, and forced us to collaborate and explore sides of ourselves and the others that we were'nt necessearly aware of. The project group has used the experiences from this day, directly or indirectly throughout the project.

Apart from the heavy raining, which noboday can be blamed for, the group agrees that this day was a sucess.

Chapter 36
Organization and Cooperation

This chapter deals with how we organized the group, and in which way the final result has been affected by our organization. We will discuss both internal cooperation and the cooperation with the customer and the advisors.

36.1 Internal Cooperation

Before starting to produce the project directive and the pre study we established different roles for each group member. The roles all had specific and clearly identifiable responsibilities through-out the project. However, we also made it clear that these roles not necessarily implied having the entire workload related to that role but rather a administrative responsibility. Establishment of different responsibilities quickly made us ready for producing the first two documents. The approach taken was starting on both documents at once. The three group members with direct construction and implementation responsibilities started on the pre study while the other members finished the project directive. As soon as the project directive was finished the entire group joined up for the pre study. This approach of alternating the working area for all group members has been a key to maintaining a common understanding for the project. The general cooperation in the group has been good. Both the Six Thinking Hats session (see 35.2.3) and the team bulding (35.2.4) established a common agreement on that one important goal is to make everyone's opinion being heard. The group has definitely not been afraid to take discussions, and we feel jointly responsible for all major decisions taken.

36.1.1 Organizing the work process

In addition to "Kundestyrt Prosjekt", all group members have been taking several other courses, with their own lectures, projects, assignments and exams. Since there exist no coordination between the different courses, the workload on group members have varied from week to week. Coordination of the work process has mainly been done in our weekly internal meetings on mondays.

We usually have started the meetings with each group member describing what have been done the past week, which problems that have occurred and which questions that need further discussion in the group before the work can continue. At each meeting we have assigned a set of tasks for each group member (often in pairs), according to project role and available work hours the upcoming week. We have also had a smaller internal meeting after each advisor meeting on wednesdays, for coordination and discussion on corrections on phase documents given by advisor. This routine seems to have worked well for us. By having these two weekly meetings, we have been able to stay updated on each other's working area, and quickly been able to re-organize when needed. Since the weekly work has mainly been done in sub groups, the groups have usually had their own internal work meetings during the week. Most of the time, several sub groups have been represented during the day in the reserved computer lab for the course studentse (see 37.2), so direct communication between group members have been easy. When not present at the same location, all group members have been using an instant messenger service, as stated in 37.1. This way of organizing the work process has made it easy to ask questions, or having smaller discussions without having to wait to next internal meeting.

36.1.2 Organizing the phase documents

When starting on a new project phase, one or two project members have had the responsibility of investigate different solutions on how to execute that phase. This includes finding a set of suitable standards (i.e. the IEEE standard used in several of our documents), or simply drawing a sketch with some ideas on how to best organize the phase. Then discussions on which solution to choose has been done in one of the weekly meetings. By doing so, we feel that the transition from one phase to the next has been been smooth. There has been some comfort in knowing that there were always someone knowing what to do next. And that when discussions started on how to execute a phase, we had already defined a set of alternatives. See chapter 39 for an evalutation of each phase.

36.1.3 Consistency of roles

The internal organization structure which were established in the first phase of the project has remained throughout the project's lifespan. The project leader and document responsible have had the general administrational responsibility. Since this responsibility includes writing the status report, the project leader have always had an overview on the project status, and by that been able to assure that the project was headed in the right direction.

36.1.4 Conflicts and Conflict management

Despite of being seven individuals, with sometimes seven different views on an issue, we have not had any personal conflicts during the project. The reason for this is maybe that we have allowed ourselves to have discussions and considering everyone's point of view on a matter. One other reason for the low conflict level is maybe that we all have found our work areas meaningful, and having had the same high ambitions for the project. We have had an open communication amongst us, and a low threshold for asking each other for help.

36.2 External Cooperation

36.2.1 Cooperation with advisor

The cooperation with our supervisor has been satisfactory. The phase documents we have wanted to be reviewed, has been delivered by hand at 12 each Tuesday. At the advisor meeting the following day at 10, we have been given a thorough examination of the documents. In addition to the specific document related review, we have also been given general advices on difficulties in the project. The advisor has also been available for questions and advice outside the regular meeting hours with quick responses to any questions.

36.2.2 Cooperation with advisor assistant

In the beginning we were uncertain on how to use our advisor assistant, since that role were more informal described in the information compendium given in the course introduction. (See 35.2.1 on review of the compendium) Being early in the project, we did not know that we were supposed to ask for response from the advisor assistant, and therefore we didn't get any. But after having finished the first two phase documents, the project leader had a meeting with the advisor advisor and asked for feedback. From then on, we got a a very in-depth response that has been of great help of the project. So, after the startup difficulties, we have had an open dialogue and good cooperation with our advisor assistant. The thorough and detailed feedback on the phase documents has been of great value for the project group.

36.2.3 Cooperation with the customer

Since the first customer meeting we have had a good and open dialogue with our customer. The first meeting was very informative, and made us quickly ready to define our approach to the assignment. We were at first worried by the lack of face to face meetings available during the three months. But we have managed to arrange several meetings via the previously described IP telephone program Skype [23]. See 37.4 for details. We feel that these remote meetings has been a good replacement for the more ordinary meetings. Perhaps the telephone meetings have been easier to arrange than face to face meetings, even if we were situated in the same city. These meetings have taken place when going through some of the phase documents, or if we felt to clarify requirements for EasyIT.

But of course, it's better to meet face-to-face. As we were restricted to one meeting of this nature, we chose to meet during the requirements specification phase. The meeting lasted one work day, and we managed to go through all the different requirements in a way that clarified the many questions we had at that time. In short, the customer has been highly available through e-mail and by telephone. We have been given quick responses on all important questions.

Chapter 37 Carrying out the practical

This chapter evaluates how we carried out the different practical matters if the project.

37.1 Managing group resources

The project group consisted of 7 members. Each member were expected to put down 24 hours of work each project week. During the 13-week timespan of the project, the group estimated to work 2170 hours in total.

We developed a system in MS Excel, where each project member had one work sheet file. Every tuesday, each member filled in hours spent on each phase. A separate worksheet summarized these hours, and presented it in an easy to read graph, and also compared the hours actually spent against what was estimated. This graph, included in Appendix B, was also included in every status report.

We also developed a forum on the internet, where the group could announce important matters, questions etc. Also, a common calendar was made available for the members both to read and to make entries on important matters.

We have used the time keeping system in Excel actively, due to its simplicity. Filling out is done in no time, and the results are immediate. The forum on the internet was also constructed with the same goal in mind - simplicity. However, as we have been spending over 2000 hours together this semester, the need for additional conversations over a forum turned out to not be that high. Instead, we have been using instant messenger tools for communicating when not being at the same place. The calendar however, has been used a lot. We see this as a simple, yet powerful tool that makes it easy for all members to be up to date on where and when of important matters.

We did also use the common email adress `kpro10@idi.ntnu.no` as a reply-to-adress when sending mail to the advisors and/or the customer. The reason for this is that we wanted important mail to be distributed to all the group members.

37.2 Hardware and software resources

The course students were given the fifth floor in the P15 building, including meeting rooms,printing facilities and a computer lab. Although it sounded promising, the conditions for thinking and problem solving were not optimal from our point of view, as the computer lab was crowded and rather noisy. Also, most of the programs we needed (see next chapter for details), including Visual Studio, were not installed on the machines, resulting in the need for Remote Desktop Connection to one of the ts-stud servers and the group's allocated computer mentioned in the previous chapter. This resulted in that when coming to the design and implementation phases, we decided to move. We found ourselves a more silent computer lab, "Kalhari", at "Verkstedteknisk". This is not the usual place to find computer students, and that suited us well.

37.3 Meeting room resources

Every Monday at 12, as stated in 36.1.1 we have kicked off the week with an internal meeting. Even though, as stated in the previous section, the fifth floor in the P15 building had some meeting rooms available, we found it necesseary to make a reservation for another room so that we were guaranteed to have a place to meet. We got the room *A161* for these meetings.

Every Wednesday we have met with our advisors. (As stated in 36.1.1). Reservation for the room used for this purpose was done by the advisors.

For the one meeting we've had with the customer, we used the meeting room at P15. See the next section for the other ways we communicated with the customer.

37.4 Remote meeting resources

Meetings with our customer was mainly carried out over internet meetings due to the distant location of our workplaces. The customer was not able to physically meet with us more than once, so we had to figure out a way to make the communcation channel as open as possible. The first, obvious solution was to use some kind of netmeeting. The customer, sat however behind a firewall that didn't allow much traffic. We then tried the program Skype [23], that provided crystal clear sound and worked well with ABB's firewall. We set up one computer with good speakers and a microphone, and it was like the customer were in the room with us. Of course, it was still a substitute, but with thorough preparation from both parts (ie. all necesseary documentation was handed over the day before, only one group member spoke at a time) the meetings were sucessfully carried out.

CHAPTER 38
TOOLS

This chapter evaluates the software tools we have used throughout the phases of the project.

38.1 DOCUMENTS

All documents generated by the project have been written in LaTeX shared within the group using CVS. ER-diagrams are created with ER-modeller, all other diagrams have been drawn with Visio.

38.1.1 LaTeX

LaTeX has been described earlier, see 6.1.1. Clearly, our use of LaTeX have lead to a heavy overhead concerning layout. This document has probably sections where placement of tables and figures look somewhat strange, even though we have tried to avoid that. You can indicate where you want the placement of tables and figures, but where it actually ends up is in the end decided by LaTeX. On the other hand, we have without doubt had an easy time managing our documents, because of the includeable structure, that is a directory structure: report/part/chapter/. Making templates and using these have been a major strength, both in standalone documents (like minutes and status-reports) and the project report (which have a root-file defining layout parameters and styles). Since LaTeX is stored as plain text, it can be written on any platform using any text editor. This have been greatly appreciated by the Linux-using members of the team, and is also enabling us to use CVS. Documentation have been easily accessible through search-engines on the web.

38.1.2 CVS

CVS has been described earlier, see 6.1.2. As mentioned above, LaTeX as a plaintext typesetting tool enables us to use CVS. This have indeed been very helpful; When team members have been working from home or at different locations, they have been able to edit files and publish these changes without having to clear this operation with other members. The tool got quickly accepted within the group and no difficulties worth noting have occurred. The overhead using CVS were for Microsoft Windows using team members; We found no CVS-tool that would work with our repository, so for each cvs operation (add, remove, commit, update), one would have to log on to a *nix-server mounting the same network share as where the checkout where, and perform the operations with the native cvs-program. As there were no further attempts to get a windows-friendly cvs-program working, the overhead is to be considered insignificant.

38.1.3 ER-modeller

This program is written by Vetle Valebjørg and Kristian Skogstrøm at NTNU. This is a fairly simple java-program available from web [25]. It is able to print diagrams, and using a PDF printer, we got nice vector-graphic files to include in our report. This program were only used to draw a handful of figures.

38.1.4 Visio

Visio is a Microsoft product aimed at supporting a broad range of modelling languages, and we employed mainly UML class diagram and dfd. We also employed a Visio stencil for drawing APM diagrams. Visio were previously known to all members of the team, so using it were pretty straightforward and without suprises. The major problem we experienced were with file types; Visio could not save files as PDF (which were our favourite format for inclusion), and printing to a PDF writer gave corrupted labels. We solved this by exporting from Visio as Encapsulated PostScript, and converting this to PDF using Acrobat Distiller.

38.1.5 Group directory

We were provided a group directory by the students' webserver at NTNU with a quota of 60MB. This proved to be too little, and we got an extension to 100MB by request. This directory was used to store the CVS repository and the SourceSafe directory, and keep shared files between us, like time lists and time budgets (as desribed in 37.1).

38.1.6 Backup

Fortunately we never needed this. It was done by a script executed at 7 am and 4 pm every day, and basically took the whole group directory and compressed it into a single file, which was stored at a separate location.

38.2 Program code

All development of program code has been done within the .NET environment; MS Visual Studio for implementation and MS Virtual SourceSafe for sharing.

38.2.1 MS Visual Studio

When developing in .NET, MS Visual Studio is an obvious choice of developer environment. This IDE is packed with a wide range of features making the implementation easier for the developer. Fortuately, students at NTNU have free access to most Microsoft products, including Visual Studio and MS Virtual SourceSafe.

38.2.2 MS Visual SourceSafe

MS Visual SourceSafe (VSS) is a tool for avoiding conflicts caused by simultaneous changes to the code done by the developers. MS Visual Studio comes with built-in support for VSS, and therefore we used VSS as a sharing tool for our code.

As as mentioned in 35.2.1, VSS wasn't installed on the computer lab. We installed it on our server and kept all source files there. However, VSS requires that it is installed on the machine where the developement is done as well. We solved this by installing MS Visual Studio and VSS on our home computers, and worked on those computers at school via Remote Desktop Connection. Unfortunately only four out of seven group members had this option. But in addition, one member could work remote on the server. As the implementation phase went by, we used four members on pure implementation and three members on documentation, so it turned pretty sucessfull after all. Perhaps we hadn't been more effective with all members implementing at the same time either.

Chapter 39 Phases

In this chapter we briefly describe what happened in each phase.

39.1 Project directive

The project directive is a document that holds information of the accomplishment of the project. This is meant as a reference for team members, and can be changed any time during the project. We made no major changes in this document. The content of this document were suggested by an appendix to the compendium we got at the first day of the project, and we followed these guidelines with minor exceptions. The phase consisted of many other (not relevant to the document) things, like getting CVS and LaTeX up and running, organizing weekly meetings, team building and so on. It was generally a good formal start to the project.

39.2 Pre study

As the customer wanted a system buildt on a technology which the team members were unfamiliar with, much of the pre study-phase were used to get to know this technology, and this really helped us a great deal in understanding the problem. There were no existing solutions that we could use, and ABB wanted us to develop our own anyway. This phase consisted of a lot of reading, and re-entering the information into the prestudy document. We did not experience any problems in this phase.

39.3 Requirements specification

The IEEE830 standard ("Recommended Practice for Software Requirements Specifications") were employed to help put the requirements in a reasonable structure. As we had a good understanding of the technology and how ABB wanted the problem solved, this phase was more writing than thinking. Still, we figured out a misunderstanding in this phase regarding the applications. We'd though that an application were sort of a program that ran on a separate computer with GUI, connecting to our server. What ABB meant was that an application were more like a script that could be run headlessly on the server, with the results possibly presented on web. This clarification eased our job a great deal. All requirements got a priority, which enabled us to not implement certain requirements if we ran out of time in the implementation-phase.

39.4 Design

We also employed an IEEE standard in this phase: IEEE1016 ("Recommended Practice for Software Design Descriptions"). We found this standard difficult to use, and think we might have been better off with another structure. As we had a good understanding of how to implement the system, we kept the document as it is. We discovered that the APP API-module rather should be considered as an entity within the Data Collector, but this did not involve any problems. We decided to keep the previous documents the way they were written, and just comment the change in the introduction to this part. This discovery did not lead to any problems. The completion of the phase-document was delayed since we were uncertain if it were complete, even thought the phase was not.

39.5 Implementation and testing

The implementation was done by two groups consisting of two persons, one group on each module. We feel that this partition were appropriate, because it allowed the groups to have close and consistent communication. It also allowed for easy division of work. The system were implemented top-down, starting with a skeleton and filling this with functions. This was done in an incremental manner. We experienced this as a very clean and tidy way of development, and we were actually able to implement more functionality than what was specifyed in the requirements specification. There were no problems in this phase.

39.6 Project evaluation

After the implementation was done and the product approved by the customer, we felt that the project was about coming to an end. However, we still had two phases to go, and should not relax. This phase contained testing with customer and writing of implementation- and evaluation document. We experienced no problems or delays during this phase.

39.7 Project presentation

This phase had not yet started when this evaluation was completed.

Chapter 40 Progression

This chapter will describe how the progression in the project has been, compared to the initial plan.

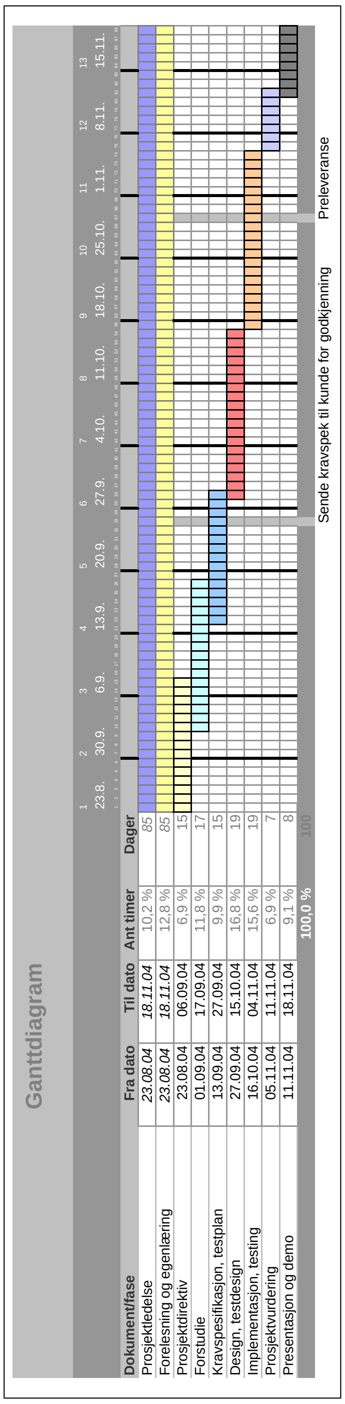
40.1 Progression and milestones

The overall project plan was set during the making of the project directive. At that time, we had little knowledge on the group's performance and the project scope. The plan was made based on previous project experience, and by analyzing earlier reports from "Kundestyrt Prosjekt". Table 40.1, shows the project plan with milestones for each phase. We have entered the actual milestone dates for the different phases in the third column.

Milestone	**Planned Date**	**Actual Date**
Finished project directive (planning phase)	Sep 06. 2004	Sep 09. 2004
Finished pre studies	Sep 17. 2004	Sep 24. 2004
Send requirements specification for approval to client	Sep 24. 2004	Oct 01. 2004
Finished requirements specification	Sep 27. 2004	Oct 13. 2004
Finished design of overall design	Oct 05. 2004	Oct 06. 2004
Finished design documents	Oct 15. 2004	Oct 30. 2004
Pre deliverance of pre study and requirements specification	Oct 28. 2004	Oct 28. 2004
Finished implementation and test documentation	Nov 04. 2004	Nov 07. 2004
Finished project evaluation	Nov 11. 2004	Nov 12. 2004
Presentation	Nov 18. 2004	Nov 18. 2004

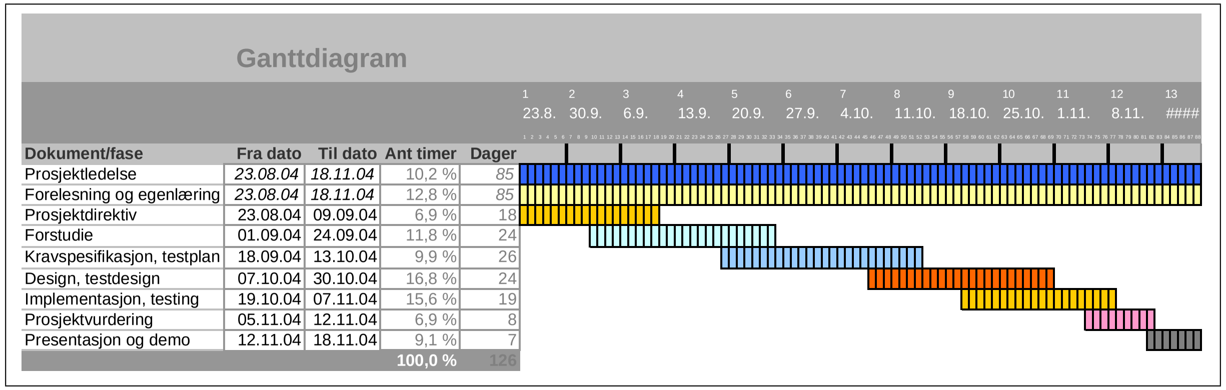
Table 40.1: Milestones

As seen in Table 40.1, we relative early got a delay varying from one to two weeks. We clearly underestimated the work load on the first couple of phases. The gantt diagram (generated after the phases were finished), further illustrates how the different phases have been executed. The diagram shows that several phases were run in parallel. This happened often because of the complementary work needed on the documents, after the main work was done. In these phases, a bigger part of the group moved on to the next phase while often two persons did the final editing from the previous phase. The complementary work done on the phase documents were an iterative process. New versions were made and reviewed until the document was considered finished. The reviewing process was more time consuming than expected, and may explain some of the reasons for the delayed milestone dates.



Dokument/fase	Fra dato	Til dato	Ant timer	Dager
Prosjektledelse	*23.08.04*	*18.11.04*	10,2 %	*85*
Forelesning og egenlæring	*23.08.04*	*18.11.04*	12,8 %	*85*
Prosjektdirektiv	23.08.04	06.09.04	6,9 %	15
Forstudie	01.09.04	17.09.04	11,8 %	17
Kravspesifikasjon, testplan	13.09.04	27.09.04	9,9 %	15
Design, testdesign	27.09.04	15.10.04	16,8 %	19
Implementasjon, testing	16.10.04	04.11.04	15,6 %	19
Prosjektvurdering	05.11.04	11.11.04	6,9 %	7
Presentasjon og demo	11.11.04	18.11.04	9,1 %	8
			100,0 %	100

Figure 40.1: Planned Gantt Diagram



Dokument/fase	Fra dato	Til dato	Ant timer	Dager
Prosjektledelse	*23.08.04*	*18.11.04*	10,2 %	*85*
Forelesning og egenlæring	*23.08.04*	*18.11.04*	12,8 %	*85*
Prosjektdirektiv	23.08.04	09.09.04	6,9 %	18
Forstudie	01.09.04	24.09.04	11,8 %	24
Kravspesifikasjon, testplan	18.09.04	13.10.04	9,9 %	26
Design, testdesign	07.10.04	30.10.04	16,8 %	24
Implementasjon, testing	19.10.04	07.11.04	15,6 %	19
Prosjektvurdering	05.11.04	12.11.04	6,9 %	8
Presentasjon og demo	12.11.04	18.11.04	9,1 %	7
			100,0 %	126

Figure 40.2: Actual Gantt Diagram

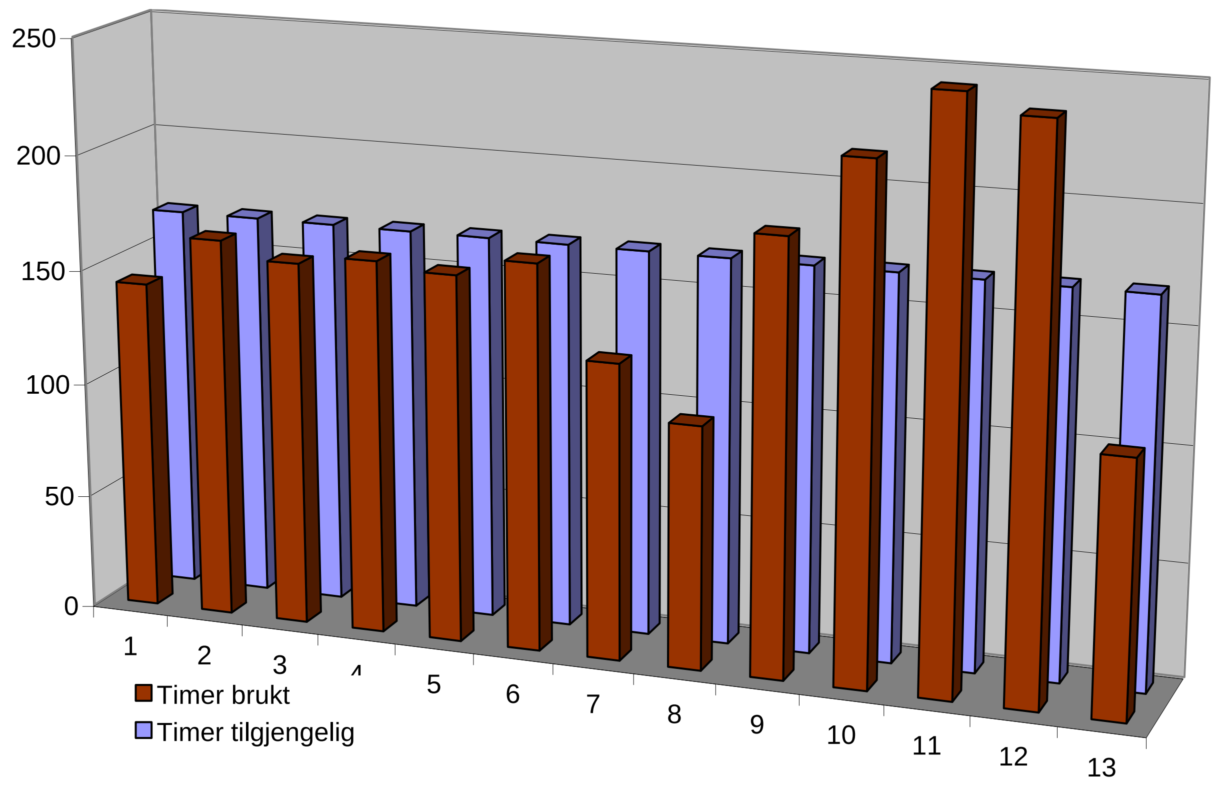


Figure 40.3: Work Hours

The most time consuming phase, was the making of requirements specification and test plan. We planned to use 15 days but ended up with 26 days in this phase. Although 13 of the 26 days of were overlapping other phases, we had clearly underestimated the work having to be done in this phase. At the time we made the gantt diagram and set the milestone dates we felt quite confident in that specifying the requirements should not take more than 15 days. A possible explanation for the miscalculation is that we had a more shallow understanding for the system when planning, than what we had when starting the requirements specification process. During the phases preceding the requirements specification, we gradually got a deeper understanding for the system and revealed more complex problems to be addressed.

40.2 Time Consumption

All project members attend several other courses than "Kundestyrt Prosjekt". All courses have had their own projects, assignments and exams. This has naturally affected the work processes in our project. Figure 40.3 illustrates the total working hours for each week on the project. The first six weeks are fairly stable and according to our expectations. In week 7 and 8, many group members had mid-term exams and larger assignments, which is clearly shown in Figure 40.3. This is also an explanation for the delay on completion of the requirements specification document. In the following three last weeks of the project we made up for the hours lost. In average the project had 178 work hours per week from the 7'Th to the 12'Th week of the project, which is only about 10 hours more per week than planned. Since the project is ending on a Thursday, there exist a difference between the estimated and actual work hours in week 13. In general the hours in a normal working week is counted from Monday. The estimation and explanation for the time spent implementing is further described in the evaluation summary.

40.3 About implementation estimate

Estimated person hours for the implementation phase are written in 19.3. To repeat this; The results were 3045 hours with the Project Estimation Method[5]. As we thought this sounded way too much, we did our own “expert estimation”, which resulted in 740 hours. As seen in B, both of these were too high. In addition to implementing all the requirements agreed upon, we implemented some functions the customer probably will find useful together with what was requested. We have in total spent 313 person hours implementing. This number is subject to change in case of last minute modifications to the system.

Chapter 41 Summary

Being now in our fourth year and having had several project assignments earlier, we feel that this course have had the highest relevance. It is also the first time where the final grade for the course relies on our performance in the project alone. As usual we have additional exams, and the project is just counted as an exercise. The fact that we are dealing with a real customer and have tried to solve a real problem has also been inspiring for us. This has made us put in an extra effort which by it self has made the experience even more instructive. Besides the technical details we have learned a lot of how a real project is carried out.

41.1 Project group matters

41.1.1 What are we particularly satisfied with?

Development model: Even thought we have used sort of an incremental model for our documents, the whole project has followed the waterfall development model. This have worked out well, thanks to excellent guidance from our advisers.
Cooperation within the group: Initially, there were few people which knew each other within the group. As the project passed, all members of the group grew together in a close manner, and it has stayed that way. There have been no critical conflicts within the group, and everyone has cooperated in a satisfying manner when asked to.
The customer: We consider ourselves lucky because our customer was easy and pleasant to work with, and he had a consistent image of what he wanted built throughout the project. Also, the task was interesting itself, giving us a glimpse into industrial use of information technology.

41.1.2 What could have been done better?

Requirements specification document: This document took more than one review to complete, and could have been more thoroughly worked through the first time. However, having experienced this, we *will* probably do better in our next project.
Design document: Following an IEEE standard and writing this document was possibly unnecessary. We had a good understanding on how to implement, and the standard just complicated our process of documenting this. The document were changed a lot and the completion of it delayed. The next time a document of this sort is to be written, we will spend more time on *how to* to it, instead of blindly following an IEEE path.

41.2 Course matters

41.2.1 What are we particularly satisfied with?

The weekly meetings with our advisors : Our advisor has met prepared for every meeting with comments on what has been delivered. At first we thought that the practice with physically giving the advisor everything we wanted reviewed was somewhat cumbersome, but later we realized that this practice made the weekly deleviries more formal. Our advisor assistant gave us higly appreciated, detailed (down to grammar corrections) feedback.
The teambuilding in Estenstadsmarka : The team building made the group a group. As most of us didn't know each other, we were more like seven guys which were put together than a real group. This day changed that. The heavy rain problably made it even more effective.

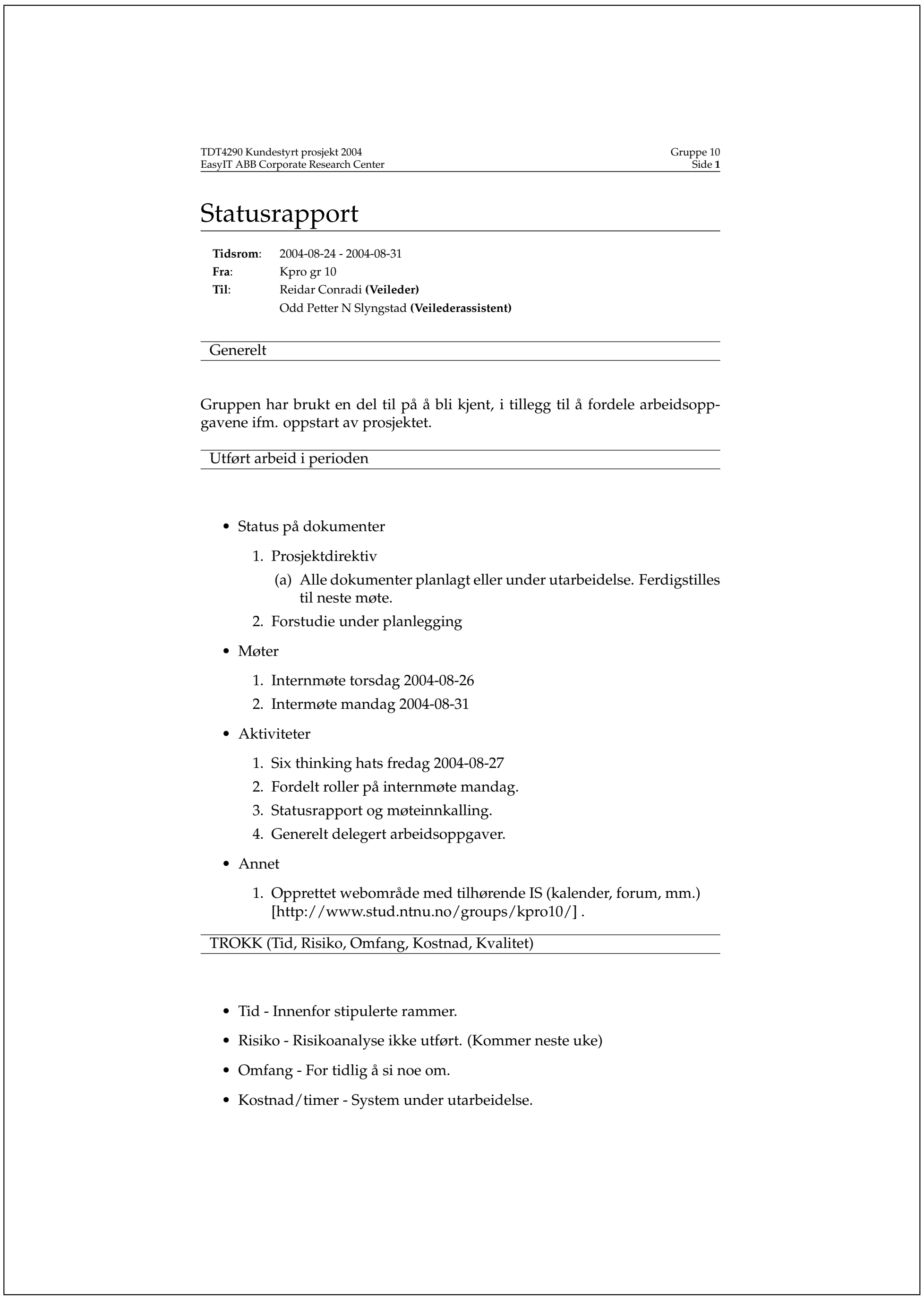
41.2.2 What could have been done better?

The course compendium : This compendium has much information, but appears poorly organized.
See 35.2.1 for more details on the evaluation of the course.

Appendix K
Status reports

This appendix provides all status reports the group has made during the project life span. The purpose of providing these is to document how we have been working. The status report has been written in Norwegian, as all our correspondence with our advisors has been in Norwegian.

The status reports also contained additional attachments not included here, such as a risk diagram and a Gantt diagram found in the appendices of the Project Directive (Gantt diagram at B, risk diagram at C), and also an updated diagram of hours spent.See the final diagram at Chapter 40.



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **1**

Statusrapport

Tidsrom:	2004-08-24 - 2004-08-31
Fra:	Kpro gr 10
Til:	Reidar Conradi **(Veileder)**
	Odd Petter N Slyngstad **(Veilederassistent)**

Generelt

Gruppen har brukt en del til på å bli kjent, i tillegg til å fordele arbeidsoppgavene ifm. oppstart av prosjektet.

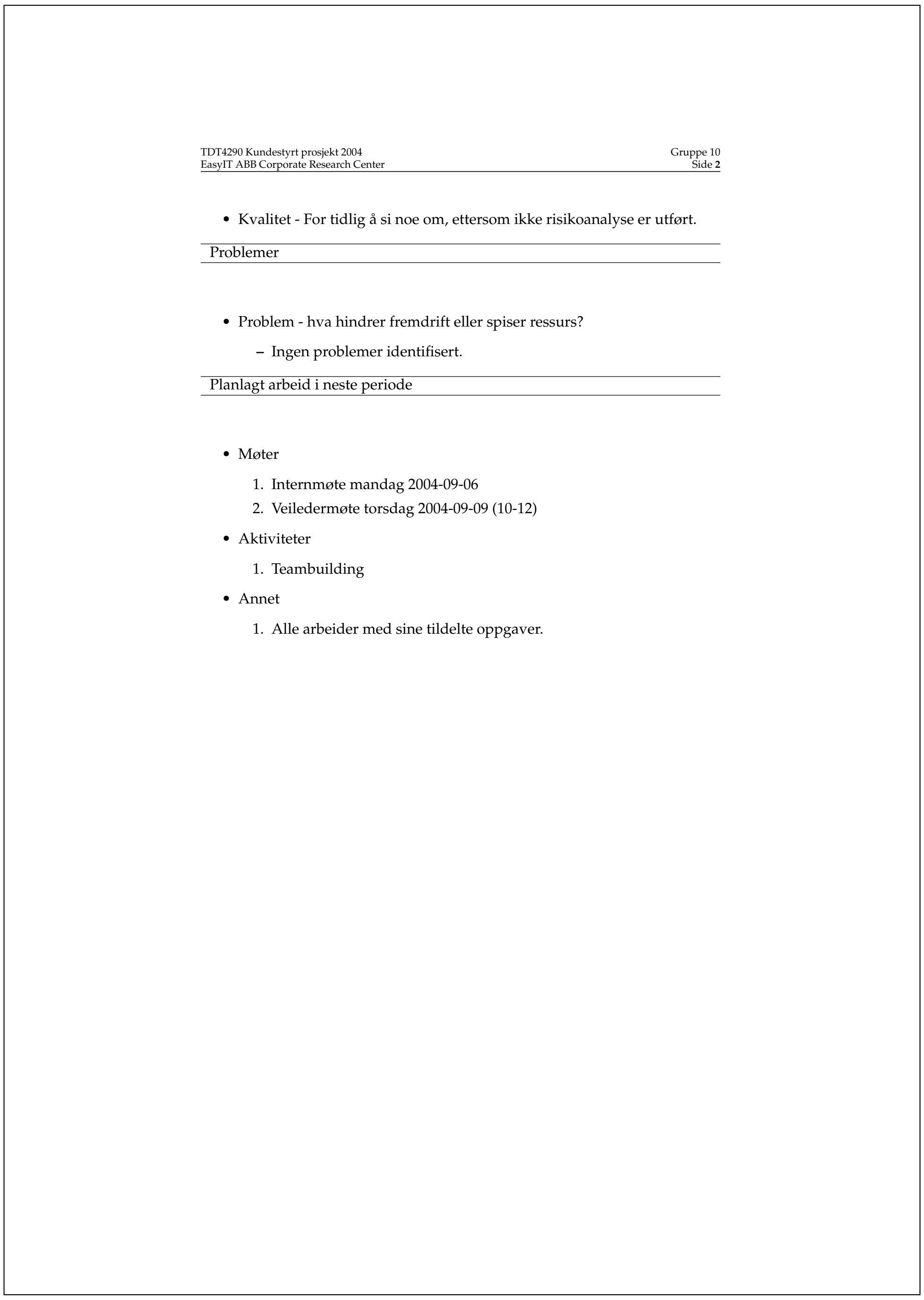
Utført arbeid i perioden

- Status på dokumenter
  1. Prosjektdirektiv
     (a) Alle dokumenter planlagt eller under utarbeidelse. Ferdigstilles til neste møte.
  2. Forstudie under planlegging
- Møter
  1. Internmøte torsdag 2004-08-26
  2. Intermøte mandag 2004-08-31
- Aktiviteter
  1. Six thinking hats fredag 2004-08-27
  2. Fordelt roller på internmøte mandag.
  3. Statusrapport og møteinnkalling.
  4. Generelt delegert arbeidsoppgaver.
- Annet
  1. Opprettet webområde med tilhørende IS (kalender, forum, mm.) [http://www.stud.ntnu.no/groups/kpro10/] .

TROKK (Tid, Risiko, Omfang, Kostnad, Kvalitet)

- Tid - Innenfor stipulerte rammer.
- Risiko - Risikoanalyse ikke utført. (Kommer neste uke)
- Omfang - For tidlig å si noe om.
- Kostnad/timer - System under utarbeidelse.

Figure K.1: Status report from 2004-08-24 to 2004-08-31- page 1



TDT4290 Kundestyrt prosjekt 2004 — Gruppe 10
EasyIT ABB Corporate Research Center — Side 2

- Kvalitet - For tidlig å si noe om, ettersom ikke risikoanalyse er utført.

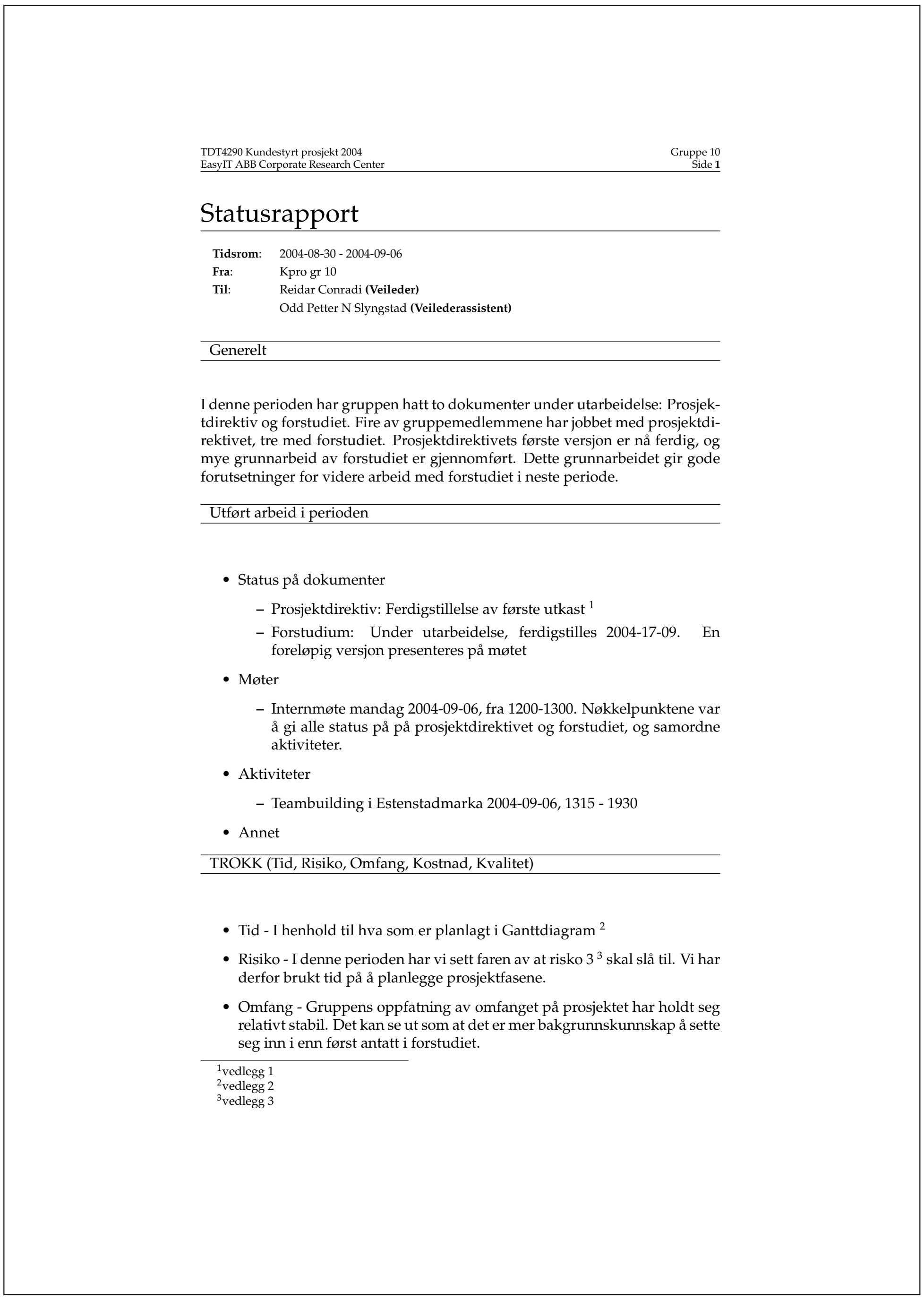
Problemer

- Problem - hva hindrer fremdrift eller spiser ressurs?
  - Ingen problemer identifisert.

Planlagt arbeid i neste periode

- Møter
  1. Internmøte mandag 2004-09-06
  2. Veiledermøte torsdag 2004-09-09 (10-12)
- Aktiviteter
  1. Teambuilding
- Annet
  1. Alle arbeider med sine tildelte oppgaver.

Figure K.2: Status report from 2004-08-24 to 2004-08-31 - page 2



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **1**

Statusrapport

Tidsrom:	2004-08-30 - 2004-09-06
Fra:	Kpro gr 10
Til:	Reidar Conradi **(Veileder)**
	Odd Petter N Slyngstad **(Veilederassistent)**

Generelt

I denne perioden har gruppen hatt to dokumenter under utarbeidelse: Prosjektdirektiv og forstudiet. Fire av gruppemedlemmene har jobbet med prosjektdirektivet, tre med forstudiet. Prosjektdirektivets første versjon er nå ferdig, og mye grunnarbeid av forstudiet er gjennomført. Dette grunnarbeidet gir gode forutsetninger for videre arbeid med forstudiet i neste periode.

Utført arbeid i perioden

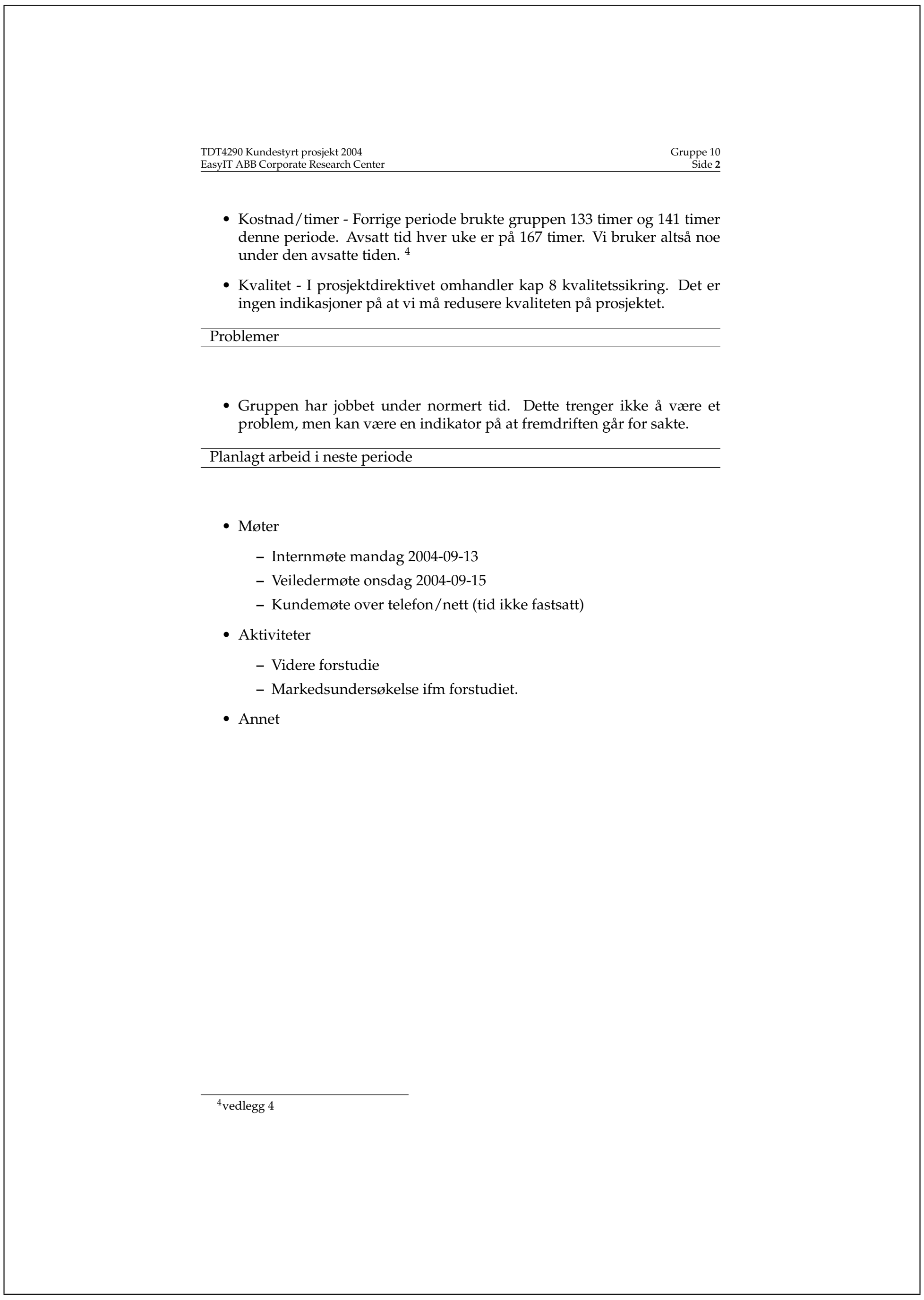
- Status på dokumenter
  - Prosjektdirektiv: Ferdigstillelse av første utkast [1]
  - Forstudium: Under utarbeidelse, ferdigstilles 2004-17-09. En foreløpig versjon presenteres på møtet
- Møter
  - Internmøte mandag 2004-09-06, fra 1200-1300. Nøkkelpunktene var å gi alle status på på prosjektdirektivet og forstudiet, og samordne aktiviteter.
- Aktiviteter
  - Teambuilding i Estenstadmarka 2004-09-06, 1315 - 1930
- Annet

TROKK (Tid, Risiko, Omfang, Kostnad, Kvalitet)

- Tid - I henhold til hva som er planlagt i Ganttdiagram [2]
- Risiko - I denne perioden har vi sett faren av at risko 3 [3] skal slå til. Vi har derfor brukt tid på å planlegge prosjektfasene.
- Omfang - Gruppens oppfatning av omfanget på prosjektet har holdt seg relativt stabil. Det kan se ut som at det er mer bakgrunnskunnskap å sette seg inn i enn først antatt i forstudiet.

[1]vedlegg 1
[2]vedlegg 2
[3]vedlegg 3

Figure K.3: Status report from 2004-08-30 to 2004-09-06 - page 1



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **2**

- Kostnad/timer - Forrige periode brukte gruppen 133 timer og 141 timer denne periode. Avsatt tid hver uke er på 167 timer. Vi bruker altså noe under den avsatte tiden. [4]
- Kvalitet - I prosjektdirektivet omhandler kap 8 kvalitetssikring. Det er ingen indikasjoner på at vi må redusere kvaliteten på prosjektet.

Problemer

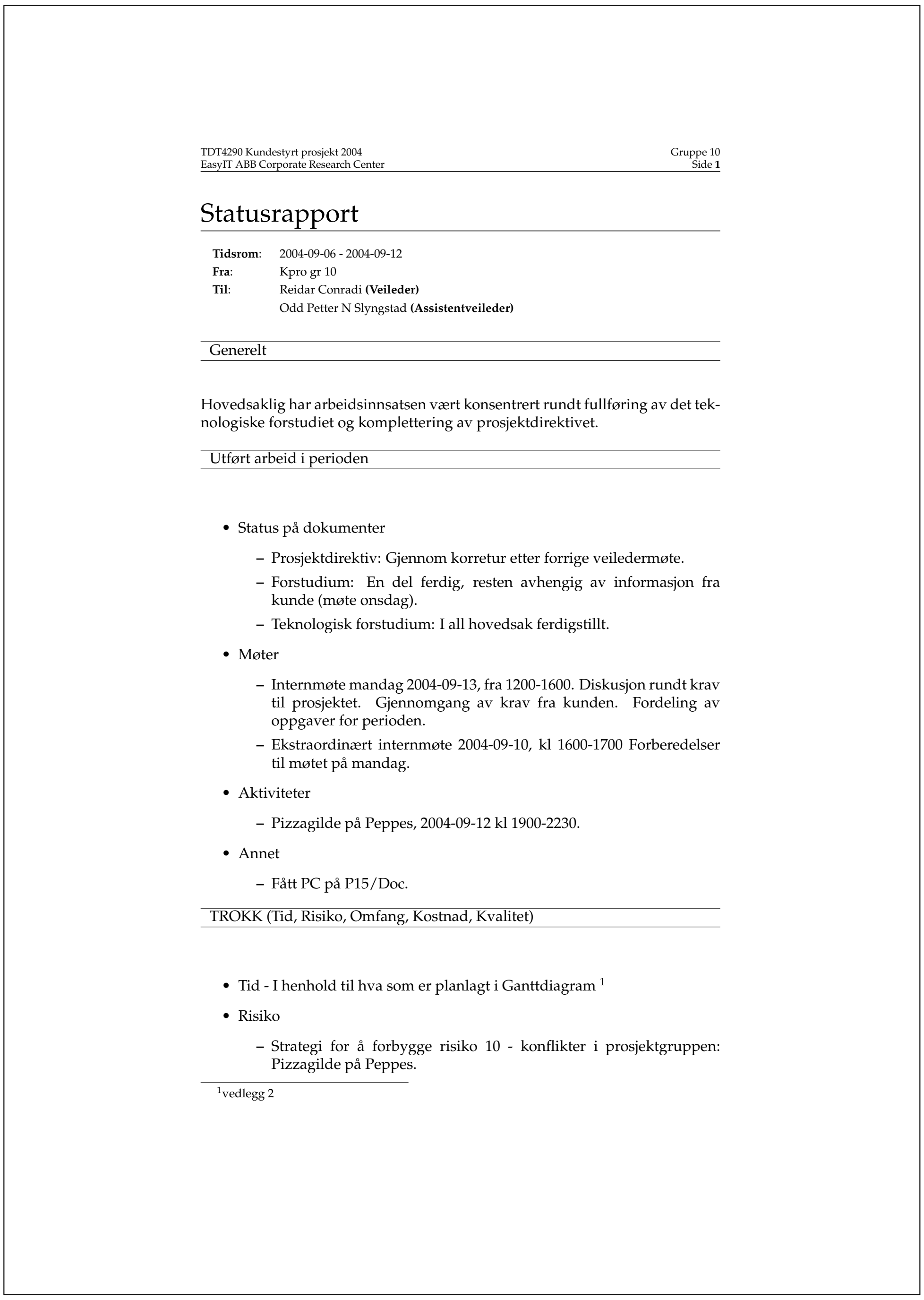
- Gruppen har jobbet under normert tid. Dette trenger ikke å være et problem, men kan være en indikator på at fremdriften går for sakte.

Planlagt arbeid i neste periode

- Møter
  - Internmøte mandag 2004-09-13
  - Veiledermøte onsdag 2004-09-15
  - Kundemøte over telefon/nett (tid ikke fastsatt)
- Aktiviteter
  - Videre forstudie
  - Markedsundersøkelse ifm forstudiet.
- Annet

[4]vedlegg 4

Figure K.4: Status report from 2004-08-30 to 2004-09-06 - page 2



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **1**

Statusrapport

Tidsrom:	2004-09-06 - 2004-09-12
Fra:	Kpro gr 10
Til:	Reidar Conradi **(Veileder)**
	Odd Petter N Slyngstad **(Assistentveileder)**

Generelt

Hovedsaklig har arbeidsinnsatsen vært konsentrert rundt fullføring av det teknologiske forstudiet og komplettering av prosjektdirektivet.

Utført arbeid i perioden

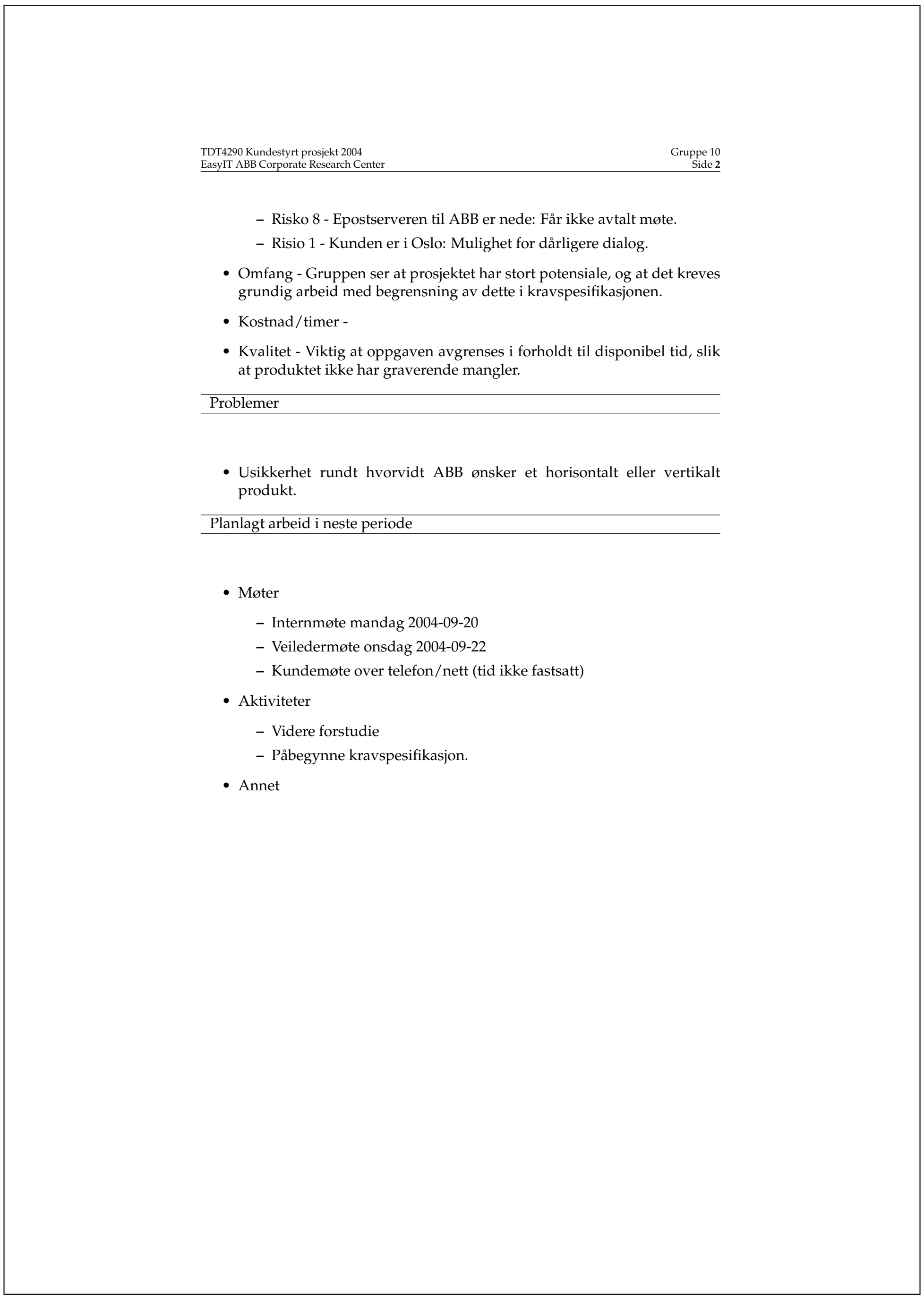
- Status på dokumenter
  - Prosjektdirektiv: Gjennom korretur etter forrige veiledermøte.
  - Forstudium: En del ferdig, resten avhengig av informasjon fra kunde (møte onsdag).
  - Teknologisk forstudium: I all hovedsak ferdigstillt.
- Møter
  - Internmøte mandag 2004-09-13, fra 1200-1600. Diskusjon rundt krav til prosjektet. Gjennomgang av krav fra kunden. Fordeling av oppgaver for perioden.
  - Ekstraordinært internmøte 2004-09-10, kl 1600-1700 Forberedelser til møtet på mandag.
- Aktiviteter
  - Pizzagilde på Peppes, 2004-09-12 kl 1900-2230.
- Annet
  - Fått PC på P15/Doc.

TROKK (Tid, Risiko, Omfang, Kostnad, Kvalitet)

- Tid - I henhold til hva som er planlagt i Ganttdiagram [1]
- Risiko
  - Strategi for å forbygge risiko 10 - konflikter i prosjektgruppen: Pizzagilde på Peppes.

[1] vedlegg 2

Figure K.5: Status report from 2004-09-06 to 2004-09-12 - page 1



TDT4290 Kundestyrt prosjekt 2004 — Gruppe 10
EasyIT ABB Corporate Research Center — Side **2**

  - Risko 8 - Epostserveren til ABB er nede: Får ikke avtalt møte.
  - Risio 1 - Kunden er i Oslo: Mulighet for dårligere dialog.
- Omfang - Gruppen ser at prosjektet har stort potensiale, og at det kreves grundig arbeid med begrensning av dette i kravspesifikasjonen.
- Kostnad/timer -
- Kvalitet - Viktig at oppgaven avgrenses i forholdt til disponibel tid, slik at produktet ikke har graverende mangler.

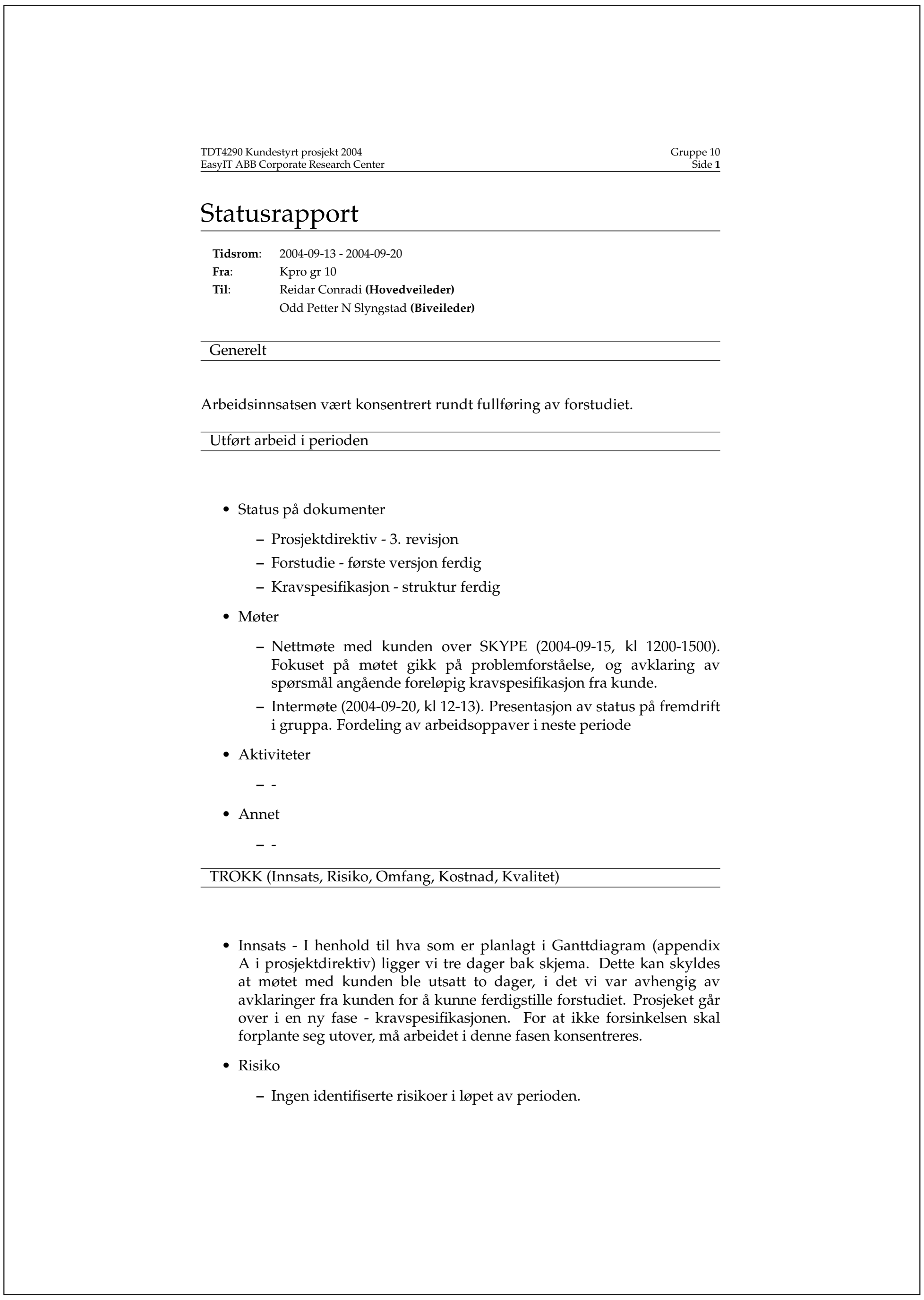
Problemer

- Usikkerhet rundt hvorvidt ABB ønsker et horisontalt eller vertikalt produkt.

Planlagt arbeid i neste periode

- Møter
  - Internmøte mandag 2004-09-20
  - Veiledermøte onsdag 2004-09-22
  - Kundemøte over telefon/nett (tid ikke fastsatt)
- Aktiviteter
  - Videre forstudie
  - Påbegynne kravspesifikasjon.
- Annet

Figure K.6: Status report from 2004-09-06 to 2004-09-12 - page 2



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **1**

Statusrapport

Tidsrom:	2004-09-13 - 2004-09-20
Fra:	Kpro gr 10
Til:	Reidar Conradi **(Hovedveileder)**
	Odd Petter N Slyngstad **(Biveileder)**

Generelt

Arbeidsinnsatsen vært konsentrert rundt fullføring av forstudiet.

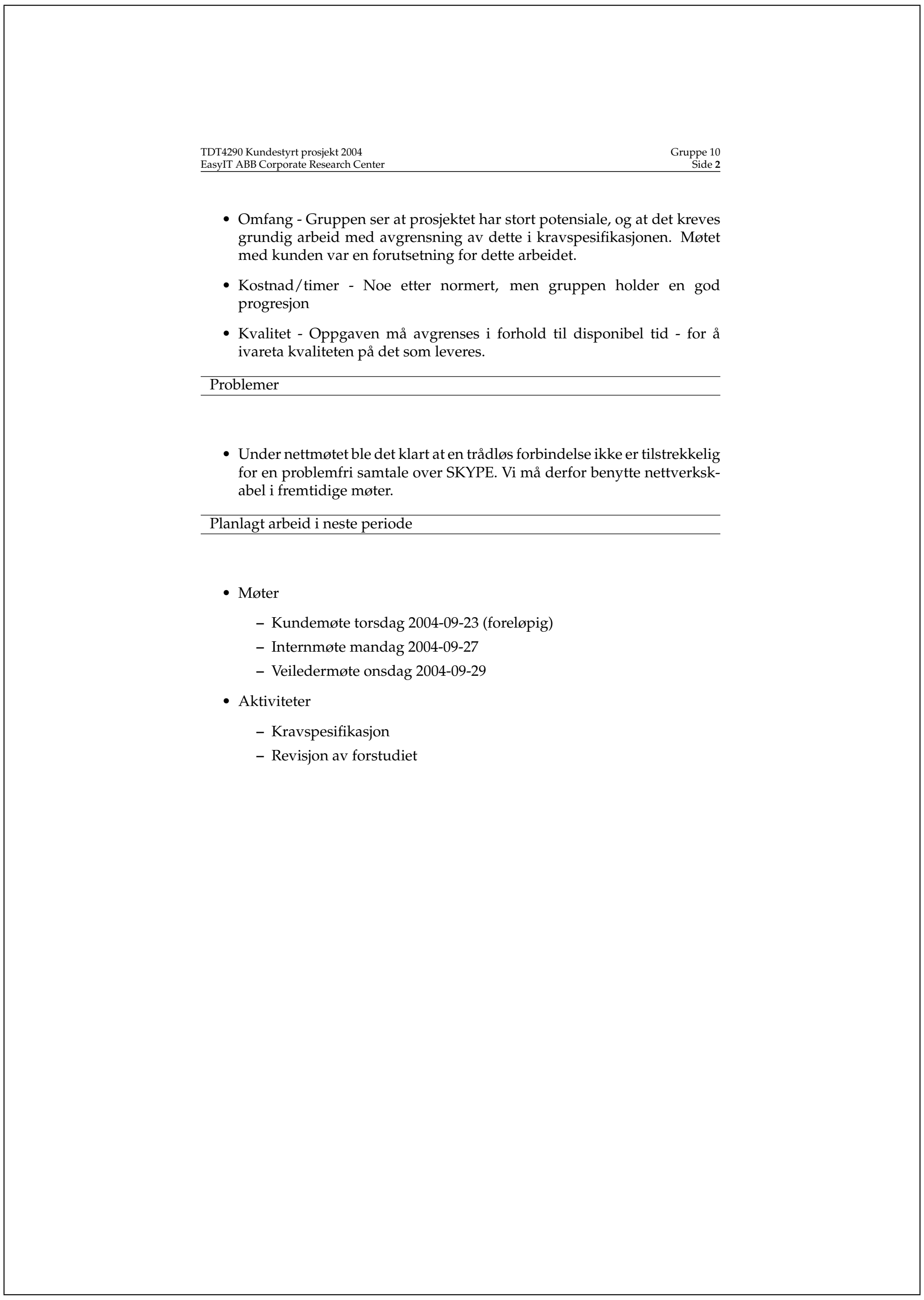
Utført arbeid i perioden

- Status på dokumenter
  - Prosjektdirektiv - 3. revisjon
  - Forstudie - første versjon ferdig
  - Kravspesifikasjon - struktur ferdig
- Møter
  - Nettmøte med kunden over SKYPE (2004-09-15, kl 1200-1500). Fokuset på møtet gikk på problemforståelse, og avklaring av spørsmål angående foreløpig kravspesifikasjon fra kunde.
  - Intermøte (2004-09-20, kl 12-13). Presentasjon av status på fremdrift i gruppa. Fordeling av arbeidsoppaver i neste periode
- Aktiviteter
  - -
- Annet
  - -

TROKK (Innsats, Risiko, Omfang, Kostnad, Kvalitet)

- Innsats - I henhold til hva som er planlagt i Ganttdiagram (appendix A i prosjektdirektiv) ligger vi tre dager bak skjema. Dette kan skyldes at møtet med kunden ble utsatt to dager, i det vi var avhengig av avklaringer fra kunden for å kunne ferdigstille forstudiet. Prosjeket går over i en ny fase - kravspesifikasjonen. For at ikke forsinkelsen skal forplante seg utover, må arbeidet i denne fasen konsentreres.
- Risiko
  - Ingen identifiserte risikoer i løpet av perioden.

Figure K.7: Status report from 2004-09-13 to 2004-09-20 - page 1



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **2**

- Omfang - Gruppen ser at prosjektet har stort potensiale, og at det kreves grundig arbeid med avgrensning av dette i kravspesifikasjonen. Møtet med kunden var en forutsetning for dette arbeidet.
- Kostnad/timer - Noe etter normert, men gruppen holder en god progresjon
- Kvalitet - Oppgaven må avgrenses i forhold til disponibel tid - for å ivareta kvaliteten på det som leveres.

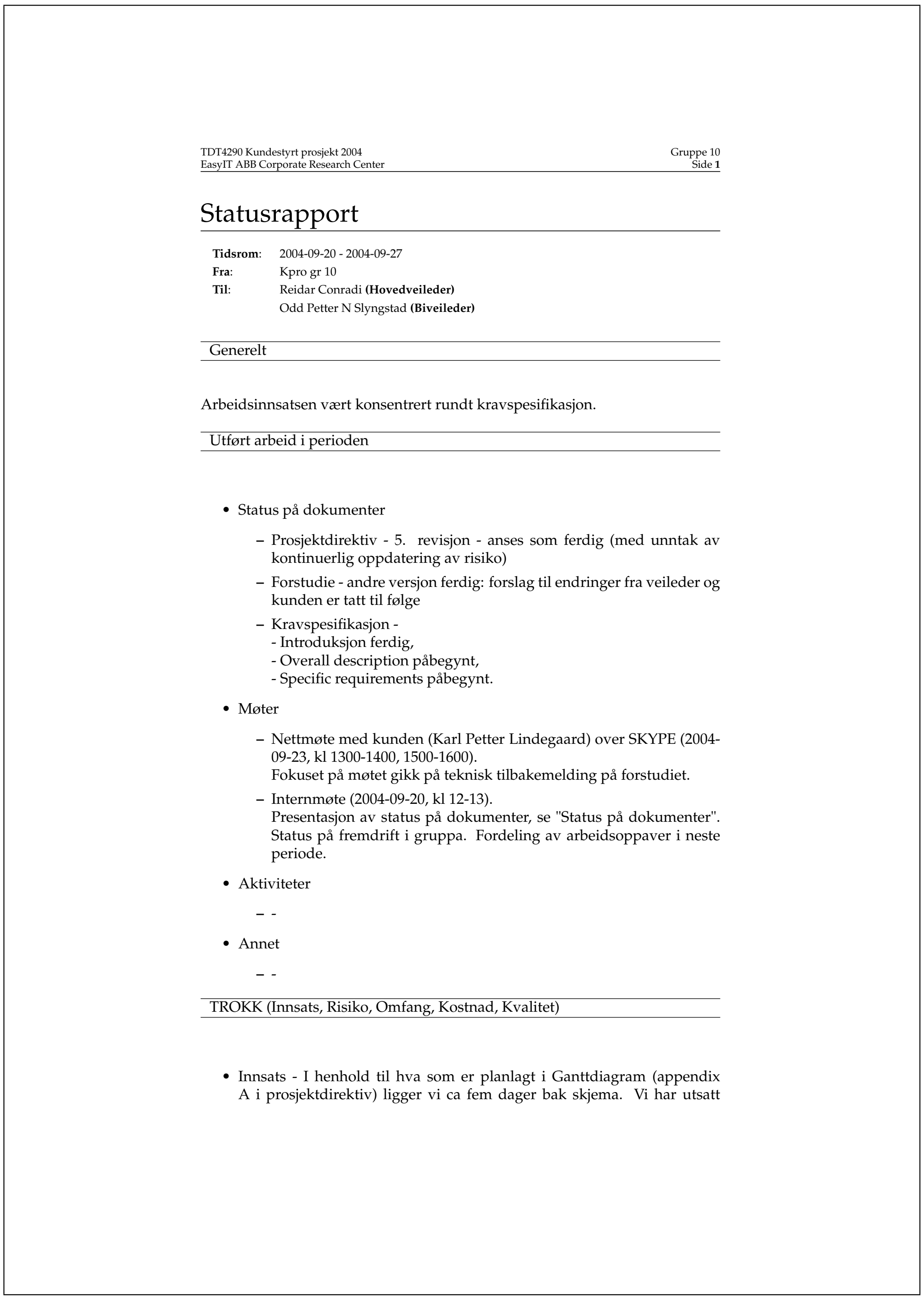
Problemer

- Under nettmøtet ble det klart at en trådløs forbindelse ikke er tilstrekkelig for en problemfri samtale over SKYPE. Vi må derfor benytte nettverkskabel i fremtidige møter.

Planlagt arbeid i neste periode

- Møter
  - Kundemøte torsdag 2004-09-23 (foreløpig)
  - Internmøte mandag 2004-09-27
  - Veiledermøte onsdag 2004-09-29
- Aktiviteter
  - Kravspesifikasjon
  - Revisjon av forstudiet

Figure K.8: Status report from 2004-09-13 to 2004-09-20 - page 2



TDT4290 Kundestyrt prosjekt 2004
EasyIT ABB Corporate Research Center

Gruppe 10
Side **1**

Statusrapport

Tidsrom:	2004-09-20 - 2004-09-27
Fra:	Kpro gr 10
Til:	Reidar Conradi **(Hovedveileder)**
	Odd Petter N Slyngstad **(Biveileder)**

Generelt

Arbeidsinnsatsen vært konsentrert rundt kravspesifikasjon.

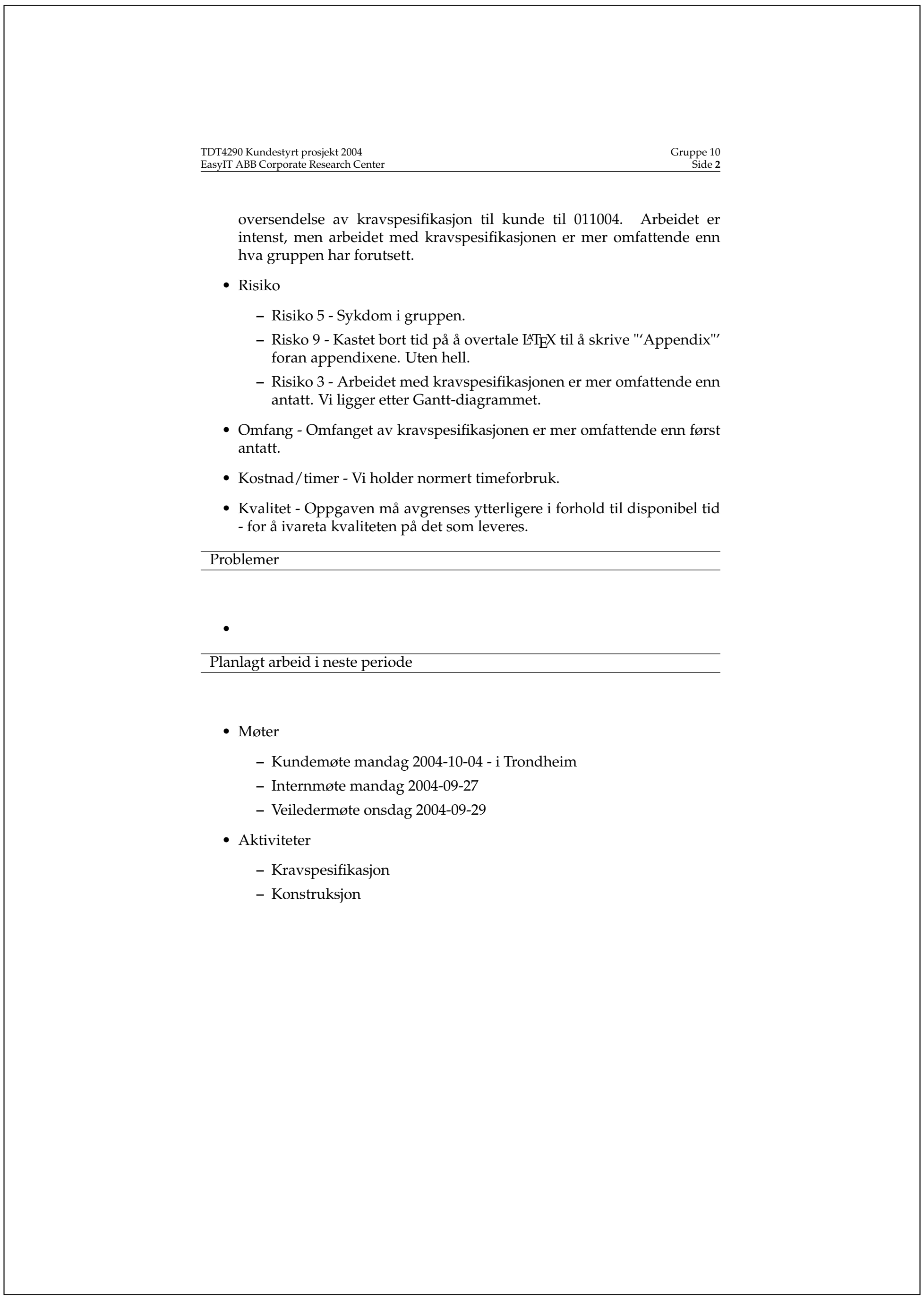
Utført arbeid i perioden

- Status på dokumenter
  - Prosjektdirektiv - 5. revisjon - anses som ferdig (med unntak av kontinuerlig oppdatering av risiko)
  - Forstudie - andre versjon ferdig: forslag til endringer fra veileder og kunden er tatt til følge
  - Kravspesifikasjon -
    - Introduksjon ferdig,
    - Overall description påbegynt,
    - Specific requirements påbegynt.
- Møter
  - Nettmøte med kunden (Karl Petter Lindegaard) over SKYPE (2004-09-23, kl 1300-1400, 1500-1600).
    Fokuset på møtet gikk på teknisk tilbakemelding på forstudiet.
  - Internmøte (2004-09-20, kl 12-13).
    Presentasjon av status på dokumenter, se "Status på dokumenter". Status på fremdrift i gruppa. Fordeling av arbeidsoppaver i neste periode.
- Aktiviteter
  - -
- Annet
  - -

TROKK (Innsats, Risiko, Omfang, Kostnad, Kvalitet)

- Innsats - I henhold til hva som er planlagt i Ganttdiagram (appendix A i prosjektdirektiv) ligger vi ca fem dager bak skjema. Vi har utsatt

Figure K.9: Status report from 2004-09-20 to 2004-09-27 - page 1



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **2**

oversendelse av kravspesifikasjon til kunde til 011004. Arbeidet er intenst, men arbeidet med kravspesifikasjonen er mer omfattende enn hva gruppen har forutsett.

- Risiko
  - Risiko 5 - Sykdom i gruppen.
  - Risko 9 - Kastet bort tid på å overtale LaTeX til å skrive "'Appendix"' foran appendixene. Uten hell.
  - Risiko 3 - Arbeidet med kravspesifikasjonen er mer omfattende enn antatt. Vi ligger etter Gantt-diagrammet.
- Omfang - Omfanget av kravspesifikasjonen er mer omfattende enn først antatt.
- Kostnad/timer - Vi holder normert timeforbruk.
- Kvalitet - Oppgaven må avgrenses ytterligere i forhold til disponibel tid - for å ivareta kvaliteten på det som leveres.

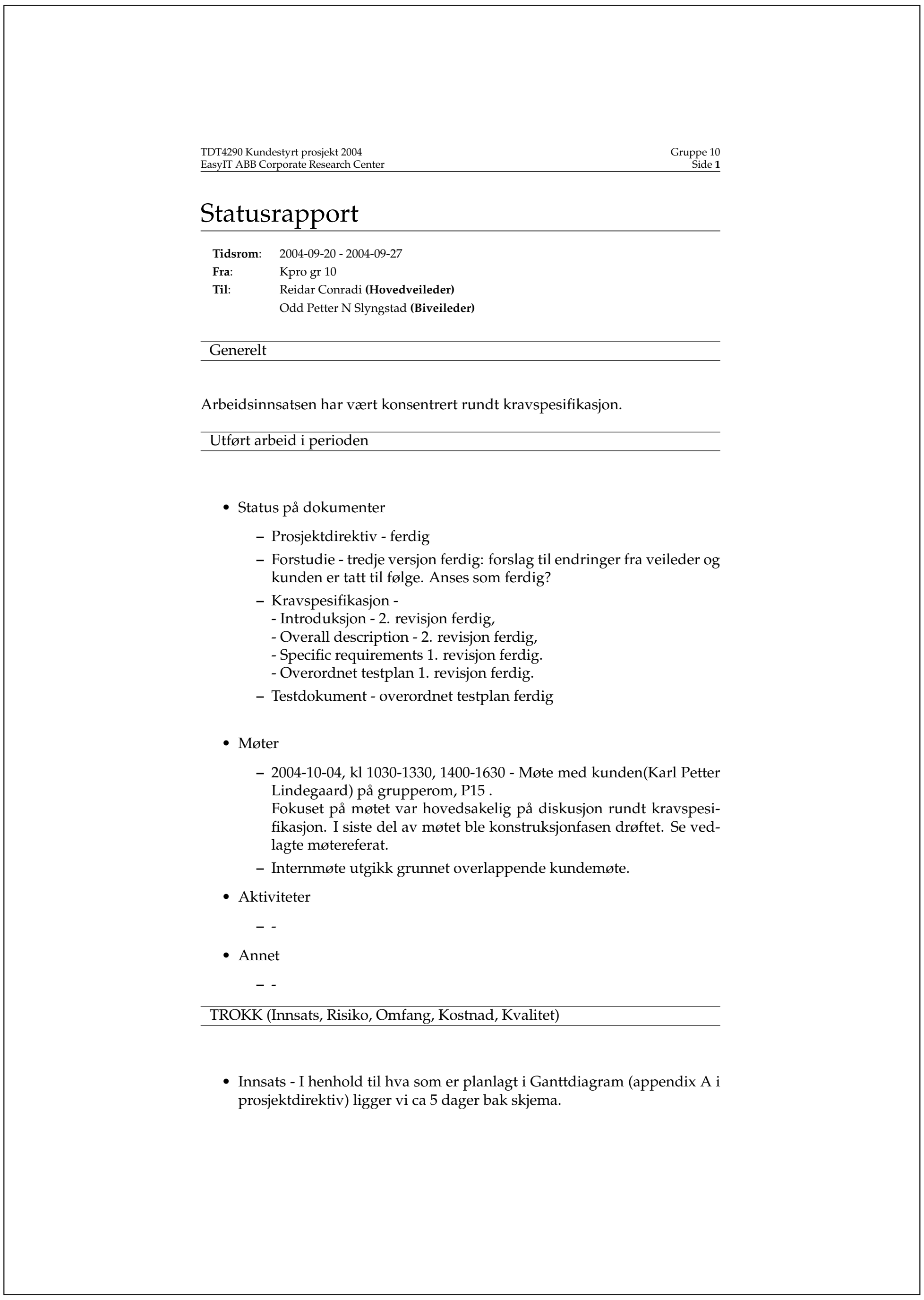
Problemer

-

Planlagt arbeid i neste periode

- Møter
  - Kundemøte mandag 2004-10-04 - i Trondheim
  - Internmøte mandag 2004-09-27
  - Veiledermøte onsdag 2004-09-29
- Aktiviteter
  - Kravspesifikasjon
  - Konstruksjon

Figure K.10: Status report from 2004-09-20 to 2004-09-27 - page 2



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **1**

Statusrapport

Tidsrom:	2004-09-20 - 2004-09-27
Fra:	Kpro gr 10
Til:	Reidar Conradi **(Hovedveileder)**
	Odd Petter N Slyngstad **(Biveileder)**

Generelt

Arbeidsinnsatsen har vært konsentrert rundt kravspesifikasjon.

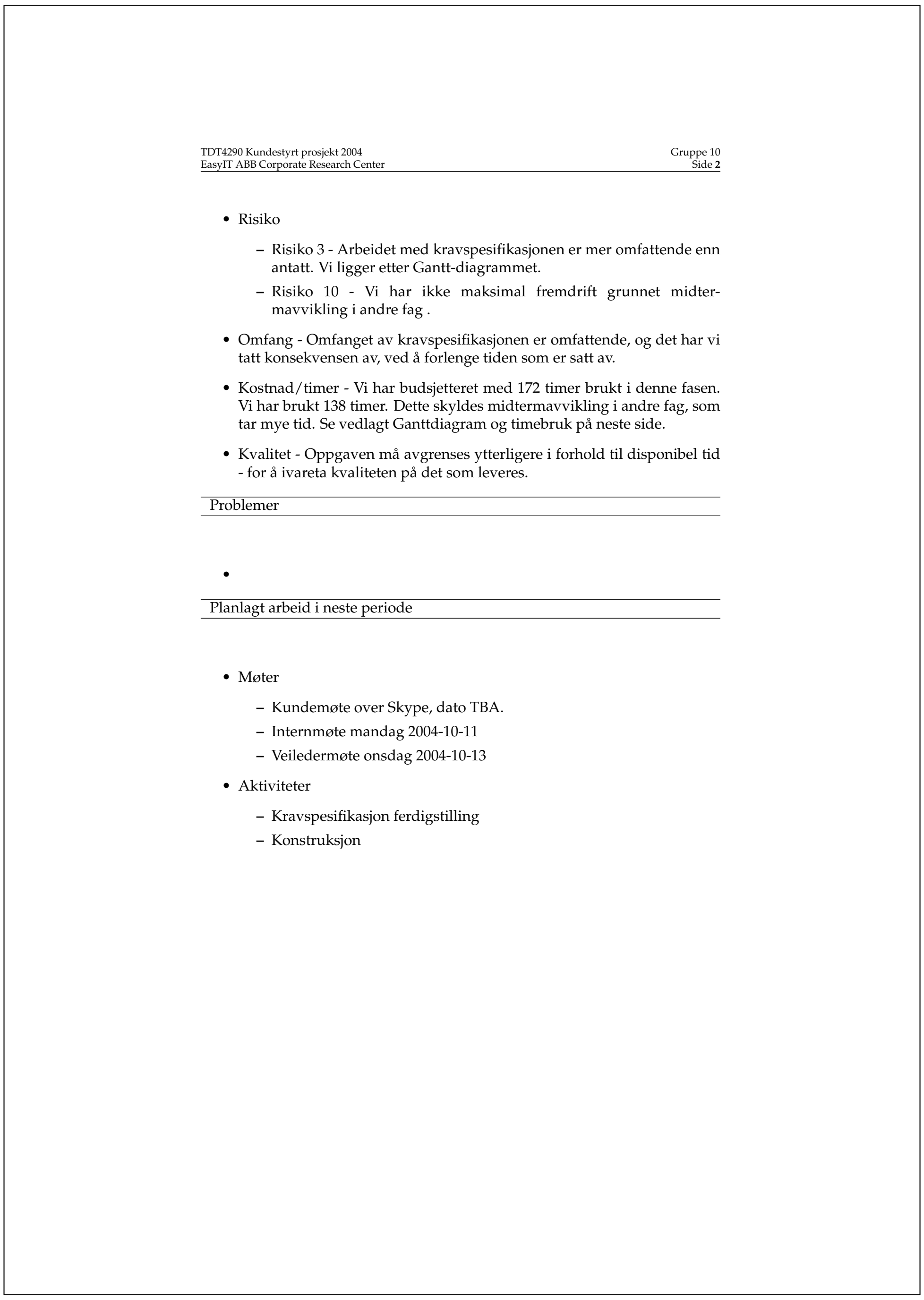
Utført arbeid i perioden

- Status på dokumenter
  - Prosjektdirektiv - ferdig
  - Forstudie - tredje versjon ferdig: forslag til endringer fra veileder og kunden er tatt til følge. Anses som ferdig?
  - Kravspesifikasjon -
    - Introduksjon - 2. revisjon ferdig,
    - Overall description - 2. revisjon ferdig,
    - Specific requirements 1. revisjon ferdig.
    - Overordnet testplan 1. revisjon ferdig.
  - Testdokument - overordnet testplan ferdig
- Møter
  - 2004-10-04, kl 1030-1330, 1400-1630 - Møte med kunden(Karl Petter Lindegaard) på grupperom, P15 .
    Fokuset på møtet var hovedsakelig på diskusjon rundt kravspesifikasjon. I siste del av møtet ble konstruksjonfasen drøftet. Se vedlagte møtereferat.
  - Internmøte utgikk grunnet overlappende kundemøte.
- Aktiviteter
  - -
- Annet
  - -

TROKK (Innsats, Risiko, Omfang, Kostnad, Kvalitet)

- Innsats - I henhold til hva som er planlagt i Ganttdiagram (appendix A i prosjektdirektiv) ligger vi ca 5 dager bak skjema.

Figure K.11: Status report from 2004-09-27 to 2004-10-04 - page 1



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **2**

- Risiko
  - Risiko 3 - Arbeidet med kravspesifikasjonen er mer omfattende enn antatt. Vi ligger etter Gantt-diagrammet.
  - Risiko 10 - Vi har ikke maksimal fremdrift grunnet midtermavvikling i andre fag .
- Omfang - Omfanget av kravspesifikasjonen er omfattende, og det har vi tatt konsekvensen av, ved å forlenge tiden som er satt av.
- Kostnad/timer - Vi har budsjetteret med 172 timer brukt i denne fasen. Vi har brukt 138 timer. Dette skyldes midtermavvikling i andre fag, som tar mye tid. Se vedlagt Ganttdiagram og timebruk på neste side.
- Kvalitet - Oppgaven må avgrenses ytterligere i forhold til disponibel tid - for å ivareta kvaliteten på det som leveres.

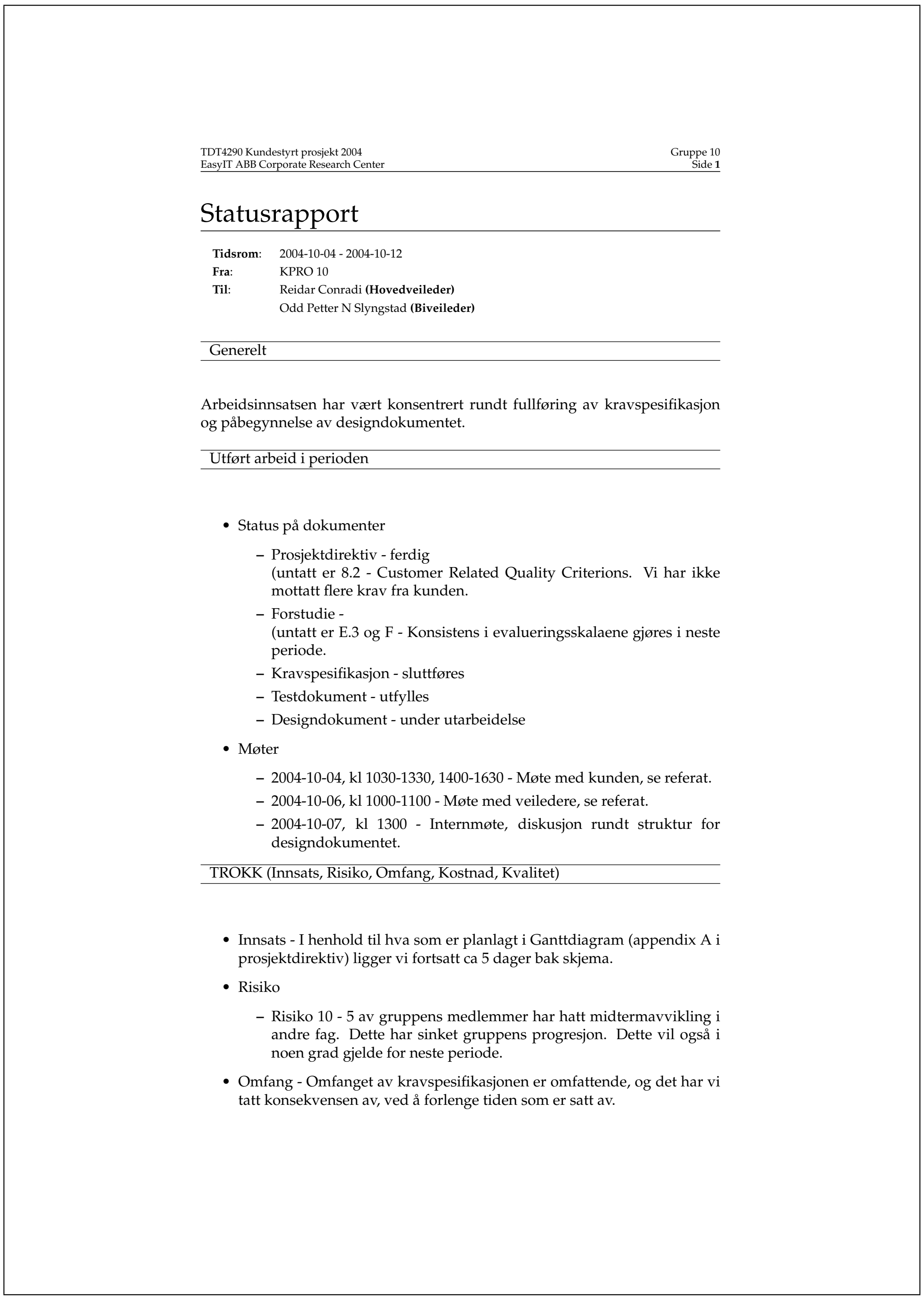
Problemer

-

Planlagt arbeid i neste periode

- Møter
  - Kundemøte over Skype, dato TBA.
  - Internmøte mandag 2004-10-11
  - Veiledermøte onsdag 2004-10-13
- Aktiviteter
  - Kravspesifikasjon ferdigstilling
  - Konstruksjon

Figure K.12: Status report from 2004-09-27 to2004-10-04 - page 2



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **1**

Statusrapport

Tidsrom:	2004-10-04 - 2004-10-12
Fra:	KPRO 10
Til:	Reidar Conradi **(Hovedveileder)**
	Odd Petter N Slyngstad **(Biveileder)**

Generelt

Arbeidsinnsatsen har vært konsentrert rundt fullføring av kravspesifikasjon og påbegynnelse av designdokumentet.

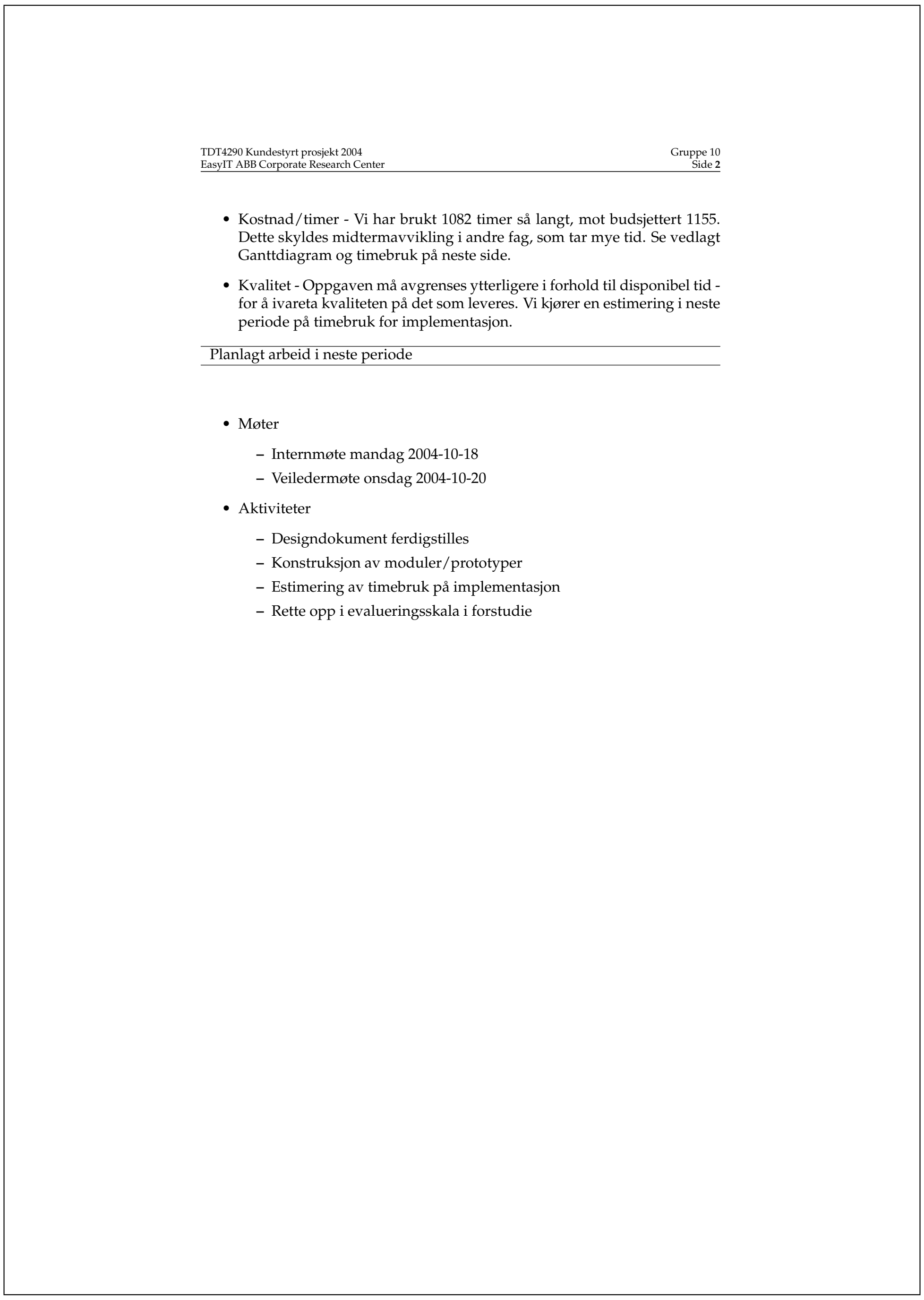
Utført arbeid i perioden

- Status på dokumenter
  - Prosjektdirektiv - ferdig
    (untatt er 8.2 - Customer Related Quality Criterions. Vi har ikke mottatt flere krav fra kunden.
  - Forstudie -
    (untatt er E.3 og F - Konsistens i evalueringsskalaene gjøres i neste periode.
  - Kravspesifikasjon - sluttføres
  - Testdokument - utfylles
  - Designdokument - under utarbeidelse
- Møter
  - 2004-10-04, kl 1030-1330, 1400-1630 - Møte med kunden, se referat.
  - 2004-10-06, kl 1000-1100 - Møte med veiledere, se referat.
  - 2004-10-07, kl 1300 - Internmøte, diskusjon rundt struktur for designdokumentet.

TROKK (Innsats, Risiko, Omfang, Kostnad, Kvalitet)

- Innsats - I henhold til hva som er planlagt i Ganttdiagram (appendix A i prosjektdirektiv) ligger vi fortsatt ca 5 dager bak skjema.
- Risiko
  - Risiko 10 - 5 av gruppens medlemmer har hatt midtermavvikling i andre fag. Dette har sinket gruppens progresjon. Dette vil også i noen grad gjelde for neste periode.
- Omfang - Omfanget av kravspesifikasjonen er omfattende, og det har vi tatt konsekvensen av, ved å forlenge tiden som er satt av.

Figure K.13: Status report from 2004-10-04 to 2004-10-12- page 1



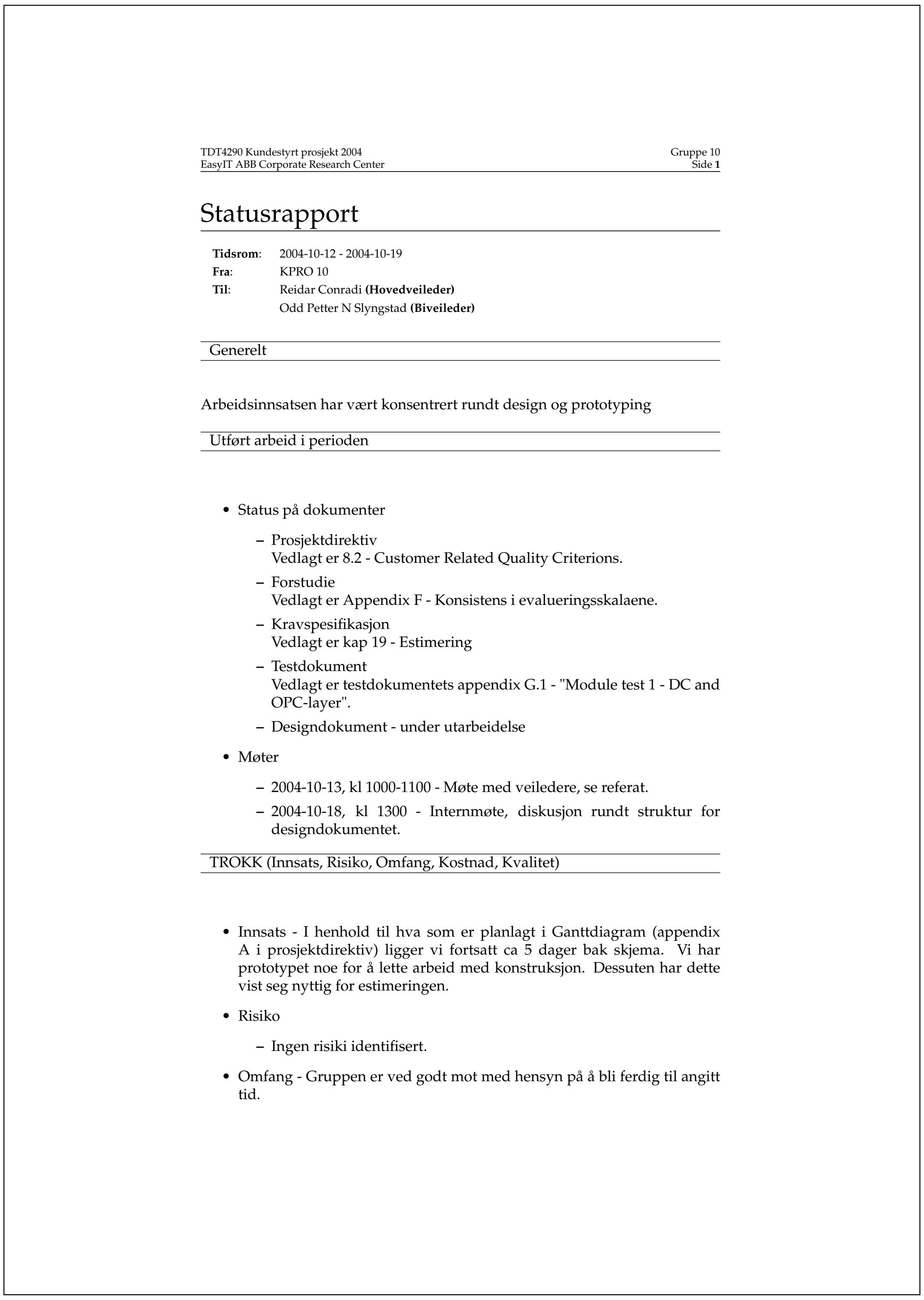
TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **2**

- Kostnad/timer - Vi har brukt 1082 timer så langt, mot budsjettert 1155. Dette skyldes midtermavvikling i andre fag, som tar mye tid. Se vedlagt Ganttdiagram og timebruk på neste side.
- Kvalitet - Oppgaven må avgrenses ytterligere i forhold til disponibel tid - for å ivareta kvaliteten på det som leveres. Vi kjører en estimering i neste periode på timebruk for implementasjon.

Planlagt arbeid i neste periode

- Møter
  - Internmøte mandag 2004-10-18
  - Veiledermøte onsdag 2004-10-20
- Aktiviteter
  - Designdokument ferdigstilles
  - Konstruksjon av moduler/prototyper
  - Estimering av timebruk på implementasjon
  - Rette opp i evalueringsskala i forstudie

Figure K.14: Status report from 2004-10-04 to 2004-10-12 - page 2



TDT4290 Kundestyrt prosjekt 2004
EasyIT ABB Corporate Research Center

Gruppe 10
Side **1**

Statusrapport

Tidsrom:	2004-10-12 - 2004-10-19
Fra:	KPRO 10
Til:	Reidar Conradi **(Hovedveileder)**
	Odd Petter N Slyngstad **(Biveileder)**

Generelt

Arbeidsinnsatsen har vært konsentrert rundt design og prototyping

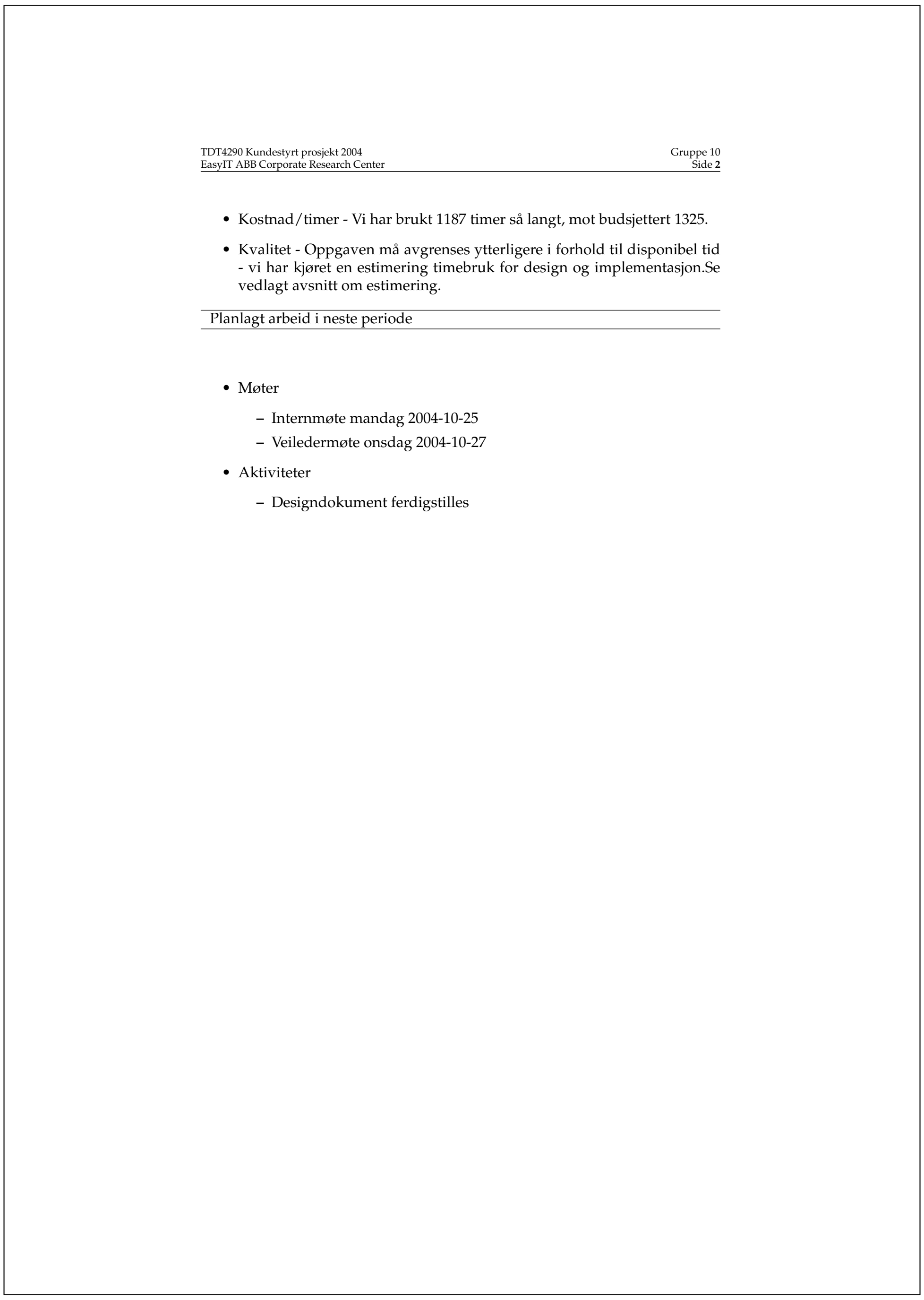
Utført arbeid i perioden

- Status på dokumenter
  - Prosjektdirektiv
    Vedlagt er 8.2 - Customer Related Quality Criterions.
  - Forstudie
    Vedlagt er Appendix F - Konsistens i evalueringsskalaene.
  - Kravspesifikasjon
    Vedlagt er kap 19 - Estimering
  - Testdokument
    Vedlagt er testdokumentets appendix G.1 - "Module test 1 - DC and OPC-layer".
  - Designdokument - under utarbeidelse
- Møter
  - 2004-10-13, kl 1000-1100 - Møte med veiledere, se referat.
  - 2004-10-18, kl 1300 - Internmøte, diskusjon rundt struktur for designdokumentet.

TROKK (Innsats, Risiko, Omfang, Kostnad, Kvalitet)

- Innsats - I henhold til hva som er planlagt i Ganttdiagram (appendix A i prosjektdirektiv) ligger vi fortsatt ca 5 dager bak skjema. Vi har prototypet noe for å lette arbeid med konstruksjon. Dessuten har dette vist seg nyttig for estimeringen.
- Risiko
  - Ingen risiki identifisert.
- Omfang - Gruppen er ved godt mot med hensyn på å bli ferdig til angitt tid.

Figure K.15: Status report from 2004-10-12 to 2004-10-19 - page 1

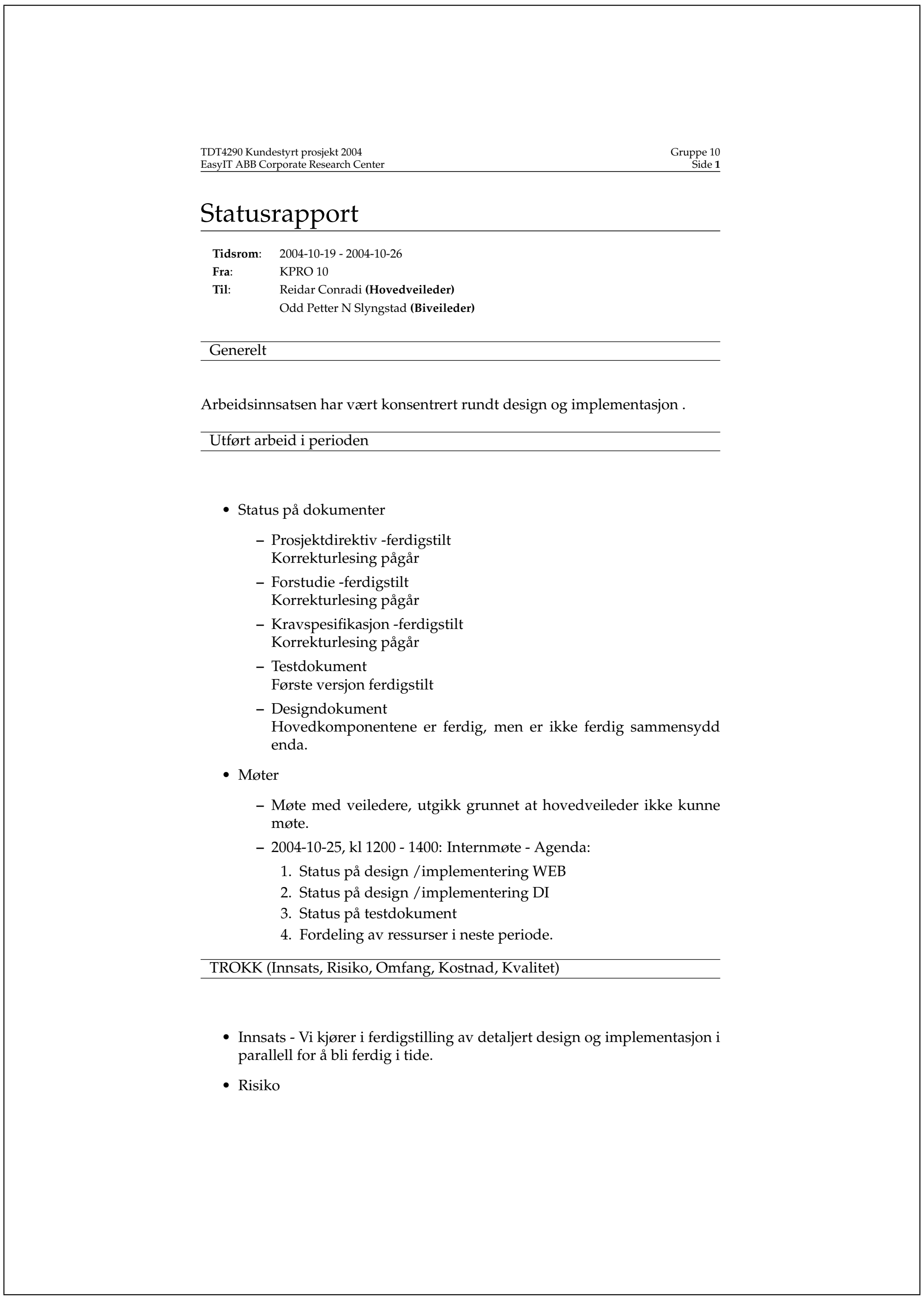


- Kostnad/timer - Vi har brukt 1187 timer så langt, mot budsjettert 1325.
- Kvalitet - Oppgaven må avgrenses ytterligere i forhold til disponibel tid - vi har kjøret en estimering timebruk for design og implementasjon.Se vedlagt avsnitt om estimering.

Planlagt arbeid i neste periode

- Møter
  - Internmøte mandag 2004-10-25
  - Veiledermøte onsdag 2004-10-27
- Aktiviteter
  - Designdokument ferdigstilles

Figure K.16: Status report from 2004-10-12 to 2004-10-19 - page 2



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **1**

Statusrapport

Tidsrom: 2004-10-19 - 2004-10-26
Fra: KPRO 10
Til: Reidar Conradi **(Hovedveileder)**
Odd Petter N Slyngstad **(Biveileder)**

Generelt

Arbeidsinnsatsen har vært konsentrert rundt design og implementasjon .

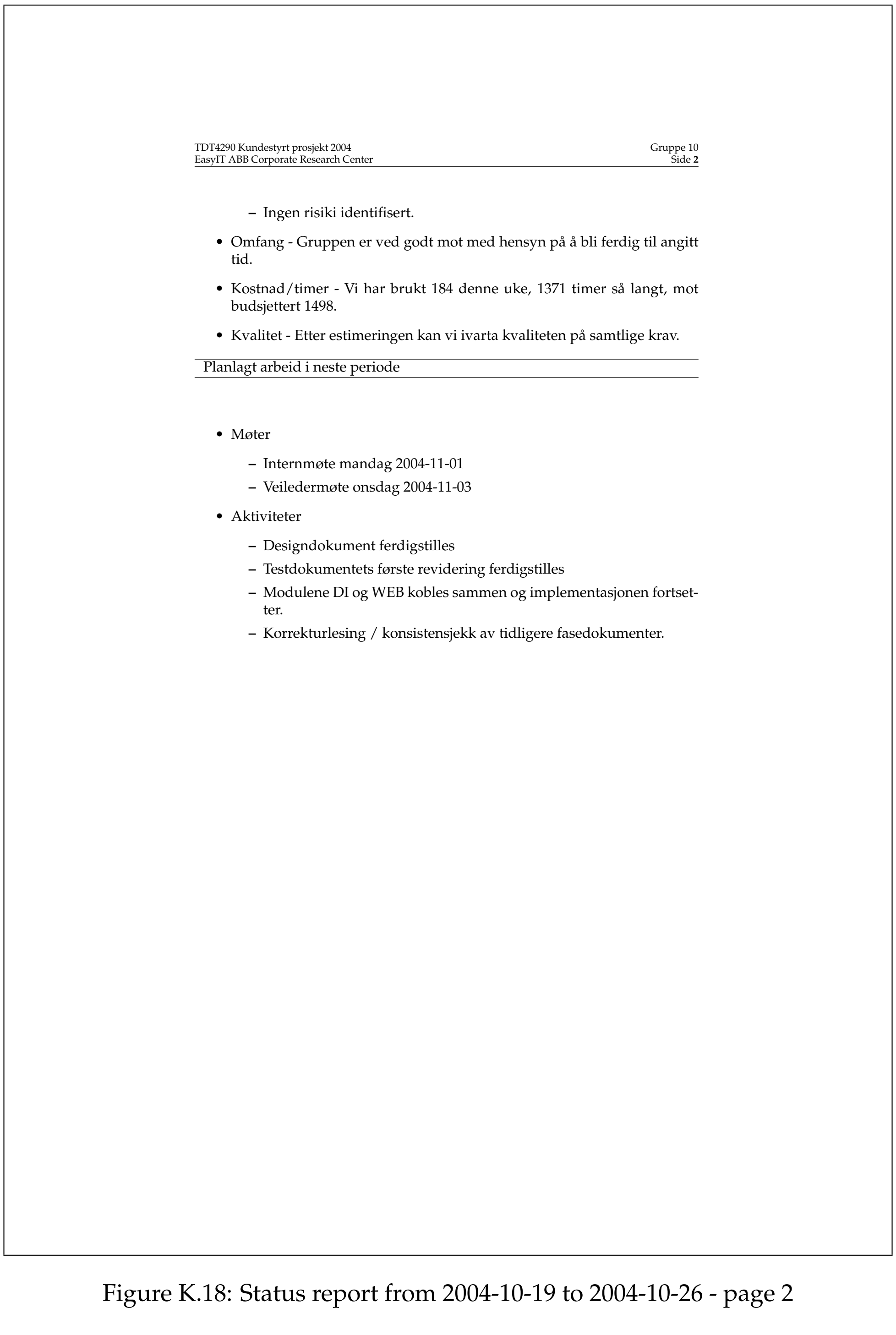
Utført arbeid i perioden

- Status på dokumenter
  - Prosjektdirektiv -ferdigstilt
    Korrekturlesing pågår
  - Forstudie -ferdigstilt
    Korrekturlesing pågår
  - Kravspesifikasjon -ferdigstilt
    Korrekturlesing pågår
  - Testdokument
    Første versjon ferdigstilt
  - Designdokument
    Hovedkomponentene er ferdig, men er ikke ferdig sammensydd enda.
- Møter
  - Møte med veiledere, utgikk grunnet at hovedveileder ikke kunne møte.
  - 2004-10-25, kl 1200 - 1400: Internmøte - Agenda:
    1. Status på design /implementering WEB
    2. Status på design /implementering DI
    3. Status på testdokument
    4. Fordeling av ressurser i neste periode.

TROKK (Innsats, Risiko, Omfang, Kostnad, Kvalitet)

- Innsats - Vi kjører i ferdigstilling av detaljert design og implementasjon i parallell for å bli ferdig i tide.
- Risiko

Figure K.17: Status report from 2004-10-19 to 2004-10-26 - page 1

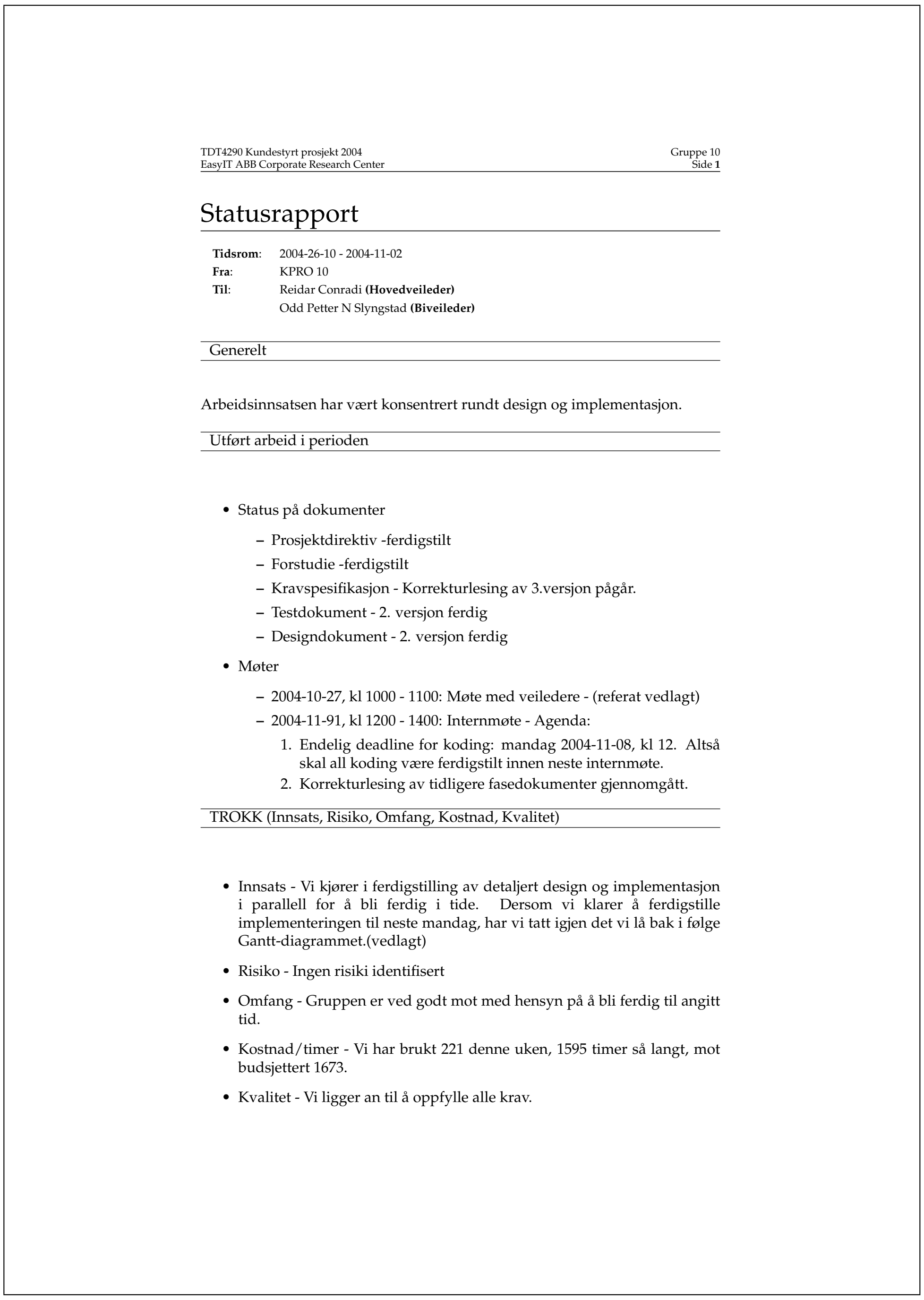


  - Ingen risiki identifisert.
- Omfang - Gruppen er ved godt mot med hensyn på å bli ferdig til angitt tid.
- Kostnad/timer - Vi har brukt 184 denne uke, 1371 timer så langt, mot budsjettert 1498.
- Kvalitet - Etter estimeringen kan vi ivarta kvaliteten på samtlige krav.

Planlagt arbeid i neste periode

- Møter
  - Internmøte mandag 2004-11-01
  - Veiledermøte onsdag 2004-11-03
- Aktiviteter
  - Designdokument ferdigstilles
  - Testdokumentets første revidering ferdigstilles
  - Modulene DI og WEB kobles sammen og implementasjonen fortsetter.
  - Korrekturlesing / konsistensjekk av tidligere fasedokumenter.

Figure K.18: Status report from 2004-10-19 to 2004-10-26 - page 2



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **1**

Statusrapport

Tidsrom:	2004-26-10 - 2004-11-02
Fra:	KPRO 10
Til:	Reidar Conradi **(Hovedveileder)**
	Odd Petter N Slyngstad **(Biveileder)**

Generelt

Arbeidsinnsatsen har vært konsentrert rundt design og implementasjon.

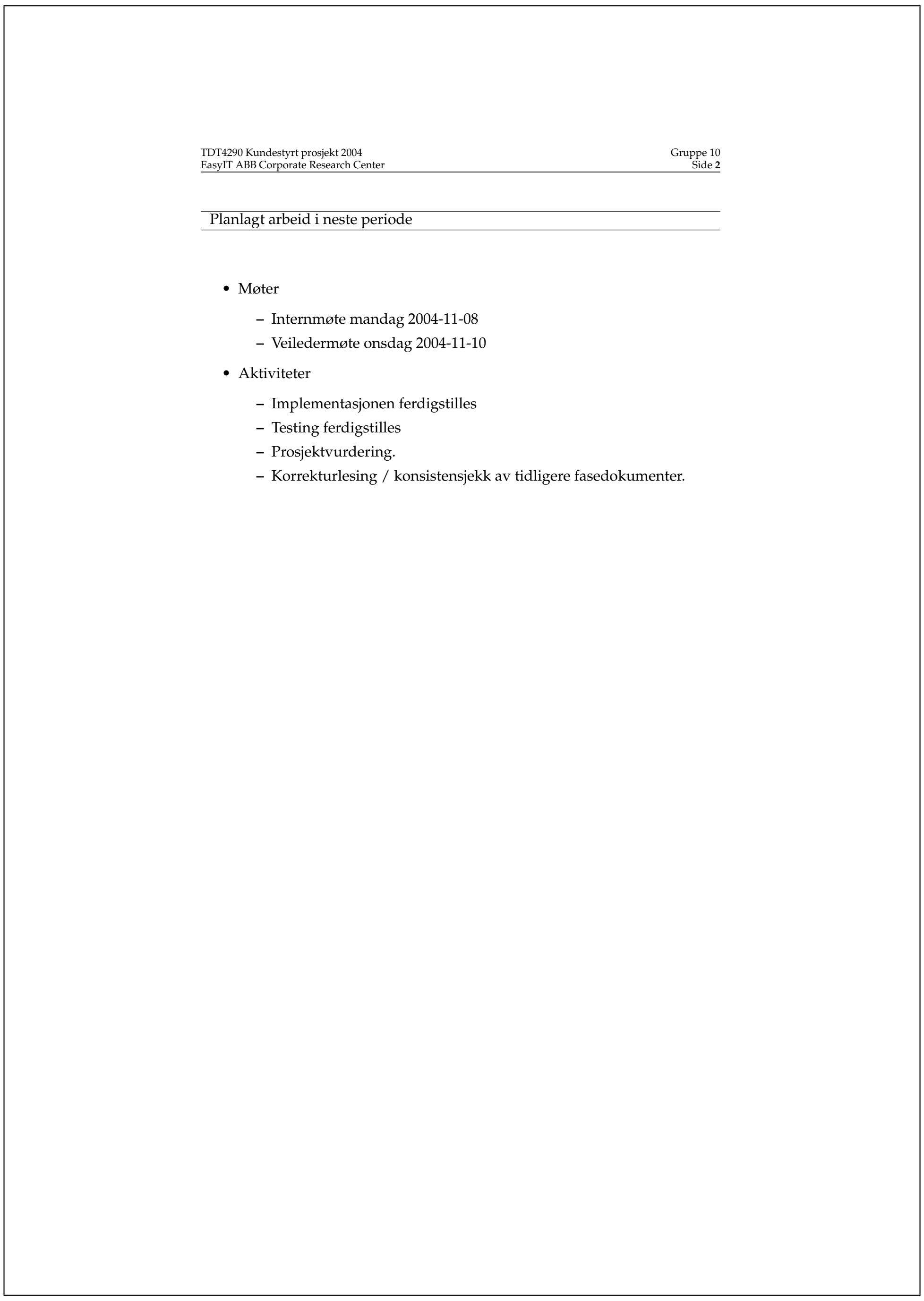
Utført arbeid i perioden

- Status på dokumenter
  - Prosjektdirektiv -ferdigstilt
  - Forstudie -ferdigstilt
  - Kravspesifikasjon - Korrekturlesing av 3.versjon pågår.
  - Testdokument - 2. versjon ferdig
  - Designdokument - 2. versjon ferdig
- Møter
  - 2004-10-27, kl 1000 - 1100: Møte med veiledere - (referat vedlagt)
  - 2004-11-91, kl 1200 - 1400: Internmøte - Agenda:
    1. Endelig deadline for koding: mandag 2004-11-08, kl 12. Altså skal all koding være ferdigstilt innen neste internmøte.
    2. Korrekturlesing av tidligere fasedokumenter gjennomgått.

TROKK (Innsats, Risiko, Omfang, Kostnad, Kvalitet)

- Innsats - Vi kjører i ferdigstilling av detaljert design og implementasjon i parallell for å bli ferdig i tide. Dersom vi klarer å ferdigstille implementeringen til neste mandag, har vi tatt igjen det vi lå bak i følge Gantt-diagrammet.(vedlagt)
- Risiko - Ingen risiki identifisert
- Omfang - Gruppen er ved godt mot med hensyn på å bli ferdig til angitt tid.
- Kostnad/timer - Vi har brukt 221 denne uken, 1595 timer så langt, mot budsjettert 1673.
- Kvalitet - Vi ligger an til å oppfylle alle krav.

Figure K.19: Status report from 2004-26-10 to 2004-11-02 - page 1

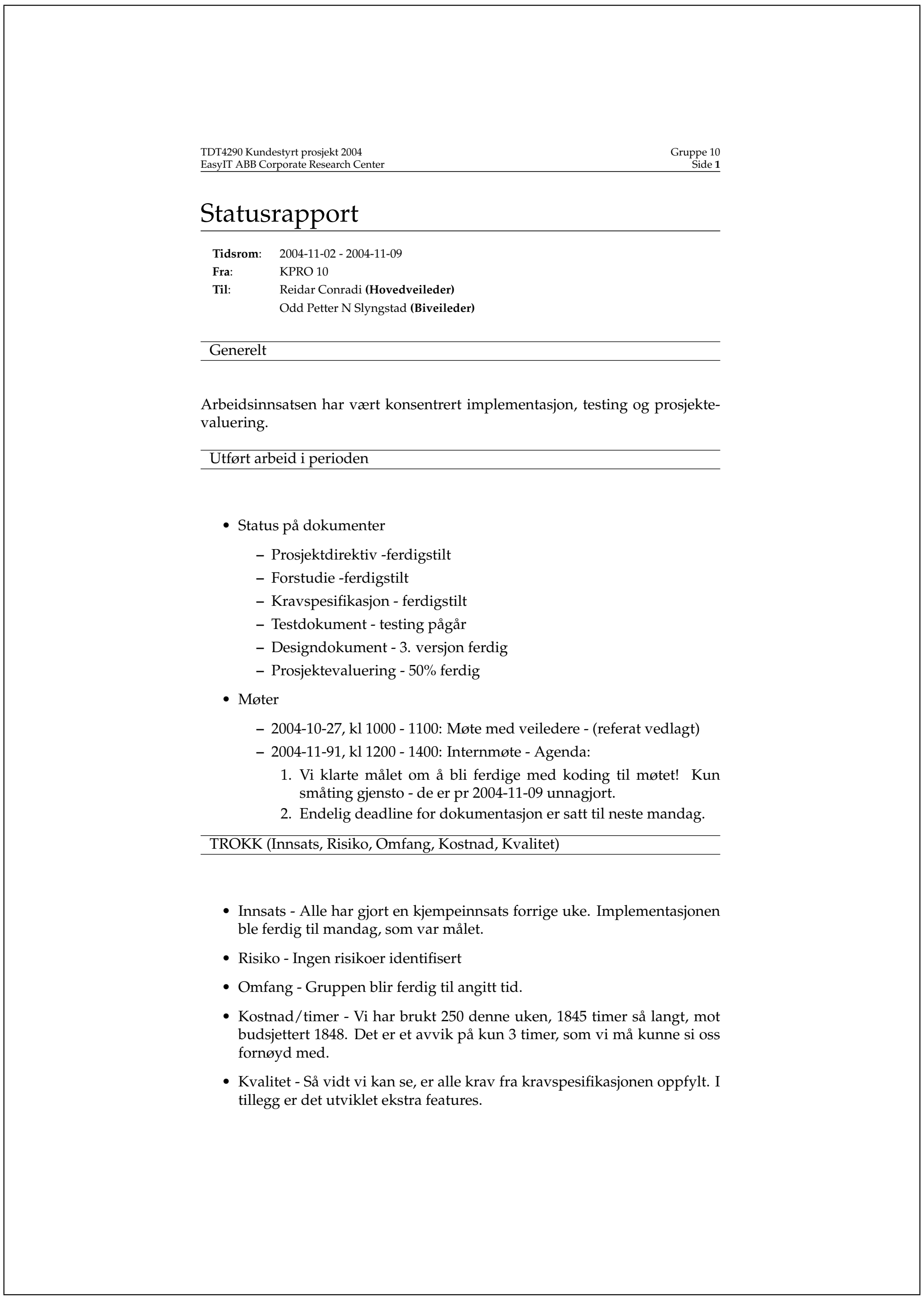


TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **2**

Planlagt arbeid i neste periode

- Møter
  - Internmøte mandag 2004-11-08
  - Veiledermøte onsdag 2004-11-10
- Aktiviteter
  - Implementasjonen ferdigstilles
  - Testing ferdigstilles
  - Prosjektvurdering.
  - Korrekturlesing / konsistensjekk av tidligere fasedokumenter.

Figure K.20: Status report from 2004-26-10 to 2004-11-02 - page 2



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side **1**

Statusrapport

Tidsrom:	2004-11-02 - 2004-11-09
Fra:	KPRO 10
Til:	Reidar Conradi **(Hovedveileder)**
	Odd Petter N Slyngstad **(Biveileder)**

Generelt

Arbeidsinnsatsen har vært konsentrert implementasjon, testing og prosjektevaluering.

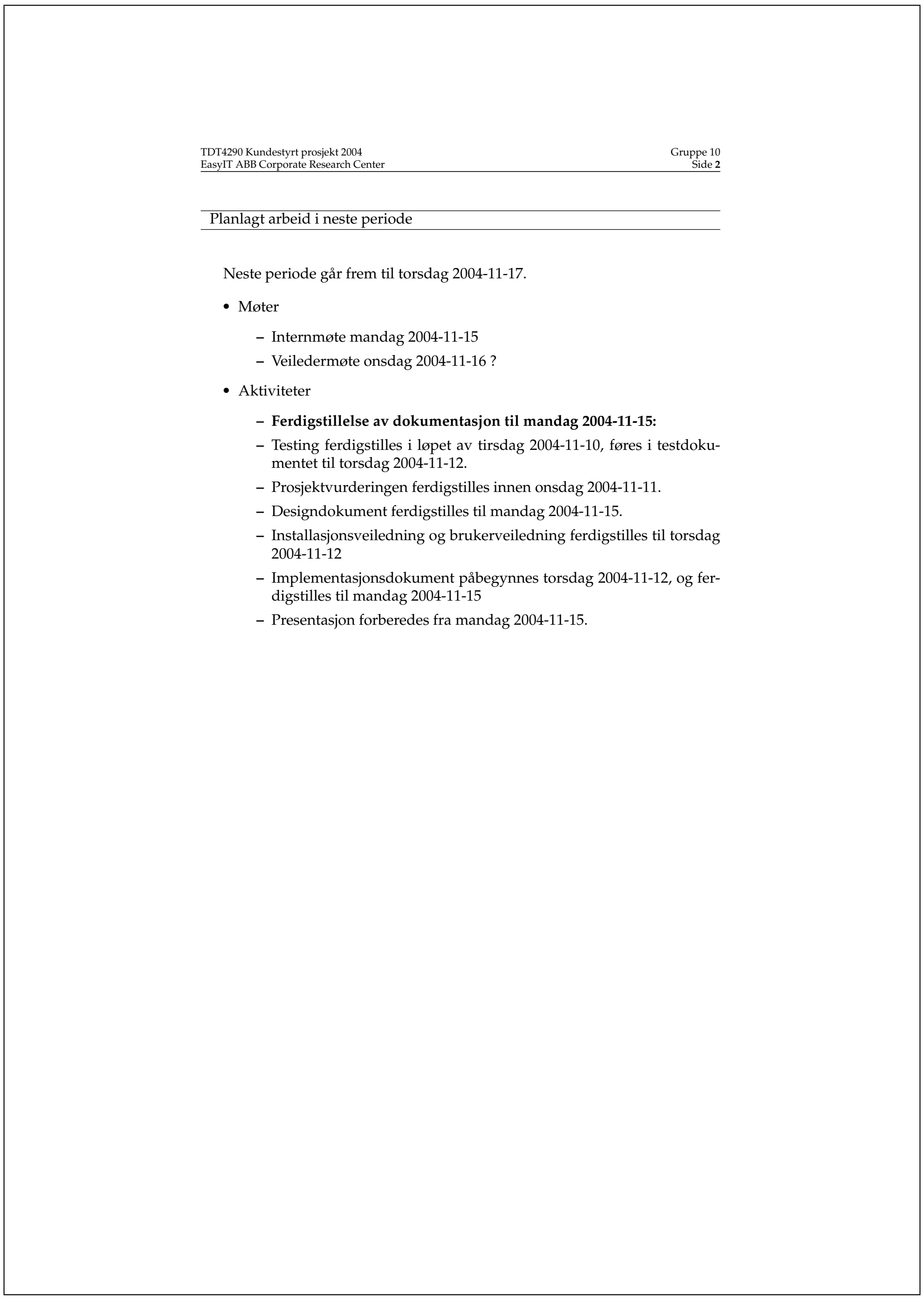
Utført arbeid i perioden

- Status på dokumenter
  - Prosjektdirektiv -ferdigstilt
  - Forstudie -ferdigstilt
  - Kravspesifikasjon - ferdigstilt
  - Testdokument - testing pågår
  - Designdokument - 3. versjon ferdig
  - Prosjektevaluering - 50% ferdig
- Møter
  - 2004-10-27, kl 1000 - 1100: Møte med veiledere - (referat vedlagt)
  - 2004-11-91, kl 1200 - 1400: Internmøte - Agenda:
    1. Vi klarte målet om å bli ferdige med koding til møtet! Kun småting gjensto - de er pr 2004-11-09 unnagjort.
    2. Endelig deadline for dokumentasjon er satt til neste mandag.

TROKK (Innsats, Risiko, Omfang, Kostnad, Kvalitet)

- Innsats - Alle har gjort en kjempeinnsats forrige uke. Implementasjonen ble ferdig til mandag, som var målet.
- Risiko - Ingen risikoer identifisert
- Omfang - Gruppen blir ferdig til angitt tid.
- Kostnad/timer - Vi har brukt 250 denne uken, 1845 timer så langt, mot budsjettert 1848. Det er et avvik på kun 3 timer, som vi må kunne si oss fornøyd med.
- Kvalitet - Så vidt vi kan se, er alle krav fra kravspesifikasjonen oppfylt. I tillegg er det utviklet ekstra features.

Figure K.21: Status report from 2004-11-02 to 2004-11-09 - page 1



TDT4290 Kundestyrt prosjekt 2004 Gruppe 10
EasyIT ABB Corporate Research Center Side 2

Planlagt arbeid i neste periode

Neste periode går frem til torsdag 2004-11-17.

- Møter
  - Internmøte mandag 2004-11-15
  - Veiledermøte onsdag 2004-11-16 ?
- Aktiviteter
  - **Ferdigstillelse av dokumentasjon til mandag 2004-11-15:**
  - Testing ferdigstilles i løpet av tirsdag 2004-11-10, føres i testdokumentet til torsdag 2004-11-12.
  - Prosjektvurderingen ferdigstilles innen onsdag 2004-11-11.
  - Designdokument ferdigstilles til mandag 2004-11-15.
  - Installasjonsveiledning og brukerveiledning ferdigstilles til torsdag 2004-11-12
  - Implementasjonsdokument påbegynnes torsdag 2004-11-12, og ferdigstilles til mandag 2004-11-15
  - Presentasjon forberedes fra mandag 2004-11-15.

Figure K.22: Status report from 2004-11-02 to 2004-11-09 - page 2

Part VIII

User guide for Easy IT

CHAPTER 42
INSTALLATION GUIDE

42.1 INSTALLATION GUIDE FOR EASYIT SERVER

Prerequisites:
- MS SQL Server is installed
- IIS web server is installed

1) Unzip the file EasyITServer.zip to where you want to install EasyIT.

2) In MS SQL Server, create a new database and add a new user to this database. The user must have the rights to create, drop, alter, select, update and delete tables in the new database.

3) Run the file EasyITDatabaseCreator.exe in the subfolder "Setup" of where EasyIT was installed.

4) Enter the following information about the database and user from step 2:

- Host to database server
- Name of database on the server
- Name of the user
- Password for the user.

Click "Create database table" and re-run EasyITDataBaseCreator.exe until you get no error messages. EasyIT will not function correctly otherwise.

5) Setup of EasyIT Server. Open the file EasyItStarter.exe.config in a text editor. Please fill in the following information:
easyITPort: This is the port EasyIT uses to listen for communication from applications. Default is port 6666, but this can be changed freely. The default should be fine for most use, but it will however need to be changed to be able to run multiple EasyIT servers on the same machine. This port MUST be the same port as the different applications use to connect to EasyIT, see 43.1.
applicationDirectory: This is the sub-folder in which EasyIT will try to locate applications for it to run. This sub-folder is not an absolute folder name, it is relative to the folder in with the EasyIT Server was installed. The default value of "apps" should work for everyone.
databaseHost: The hostname of the database server. This MUST be changed to the database server from step 2. Default value is completely unusable for everyone.
databaseName: The name of the database on the server given in databaseHost. This is the database EasyIT Server will use to log incoming data. Default value is completely unusuable for everyone.
databaseUser: The user that was created in step 2. Default value completely unusable, it must be changed.
databasePassword: The password for the user created in step 2. Default value is completely unusable, it must also be changed.

6) EasyIT Server is now ready for use. It can be started by running the EasyItStarter.exe file. Any problems will be reported in the log windows of EasyIT Server.

Next you should follow the 42.2 and if you want to add applications to the system; 43.1.

Application developers have the following methods available as an interface for communication with EasyIT Server

- public CSeries GetTags(DateTime dtStart, DateTime dtEnd, CTag[] tags, int gran)
- public CSet ReadTagsFromOPC(CTag[] tags)
- public bool WriteTagsToOPC(CTag[] tags)
- public void WriteTagsToDB(CTag[] tags)
- public long SubscribeSet(CSet setm)
- public void UnsubscribeSet(long setID)
- public CSet SpliceSets(CSet set1, CSet set2)
- public void SetConfigValue(int id, string name, string valueString)
- public string GetConfigValue(int id, string name)

For details about the use each of these methods, including description of the different parameters see the implementation document. A more advanced sample application than in figure 43.1 can be found in the in the Sample sub folder in the folder EasyIT Server was installed to.

42.2 Installation guide for EasyITWeb

1) Unzip EasyITWeb.zip to the wwwroot folder of IIS. This is usually c:/inetpub/wwwroot, but it might be placed in another location. The IIS-administrator should know this. EasyITWeb will be unpacked in a sub-folder called EasyITWeb, this is the EasyITWeb root folder.

2) Set file permissions: The web portal will ble placed under c:/inetpub/wwwroot/EasyITWeb by default. To be able to create custom pages, the web server needs permission to write files in the custom page directory. This is how to achieve it:

- Open Windows Explorer and go to the EasyITWeb root folder.
- Double click the 'modules' folder, and right click the 'Custom' folder
- Select 'Properties' and then click the 'Security' tab
- If no 'Security' tab appears follow these steps before continuing:
  - Select 'Folder options' from the 'Tools' menu bar in Windows Explorer
  - Select the 'View' tab and scroll to the bottom of the list
  - Uncheck the 'Use simple file file sharing' box and click ok
  - Now right click the 'Custom' folder once again, and the 'Security' tab should appear.
- Click the 'Add' button, and then the 'Advanced' button on the next screen
- Click 'Find now' in the Select users and groups dialog.
- Select the ASP.NET user account in the list, and press 'OK' twice
- Back in the Properties dialog for the Custom folder, select the ASP.NET account and check the 'Full control' box in the Permissions list. Then click OK.
- The EasyITWeb will now be able to write files to this folder

3) Configure server contact point: The address for connecting to the EasyIT server is configured in the EasyITWeb.config file in the EasyITWeb root folder. The connection uses a tcp connection, and you need to supply a hostname to the server and a port number.

4) Configure user accounts: A default administrator account is created with username 'admin' and password 'admin'. Once logged in, new accounts can be created through the 'Configure users' page available from the menu.

CHAPTER 43
DEVELOPING APPLICATIONS AND CUSTOM WEB PAGES

43.1 CONSTRUCTING APPLICATIONS

This is a guide to make applications that communicate with EasyIT.

43.1.1 Step by step

1. Open Visual Studio
2. Click File -> New -> Project
3. Select Visual C# Project and Console Application
4. Register a new application on the EasyIT web portal as described in 44.2
5. Add the following DLL files as .NET references to the new project: RemotingInterfaces.dll, ApplicationLibrart.dll, DataItems.dll. They are located in the "client-libraries" folder where you installed EasyIT.
6. Development can now start. See figure 43.1
7. Build the application
8. Copy the resulting exe file to the applications folder defined in "EasyItStarter.exe.config". Usually the applications folder is the sub-folder named "apps" in the folder that EasyIT was installed to.
9. The application is now ready for use in EasyIT. It can be started and stopped through the "Monitor Application" module on the web portal. For detailed description of this see 44.2

43.1.2 Explaination of the code

```
using EasyIt.Application;
using EasyIt.DataItems;
using EasyIt.RemoteInterfaces
```

We start by declaring that we want to use these three namespaces.

```
AppAPIClientBase clientBase = new AppAPIClientBase(25, 6666, "localhost");
```

This is the call that connects us to the EasyIt server. The first parameter is an unique identifier for this application. This identifier is generated when the application is registered on the EasyIT web portal in step 4. It is VERY IMPORTANT to register each new application on the web portal and use the id that the portal generates here!

```
namespace TestApp1
{
/// <summary>
/// Summary description for Class1.
/// </summary>
class Class1
{
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main(string[] args)
{
AppAPIClientBase clientBase = new AppAPIClientBase(25, 6666, "localhost");
clientBase.DataArrival +=new .DataReceivedEventHandler(clientBase_DataArrival);

CTag tag = new CTag("opcda://localhost/ICONICS.SimulatorOPCDA.2", "", "Numeric.Step");
CSet set = new CSet("", 3);
set.AddTag(tag);
clientBase.SubscribeSet(set);
while(true){
Thread.Sleep(6000);
}
}

private static void clientBase_DataArrival(CSet setm)
{
Console.WriteLine("got data from EasyIt");
}
}
 }
```

Figure 43.1: An example application

The second parameter is the port server and the third is the hostname of the machine the server is running on. These are defined by the EasyIT administrator and are located in the file "EasyIt-Starter.exe.config" in the folder that EasyIT was installed to. It is essential that these parameters are correct, or no contact to the EasyIT server can be made.

```
clientBase.DataArrival +=new .DataReceivedEventHandler(clientBase_DataArrival);
....
private static void clientBase_DataArrival(CSet setm)
{
Console.WriteLine("got data from EasyIt");
}
```

This registers an eventhandler on the DataArrival event. This event gets triggered everytime the EasyIT server has new data for the client. In this case we say that the method clientBase_DataArrival should be called everytime EasyIT server sends new data to us. clientBase_DataArrival just prints "got data from EasyIt" each time it is called but in a real application this method would inspect the incoming data, in setm, and do some kind of calculations on it.

```
CTag tag = new CTag("opcda://localhost/ICONICS.SimulatorOPCDA.2", "", "Numeric.Step")
```

This creates a new Tag object. The first parameter is the hostname of the opc server where the tag can be found, the second parameter is the path to the tag on server and the third parameter is the name of the tag on the server. These three parameters should match an actual tag on an actual OPC DA server. In this case we say we want to read the tag named "Numeric.Step" from the server "opcda://localhost/ICONICS.SimulatorOPCDA.2".

```
CSet set = new CSet("", 3); set.AddTag(tag);
```

We create a new set, give it rate of 3, and add our tag to it. A rate of 3 means we want callback with new values for each of the tags in the set every third second.

```
int subscriptionID = clientBase.SubscribeSet(set);
```

And finally we ask EasyIT to subscribe to the tag. From this call we get back a subscription id. This should be saved for later use, as this id must used if we want to unsubscribe from the set.

The end result is an application that gets a callback from the EasyIT server every third second with the newest value of the tags in the set is subscribed to.

43.2 Contructing custom web pages

43.2.1 Step by step

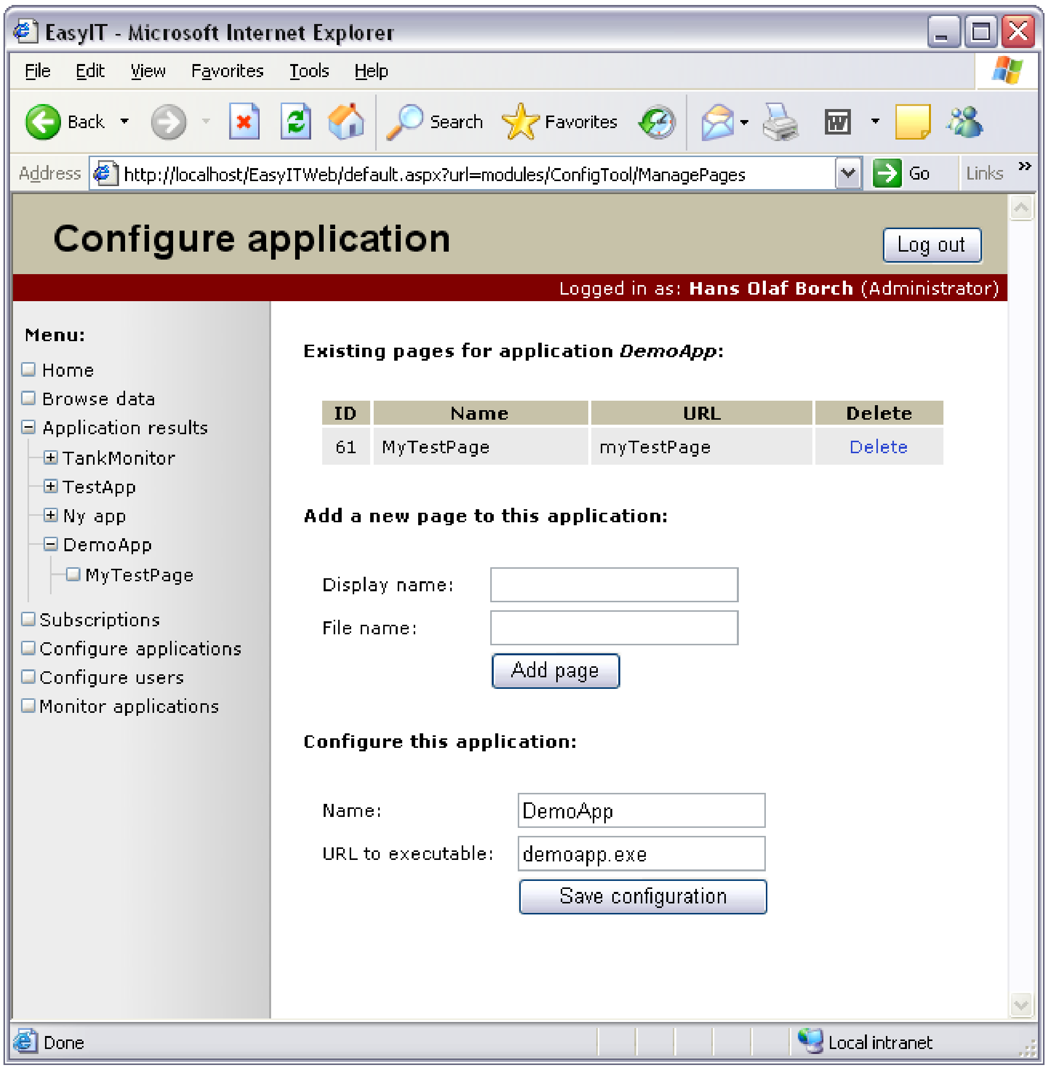
Follow these steps to create and customize a page for an existing application:

1. Choose 'Configure applications' from the menu, end click the 'Edit' button next to the application you want the page to be associated with.
2. Enter desired display name (used as page title) and file name (physical name on the disk) for the new page, and press 'Add page'. This creates empty 'shells' which can be edited in order to customize the page.
3. Navigate to the right application under 'Application results' in the menu, and select the page the you've just created. This pages contains information on where to find the file on the disk, for instance 'myPage.ascx'.
4. Using your favourite editor, open the 'myPage.ascx' and 'myPage.ascx.cs' files, and edit these.
5. Once you save your changes, the page is updated and compiled. There is no need to restart the EasyIT server.

The following code listings shows an example of a page for getting some data from the database, and displaying it as a table on the web page. The code is based on the empty shells automatically generated when adding the page. Lines that have been edited are marked with bold face letters. See listing 43.3 and 43.4

43.2.2 Explanation of the code

The myTestPage.ascx files declares an asp:DataGrid, which is a component which can be filled with a DataSet from the database in order to produce an html table. In the myTestPage.ascx.cs file, the datagrid is declared using the line



Configure application

Logged in as: **Hans Olaf Borch** (Administrator)

Existing pages for application *DemoApp*:

ID	Name	URL	Delete
61	MyTestPage	myTestPage	Delete

Add a new page to this application:

Display name:	
File name:	

Configure this application:

Name:	DemoApp
URL to executable:	demoapp.exe

Figure 43.2: Configuring the demo application

```
<%@ Control Language="c#" AutoEventWireup="false"
Codebehind="myTestPage.ascx.cs" Inherits="WEB.modules.Custom.App25.myTestPage"
TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>

<h1>Demo!</h1>
<p>Here are the 30 first datavalues requested logged by the demo
application:</p>
<p><asp:DataGrid id="DataGrid1" runat="server"></asp:DataGrid></p>
```

Figure 43.3: Example custom web page application: myTestPage.ascx

```
protected System.Web.UI.WebControls.DataGrid DataGrid1;
```

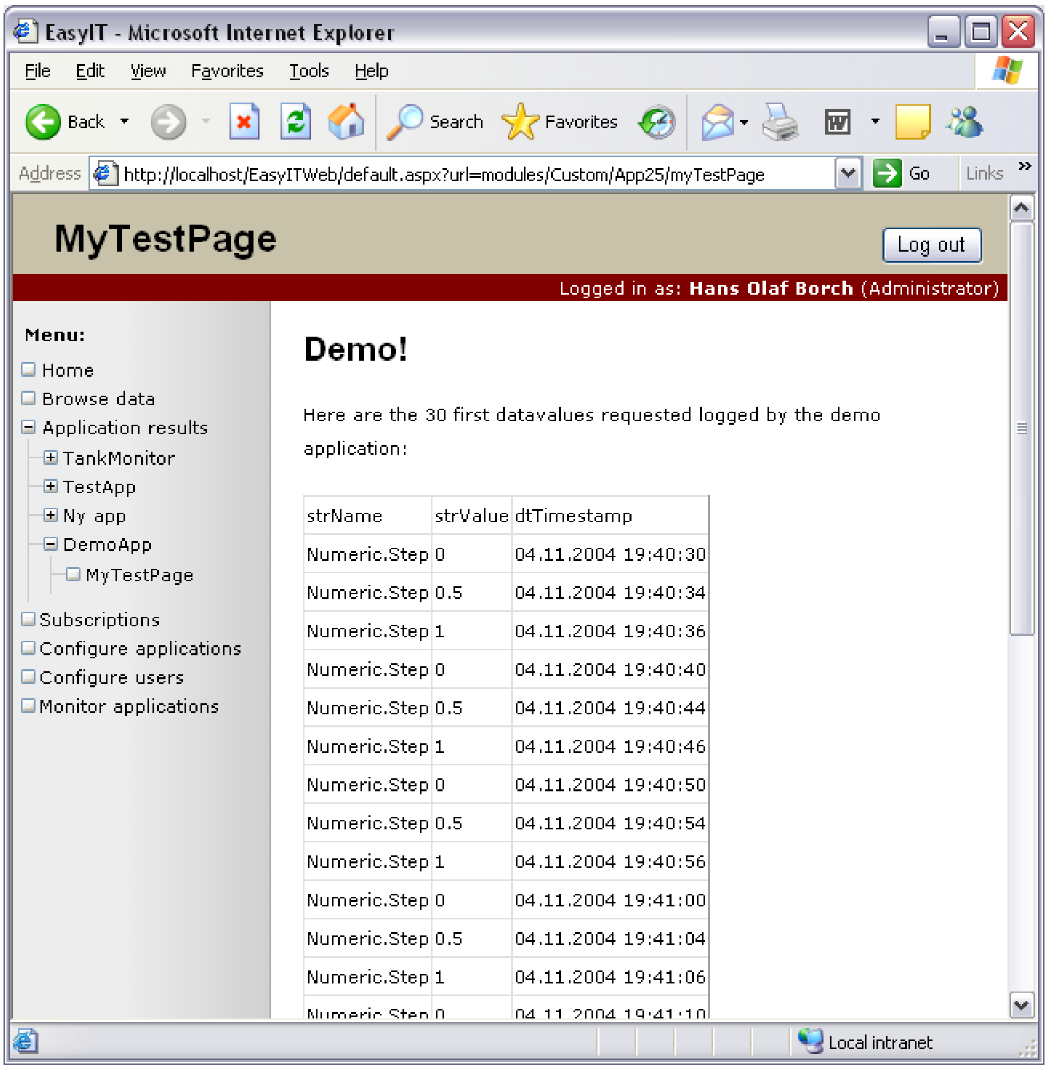
Three lines are added to the Page_Load method, which is run each time the page is loaded in the browser.

```
string query = ”SELECT top 30 strName, strValue, dtTimestamp FROM tblTag
WHERE strServer = ’opcda://localhost/ICONICS.SimulatorOPCDA.2’
AND strPath = ’ ’ AND strName = ’Numeric.Step’ ”;
DataGrid1.DataSource = webapi.ExecuteSelectQuery(query);
DataGrid1.DataBind();
```

The first line defines a query asking for the first 30 rows logged by the server of the tag subscribed to by testapp1 (shown in the previous section). The second line executes the query by calling the ExecuteSelectQuery in the WebAPI. The ExecuteSelectQuery method returns a DataSet, which is set as the DataSource for the DataGrid component. The component is then filled with the data by calling the DataBind() method. The resulting page is shown in figure 43.5.

```
namespace WEB.modules.Custom.App25
{
  using System;
  using System.Data;
  using System.Web;

  public class myTestPage : EasyITControl
  {


 protected System.Web.UI.WebControls.DataGrid DataGrid1;


    public myTestPage()
    {
      PageTitle = "MyTestPage";
    }

    private void Page_Load(object sender, System.EventArgs e)
    {
      WebAPI webapi = (WebAPI)Application["webapi"];


 string query = "Select top 30 strName, strValue, dtTimestamp from
tblTag where strServer = ’opcda://localhost/ICONICS.SimulatorOPCDA.2’
and strPath = ” and strName = ’Numeric.Step’"; DataGrid1.DataSource =
webapi.ExecuteSelectQuery(query);
DataGrid1.DataBind();

    }

    #region Web Form Designer generated code
    override protected void OnInit(EventArgs e)
    {
      InitializeComponent();
      base.OnInit(e);
    }

    private void InitializeComponent()
    {
      this.Load += new System.EventHandler(this.Page_Load);
    }
    #endregion
  }
}
```

Figure 43.4: Example custom web page application: myTestPage.ascx.cs



Demo!

Here are the 30 first datavalues requested logged by the demo application:

strName	strValue	dtTimestamp
Numeric.Step	0	04.11.2004 19:40:30
Numeric.Step	0.5	04.11.2004 19:40:34
Numeric.Step	1	04.11.2004 19:40:36
Numeric.Step	0	04.11.2004 19:40:40
Numeric.Step	0.5	04.11.2004 19:40:44
Numeric.Step	1	04.11.2004 19:40:46
Numeric.Step	0	04.11.2004 19:40:50
Numeric.Step	0.5	04.11.2004 19:40:54
Numeric.Step	1	04.11.2004 19:40:56
Numeric.Step	0	04.11.2004 19:41:00
Numeric.Step	0.5	04.11.2004 19:41:04
Numeric.Step	1	04.11.2004 19:41:06
Numeric.Step	0	04.11.2004 19:41:10

Figure 43.5: The resulting demo page

CHAPTER 44
USING THE EASYITWEB

There are two kinds of users of the EasyITWeb: Analysts and Administrators. Analysts have access to the ’Browse data’ and ’Application results’. In addition to these, Administrators have access to ’Subscriptions’, ’Configure applications’, ’Configure users’ and ’Monitor applications’. These are all described below, in the order in which they appear in the menu.

44.1 BASIC FUNCTIONALITY

Browse data
This page allows you to select which tags you want to browse, and set the granularity and time span for the data. Tags are selected by first choosing a server in the server list. This list is automatically populated, depending on which servers the tags available at the server are logged from. This includes both applications (represented by the application name) and OPC Servers (represented by their URL). When a server is chosen, the available tags appear in the tags list box. By selecting them one by one and pressing the ’Add tag’ button, these are added to the list of selected tags. The time span for the data plot is set in the To/From boxes. If a tag is selected, the ’Copy timespan’ button is used for setting the time span to the exact time in which the selected tag is logged. Granularity is the number of samples to be displayed when browsing data. To view the selected tags press the ’View data’ button.

The data is now displayed as a time plot by default. You can change the chart type, time span and other properties. If you want to view real-time data that is being logged at the moment, set the desired time frame and choose ’Autorefresh’.

Application results
Displays a list of applications. Click the name of the application to show which pages are available. Click the name of the page you want to view.

44.2 ADMINISTRATOR SPECIFIC FUNCTIONALITY

Subscriptions
This page displays all active subscriptions, and allows you to edit these. Web subscriptions are needed when a user wants to browse data from an opc-server without having to write a separate application just for subscribing to the data. The first page allows you to add, edit or delete subscriptions. The ’delete’ button stops the logging of the data on the DI. Add/edit lets you edit the tags to be contained in the set, and set the granularity and deadband for the set. If you save a set with no tags, the subscription is automatically removed.

Configure applications
Displays a list of all applications containing the following information: ID of the application (used for recognizing applications that connect to the DI). Name of the application. The number of custom pages associated with the application (nofPages). URL to the executable file for the application.

To add a new application simply enter its name and URL to executable file (if created) in the ’Add new application’ form. To delete an application, press the ’Delete’ button next to the application. To edit existing configuration of an application, press the ’Edit’ button next to the application. This presents you with a new page showing the custom pages associated with the chosen application. This page allows you to add or delete pages, and to configure the application itself. To configure the application, change Name and URL to desired values and press 'Save configuration'.

To add new page, enter its display name and name of the file to be written to disk. When 'Add page' is clicked an empty shell of an asp.net page is outputted to %webroot%/EasyITWeb/modules/Custom/AppX. The folder is created automatically from the ID of the application, so application 15 would have its files under the App15 directory. Pages can be deleted from the database with the 'Delete' button next to each page. This deletes the database entry, but does not delete the actual files on disk.

Configure users
A list of currently registered users is displayed. Press 'Delete' next to the user if you want to delete the account from the database. 'Edit' og 'New user' brings you to the page for editing user account details. A unique username must be supplied (this is not possible to change after creating the user account).

Monitor applications
Gives a list of applications and their status. Allows an administrator to start and stop these applications directly through the 'Start' or 'Stop' button next to each application. This only works provided that the exe files are placed under the EasyITStarter/apps/ directory and that the name of the exe file is stored in the application configuration.