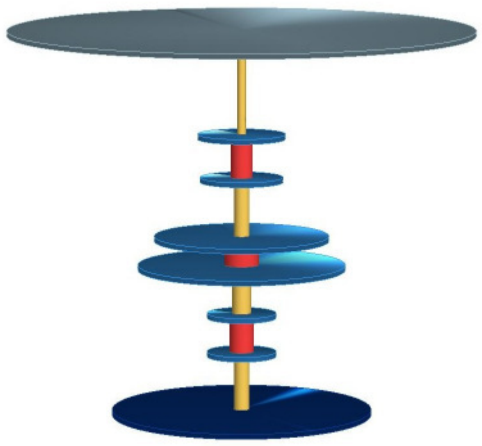
On-line Simulations –

Advanced Web-enabled services

Inger Bolin[1], Maria Borg[2]
ABB Process Industries, PI/FVT, 721 59 Västerås, Sweden
[1]ibn98008@student.mdh.se
[2]mbg98007@student.mdh.se



Department of Computer Engineering
Mälardalen University

Västerås, Mars 2002

Supervisor and Examiner at university: Johan Stärner
Supervisor at ABB Process Industries: Mats Tallfors

Summary

ABB Process Industries has developed a simulation tool called Drive Dynamics Analyzer (DDA) to be able to analyse dynamics features in rotating machines. DDA is written in Matlab and can be compiled and run as an ordinary application principally, but a few simulations are only working when DDA runs directly under Matlab.
The main goal of this thesis is to set up an on-line communication between a future client/customer, which is going to have a compiled DDA application, and ABB, which does the advanced simulations under Matlab. For this purpose Matlab Web Server (MWS) is acquired. The MWS makes it possible to deploy Matlab based applications over a network using standard web technology.

The thesis also includes installation, configuration and learning the new technology involved for developing with the MWS.

The result is a MWS application implemented in the client DDA, enabling advanced services over the web without having Matlab installed. A file is transmitted from the client through the MWS to Matlab for calculation and a file holding the result is sent back and the simulation is plotted in the client DDA.

Preface - Acknowledgement

This degree thesis was done at ABB Process Industries, department FVT, during the period December 2001 to Mars 2002, as a part of the education Bachelor of Science in Computer Science at Mälardalen University in Västerås, Sweden. Inger Bolin and Maria Borg performed the work.

We sincerely thank our supervisors Mats Tallfors at ABB Process Industries and Johan Stärner at Mälardalen University.

For technical support we especially want to thank Mattias Olsson at ABB Process Industries for his support and help with Matlab and the DDA application, Ulf Arvehed at Comsol Support for his ideas and feedback on our work.

We want to thank Mattias Nordin for giving us the opportunity to do this thesis and we will never forget all the nice coffee breaks with the other staff members.

TABLE OF CONTENT

SUMMARY 2
PREFACE - ACKNOWLEDGEMENT 3

1 INTRODUCTION 5

1.1 BACKGROUND 5
1.2 PURPOSE 6
1.3 DEFINITIONS 6

2 RELEVANT TECHNOLOGIES 6

2.1 MATLAB WEB SERVER (MWS) 7
2.1.1 Building MWS Applications 7
2.1.2 Product Requirements 8
2.1.3 MWS Components 8
2.1.4 Important MWS functions 10
2.1.5 Understanding MWS 10
2.2 APACHE SERVER 11
2.3 PHP-SCRIPT 11
2.4 CGI 12
2.5 TCP/IP 12

3 PROBLEM DESCRIPTION AND ANALYSIS 12

3.1 MODEL/METHOD 13

4 SOLUTION 14

4.1 INSTALLING, CONFIGURING AND STUDYING MWS 14
4.2 MAKING A TEST PROGRAM 14
4.2.1 File uploading to a HTML page 14
4.2.2 Establish communication with MWS 15
4.2.3 Get a result file back to the client 15
4.3 IMPLEMENTATION IN THE DDA APPLICATION 16
4.4 USABILITY 16
4.4.1 Input and Output HTML documents 16
4.4.2 DDA application 17
4.5 TEST 18
4.5.1 Usability Evaluation 18

5 RESULT 20

5.1 RESULT ANALYSIS 21

6 RECOMMENDATION, FUTURE WORK 22

7 CONCLUSIONS 23

8 REFERENCES 24

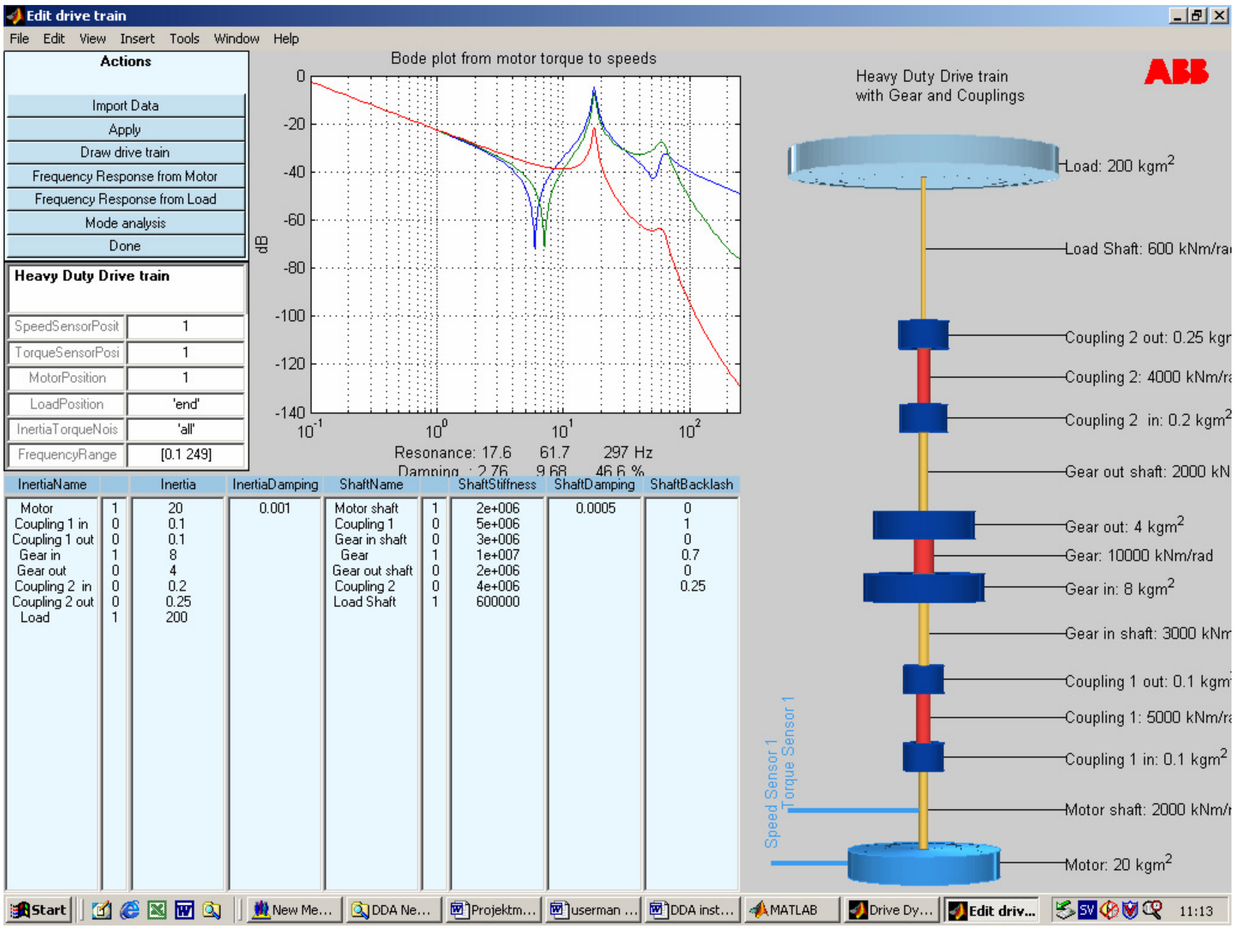
9 APPENDIX 24

1 Introduction

This chapter gives a short introduction to the background and purpose with this thesis and also the abbreviations used in this report.

1.1 Background

ABB Process Industries has developed a simulation tool called Drive Dynamics Analyzer (DDA) to be able to analyse dynamics features in rotating machines. An example of how to use this tool is building a model for a rolling mill drive. The model contains information about the mechanical and electric features of the engine together with stiffness and moment of inertia in shafts, couplings, gearboxes, and in the roll. The model is used for studying wear and tear during drive. It is also utilised to decide regulator parameters for speed control of the equipment.



Heavy Duty Drive train

SpeedSensorPosit	1
TorqueSensorPosi	1
MotorPosition	1
LoadPosition	'end'
InertiaTorqueNois	'all'
FrequencyRange	[0.1 249]

Resonance: 17.6 61.7 297 Hz
Damping : 2.76 9.68 46.6 %

InertiaName		Inertia	InertiaDamping	ShaftName		ShaftStiffness	ShaftDamping	ShaftBacklash
Motor	1	20	0.001	Motor shaft	1	2e+006	0.0005	0
Coupling 1 in	0	0.1		Coupling 1	0	5e+006		1
Coupling 1 out	0	0.1		Gear in shaft	0	3e+006		0
Gear in	1	8		Gear	1	1e+007		0.7
Gear out	0	4		Gear out shaft	0	2e+006		0
Coupling 2 in	0	0.2		Coupling 2	0	4e+006		0.25
Coupling 2 out	0	0.25		Load Shaft	1	600000		
Load	1	200						

Figure 1. Data sheet for drive line and process performance analysis for rolling mill drives.

Figure 1 describes the data needed for analysis of driveline and process performance analysis for rolling mill drives. Added to this a form should be filled in with all relevant data. Some data is optional (e.g. backlash sizes), but gives important input for a thorough analysis.

With the right input data, the properties of the driveline can be optimized before final dimensioning and design. Also the benefits of the special rolling mill control functions can be presented to the customer. Simulations and frequency responses of the driveline can also easily be given.

1.2 Purpose

The main goal of this thesis is to create on-line data communication for simulations, using Matlab Web Server (MWS), between ABB and their customers. Engineers and sellers at ABB can also take advantage of this application. An important part is also the usability of the customer interface to minimize error and effort for the client.

DDA is written in Matlab. DDA can be compiled and run as an ordinary application principally, but a few simulations are only working when DDA runs directly under Matlab.

At the moment the customer has to write down all parameters needed for these simulations in paper form. ABB receives this information and inserts the parameters in the DDA-application. The generated result report is returned to the customer by mail. With this procedure one runs the risk of getting wrong parameters and it is also very time consuming. Therefore ABB wants to computerize the communication with their customers. The idea is that the customer is going to have a compiled version of the DDA application and through on-line data communication with ABB being able to do advanced web-enabled services.

The development environment, MWS, chosen for this project is new to ABB SEPID, meaning that a part of this thesis is to handle the installation and configuration of MWS.

1.3 Definitions

Abbreviation	**Definition**
CGI	Common Gateway Interface
DDA	Drive Dynamics Analyzer
HTML	Hypertext Markup Language
HTTPD	Hypertext Transfer Protocol Daemon
MWS	Matlab Web Server
PHP	Hypertext Preprocessor
TCP/IP	Transmission Control Protocol/Internet Protocol
WWW	World Wide Web

Components are written with the font Verdana to make it clearer….

2 Relevant technologies

In order to assimilate this report a brief description of important technologies is given.

2.1 Matlab Web Server (MWS)

The MWS enables the creation of Matlab applications that use the capabilities of the WWW to send data to Matlab for computation and to display the results in a web browser. The MWS depends upon TCP/IP networking for transmission of data between the client system and Matlab. The required networking software and hardware must be installed on a system prior to use the MWS. The configuration can be done in different ways, see figure 2 and 3.

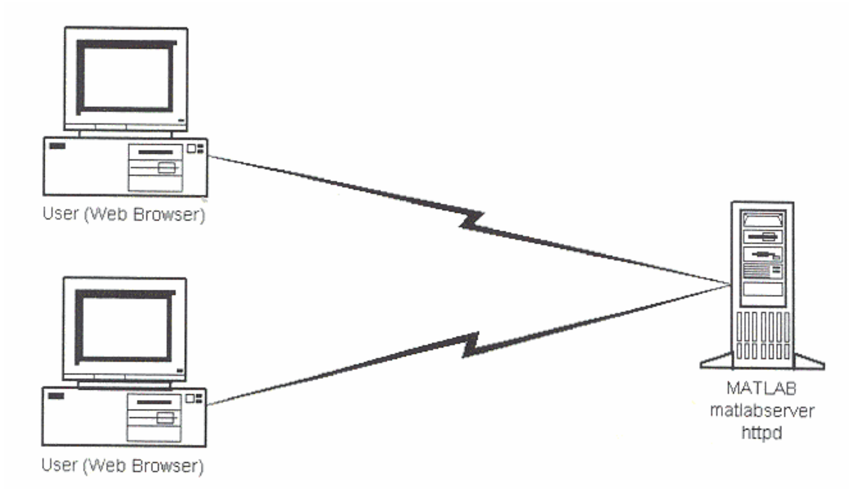


Figure 2. In the simplest configuration a web browser runs on a client workstation, while Matlab, the MWS, and the web server daemon (HTTPD) run on another machine.

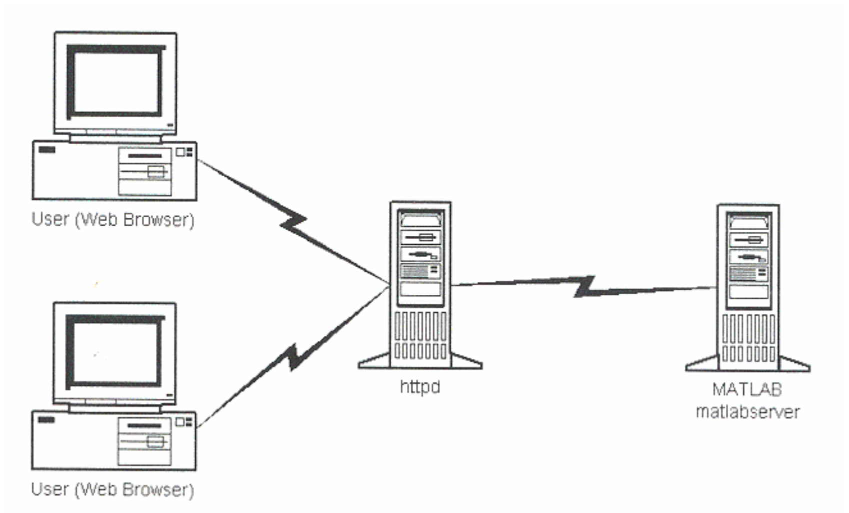


Figure 3. In a more complex network the web server daemon can run on a different machine.

2.1.1 Building MWS Applications

MWS applications are a combination of m-files, HTML, and graphics whenever needed. The application development process requires a small number of steps:

- Create the HTML documents for collection of the input data from users and display of output.
- List the application name and associated configuration data in the configuration file `matweb.conf`.

- Write a Matlab m-file that:
  - Receives the data entered in the HTML input form.
  - Analysis the data and generates any requested graphics.
  - Places the output data into a Matlab structure.
  - Calls `htmlrep` to place the output data into an HTML output document template.

2.1.2 Product Requirements

The MWS requires Matlab release 11 or later and that TCP/IP networking software are installed on the computer.

2.1.2.1 Web Requirements

To submit input to and receive output from the MWS, a web browser suitable for the platform must be installed. Both Netscape Communicator Release 4.7 and Microsoft Internet Explorer 5.0 or later have been tested.

Web server software (HTTPD or similar) is needed on the system where Matlab is running or in a machine that has network access to the machine where Matlab is running. The web server software must be capable of running CGI programs.

2.1.3 MWS Components

The MWS consists of a set of programs that enable Matlab programmers to create Matlab applications and access them on the web, (see figure 4):

- MWS:
  - manages the communication between the web application and Matlab.
  - a multithreaded TCP/IP server (service on Windows NT) running Matlab continuously.
  - invokes `matweb.m`, which in turn runs the m-file specified in the input HTML-document.

- `matweb.exe`:
  A TCP/IP client of MWS. Using CGI this program translates HTML form data into a Matlab object, passes it to MWS, receives HTML results from MWS, and transmits these results back to the end user's web browser.

- `matweb.m`:
  Main entry point for MWS. Calls the m-file that is given in `matweb.exe` and run the Matlab Application.

- Input HTML form data functions – Matlab utility functions that retrieve data from input HTML documents and then provide that data in a form convenient to Matlab programmers.

- Output HTML document functions – Matlab functions for inserting Matlab data into template HTML forms for return transmission to the end user's web browser.

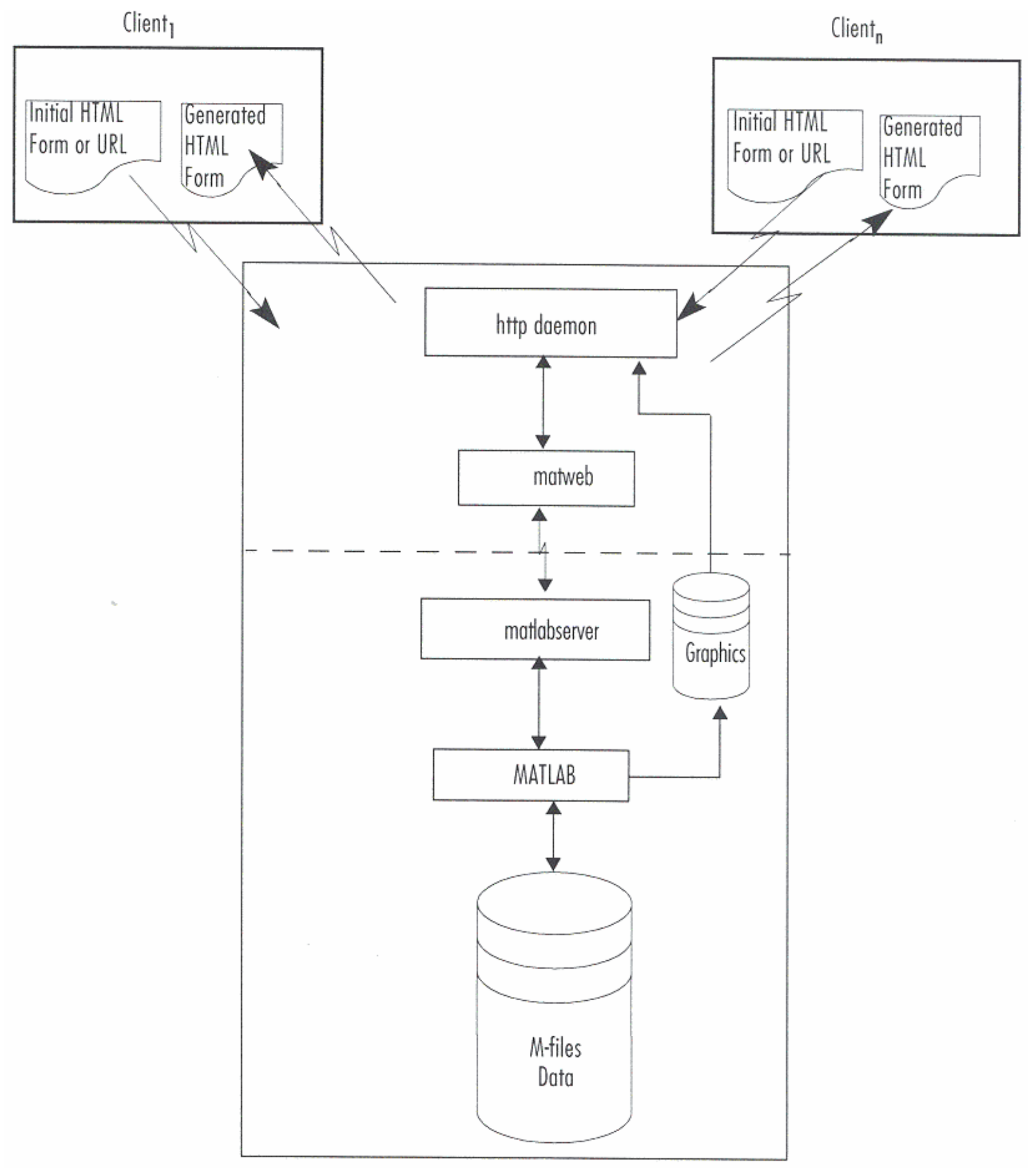


Figure 4. Matlab on the web, is showing how Matlab operates over the web

Three configuration files are used in conjunction with the MWS programs:

- matweb.conf:
  A configuration file that matweb.exe needs for connecting to MWS. Applications must be listed in matweb.conf.

- matlabserver.conf:
  MWS can be configured to listen on any legal TCP/IP port by editing the matlabserver.conf file on Windows NT. The number of simultaneous Matlabs is specified here.

- hosts.conf:

An optional file providing additional security. If hosts.conf is present, only listed machines can connect to the MWS. Machines are listed by name in a single column. Machines must be listed by name not by IP number. The operating system resolves the name into a valid IP address.

2.1.4 Important MWS functions

htmlrep(instruct, infile) replaces all Matlab variables in infile, an HTML document, with corresponding values of variables of the same name in instruct. Variables can be character strings, matrices, or cell arrays containing strings and scalars. String and scalar variables are replaced by straight substitution. Output is returned in outstring.

```
outstring = htmlrep(instruct, infile)
```

instruct is a Matlab structure containing variable names (field names) and corresponding values.
infile is a HTML file with Matlab variable names enclosed in dollar signs, e.g. $variable_name$.

matweb(instruct) is an m-file that in turn calls a Matlab application m-file stored in the mlmfile field of Matlab structure instruct. It also passes instruct to the application. The matweb.m function is invoked by MWS.
instruct contains the fields:

- All the data from the HTML input document
- mlmfile, which stores the name of the m-file to call
- mldir, the working directory specified in matweb.conf
- mlid, the unique identifier for creating filenames and maintaining contexts

2.1.4.1 File Locations

Any m-file used in conjunction with a web application, including matweb.m must appear on the Matlab path. The matweb.exe and matweb.conf files must appear under a /cgi-bin alias. Any generated graphics must be located where the web Server can find them and programs can write them.

2.1.5 Understanding MWS

MWS is designed to run continuously in the background as a Windows NT service or as a background process on other systems.

2.1.5.1 matlabserver.conf

When MWS starts up, it looks in the file matlabserver.conf for its initial setting data. On Windows NT the installation procedure creates this file in the <Matlab>/webserver directory while installing the MWS.

Configuration settings must appear on the first line of the matlabserver.conf file. The basic options set by the data in matlabserver.conf are:

- Port number
- Threads (maximum number of simultaneous Matlabs)

The default version of matlabserver.conf is simply -m 1 meaning it will run one copy of Matlab with the MWS port defaulted to 8888.

2.2 Apache Server

Apache is an HTTP Server, originally designed for Unix systems. Apache version 1.3 (and up) is a version for Microsoft Windows 2000, NT, 98 and 95. TCP/IP networking must be installed but Apache cannot share the same port with another TCP/IP application. This includes (but are not limited to) other web servers, and firewall products.

Apache can be installed either as a service (for all users) or to run in a consol window. In this thesis Apache is used to regulate the communication between Matlab and the web browser therefore must run as a service due to MWS design. In this way it will automatically start when the machine boots and keep it running when logging off.

After installing Apache the configuration files should be edited, in a suitable way, in the conf directory. There are lots of options, which should be set before starting to use Apache.

Because Apache for Windows is multithreaded, it does not use a separate process for each request, as Apache does with Unix. Instead there are usually only two Apache processes running: a parent process, and a child, which handles the requests. Within the child a separate thread handles each request. So, "process"-management directives are different: The directives that accept filenames, as arguments now must use Windows filenames instead of Unix ones. However, because Apache uses Unix-style names internally, forward slashes must be used, not backslashes. Drive letters can be used; if omitted, the drive with the Apache executable will be assumed.

Apache for Windows has the ability to load modules at runtime, without recompiling the server. If Apache is compiled normally, it will install a number of optional modules in the modules directory. To activate these, or other modules, the new LoadModule directive must be used. [4]

2.3 PHP-script

PHP is an open-source server-side scripting language that is especially suited for web development and can be embedded into HTML code. The PHP code is enclosed in special start and end tags that allows jumping into and out of "PHP mode". What distinguishes PHP from something like client-side JavaScript is that the code is executed on the server.

Server-side scripting is the most traditional and main target field for PHP. To make this work three things are needed, the PHP parser (CGI or server module), a web server and a web browser. The web server needs to run with a connected PHP installation. With a web browser it is possible to access the PHP program output, viewing the PHP page through the server.

PHP can be used on all major operating systems. PHP also has support for most of the web servers today. This includes Apache, Microsoft Internet Information Server, Personal Web Server, Netscape and many others. For the majority of the servers, PHP have a module, for the others supporting the CGI standard, PHP can work as a CGI processor. PHP also has support for talking to other services using protocols such as HTTP and COM (on Windows).

PHP-enabled web pages are treated just like regular HTML pages and can be created and edited the same way normally regular HTML pages are created.

One of the most powerful features of PHP is the way it handles HTML forms. The basic concept, that is important to understand, is that any form element in a form will automatically result in a variable with the same name as the element being created on the target page.

There are two ways to set up PHP to work with the Apache server on Windows. One is to use the Apache module interface (dll) and the other is the CGI binary. For many servers PHP has a direct module interface (also called SAPI). Many servers have support for ISAPI, the Microsoft module interface. If PHP has no module support for the web server in use, it is possible to use it as a CGI processor.

The configuration file (called php.ini in PHP 4.0) is read when PHP starts up. For the server module versions of PHP, this happens only once when the web server is started. For the CGI version, it happens on every invocation. [5]

2.4 CGI

CGI is a standard for interfacing external applications with information servers, such as HTTPD (web server daemon) or web servers. A plain HTML document that the Web daemon retrieves is static, which means it exists in a constant state: a text file that does not change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information.

A CGI program can be written in any language (C/C++, PERL, TCL, Any Unix shell, Visual Basic) that allows it to be executed on the system. CGI scripts are easier to debug, modify, and maintain than a typical compiled program.

2.5 TCP/IP

TCP - is responsible for verifying the correct delivery of data from client to server. Data can be lost in the intermediate network. TCP adds support to detect errors or lost data and to trigger retransmission until the data is correctly and completely received.

IP - is responsible for moving packet of data from node to node. IP forwards each packet based on a four-byte destination address (the IP number). The Internet authorities assign ranges of numbers to different organizations. The organizations assign groups of their numbers to departments. IP operates on gateway machines that move data from department to organization to region and then around the world.

3 Problem Description and Analysis

The main topic of this thesis is to add functionality to the DDA application, in order to make it possible to run a compiled version of the application without having Matlab installed and through a web server where Matlab is installed being able to do advanced simulations. For this purpose MWS is provided.

The task is divided into four parts. The first one is to install and get the MWS work properly and to study the MWS application demo examples following with the installation. Next step is to make a test program that calls MWS, performs a simulation and send the result back again.

The third part is to implement the MWS application into the DDA application. After that usability is in focus and finally a usability test is planned.

Problem hierarchy

1. Installing and getting the MWS work and study MWS demo examples.
2. Make a test program that calls the MWS, does a simulation and send the result back again.
3. Implementation in the DDA Application
4. Usability
5. Test

3.1 Model/Method

Due to the structure of the problem including the structure of the MWS and the rather informal requirements, the prototyping model (figure 5) was chosen as a process model for this project. It can itself be the basis for an effective process model, since the prototyping model allows all or part of a system to be constructed quickly to understand or clarify issues. It has the same objective as an engineering prototype, where requirements or design require repeated investigation to ensure that the developer, user, and customer have a common understanding both of what is needed and what is proposed. In this thesis the prototyping model was used to demonstrate feasibility of an approach and design as well as it was a way of getting the right functionality, step by step.

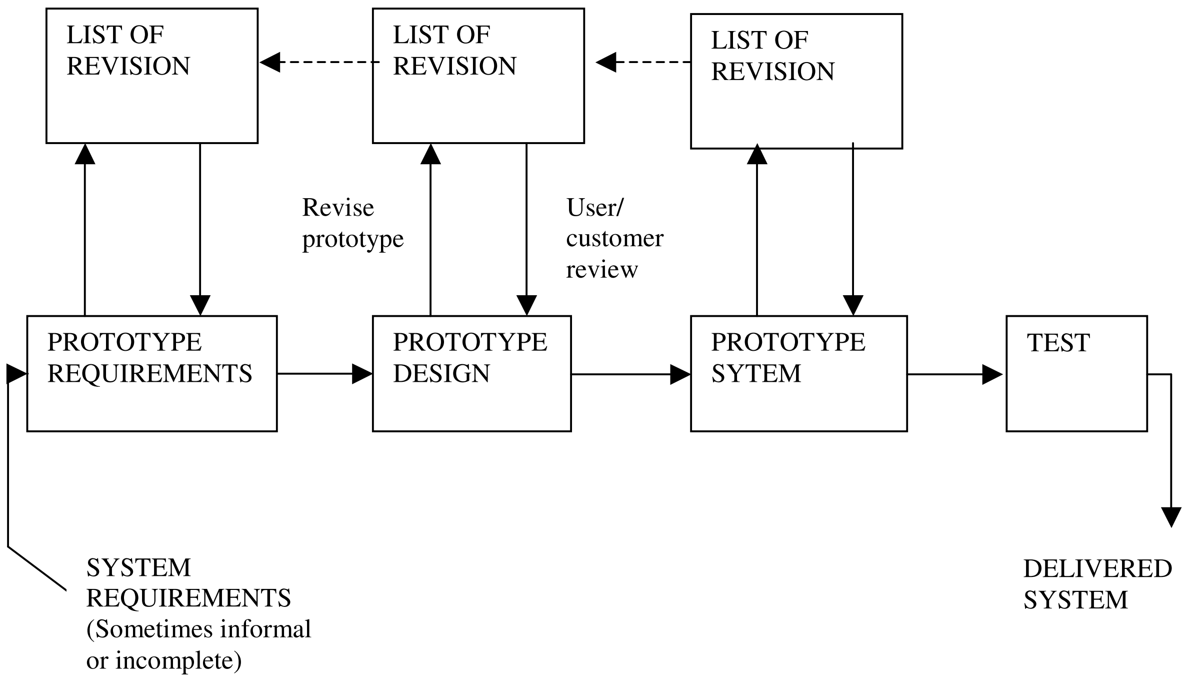


Figure 5. The prototyping model

4 Solution

After installing, configuring and learning how MWS works a test program was made. The starting point was to use MWS and that the MWS application should be implemented in the existing DDA application in a suitable way. The predestined way to store the DDA parameters is in .mat-files. According to this the solution to transmit data in files to and from MWS was a natural choice. On the other hand how the file transmission should be done was an open question. The wish was that the transmission should be as automatically as possible in both directions. From this test program the final result has been developed using the prototyping model, step-by-step getting the right design and functionality.

4.1 Installing, configuring and studying MWS

Last summer a trainee, Lisa Gelin, did a pilot study in the MWS. The first task was to take a look at her work; some instructions and a project map with files. As local server to the MWS she had downloaded OmniHTTPD. She gave us some tips, but in the end it turned out to be easier following MWS installation guide in order to get the MWS demo files to run. Finally after a couple of reinstallations the MWS could run the demos properly.

Next step was to study the MWS demo applications and Matlab in order to learn more about the programming. Having contact with Matlab support resulted in advice and more material how to do the programming part.

4.2 Making a test program

This part is divided into three parts; upload a file to the HTML page, establishing communication with MWS and finally get the result file back to the client.

4.2.1 File uploading to a HTML page

One solution, doing a file upload, is to use PHP-script. Installation and configuration of PHP is necessary. Through the following lines of HTML code a file can be uploaded.

```
<form enctype=”multipart/form-data”  method=“POST”>
<input type=”file” name=”filename”>
<input type=”submit” name=”submit” value=”send file”>
```

Configuring a form for file uploads requires setting two attributes in the `<form ...>` tags in addition to use `<input type=”file”>`: “POST” and “multipart/form-data” (as in the example above). When the data is sent, the original name of the file is sent to the web server. `enctype` determines how the form data is encoded. Whenever data is transmitted from one place to another, there need to be an agreed upon means of representing that data.

A PHP-script is then able to move the uploaded file from the web page to a path on the MWS, give the file a unique ID and finally parse the file to MWS. File uploads allows sending an entire file from a client computer to the web server as part of the form input.

Another conceivable solution for file upload is for example to use FTP in combination with Visual Basic. This demands additionally investigations due to the internal data security policy at ABB.

4.2.2 Establish communication with MWS

The standard procedure for establishing communication is this line of HTML code (in the input HTML document)

```
<form action="/cgi-bin/matweb.exe" method="POST">
```

matweb.exe is a program that resides on the HTTP server and communicates with the MWS. It requires information found in matweb.conf to locate the MWS (which could be running on a different machine). matweb.exe is a client of MWS that uses CGI to get data from HTML forms. It transfers the information to MWS, which invokes matweb.m, which in turn runs the m-file mtest.m from this line of HTML code:

```
<input type="hidden" name="mlmfile" value="mtest">
```

This line sets argument mlmfile to the value mtest. The mlmfile argument contains the name of the Matlab m-file to run. MWS uses the value of the mlmfile obtained from the matweb.m to run the Matlab application. However when a file is going to be uploaded to MWS the form tag must be

```
<form enctype="multipart/form-data" method="POST">
```

This in turn makes it necessary to call matweb.exe in another way than standard procedure. For example using the PHP-function virtual(), which also passes the string &filename to MWS with this line of PHP script:

```
virtual("/cgi-bin/matweb.exe?mlmfile=mtest" . "&filename=" . $strFilename);
```

There were some problems with getting virtual() to run. It turned out to be an Apache specific function, meaning that a change of the original local server for MWS was necessary. After installation and configuration of the Apache server virtual() still did not work, information was found on the official homepage that virtual() did not run under Windows. Since PHP is an open-source code there are some difficulties to get support. Some other functions were tested to substitute virtual(), but none of them worked. Finally, after a lot of effort, the conclusion was that PHP was installed as a CGI-script instead of as a module. After reinstallation of PHP as a module virtual() was finally working.

Another solution was to use a link to start MWS, since this solution resulted in an extra mouse click the usability decreased and was therefore abandoned.

4.2.3 Get a result file back to the client

The main issue in this part was how to get the result file back to the client's computer. The standard procedure for output from the MWS is showing the result statically in the web browser. The mtest.m function uses the htmlrep command to places the computed values into the output HTML document (toutput.html) using the code

```
str = htmlrep(s, 'toutput.html');
```

Here `s` is a Matlab structure containing the results of the `mtest` computation. `htmlrep` extracts data from `s` and replaces variable fields in `toutput.html` with the results of Matlab computation. The completed `toutput.html` form is transmitted to the user's browser.

Presenting the graphics in the browser with this method, there is no way of having a closer look into graphics the way it is possible under Matlab. The best solution for the client would be getting the result file back to his computer and using the client DDA application for showing the result.

One idea of how to get the result file back to the client's computer, as well as to the server, was to use Java Sockets, but the idea failed because it was not possible to use the Matlab Compiler on files consisting of Java.

Another suggestion was to use GNU Wget, which is a free software package for retrieving files from the World Wide Web using HTTP and FTP, the two most widely used Internet protocols. Using Wget would imply installations at the client side and was therefore abandoned.

A simple solution is to have a link for download, consisting of the result file, in the user's web browser. This way seemed to be the right way to go.

4.3 Implementation in the DDA Application

To make the test program work from the DDA application on the client side a new function was needed. This function, named `Webfun()` was made to be a link between the DDA application and the test program (MWS). Some changes were also required in other functions in the DDA. The main one, `Simulateandplot()`, calls `Webfun()` if any parameter are changed (if calculations are needed by the MWS).

On the server side (ABB) some changes, made by Mattias Olsson, was required to `simprocess()` and functions called by `simprocess()`, the main function for simulations. At first the test program did not work running it through the MWS. The debugging part took quite some effort because it was not possible to use the debugger on m-files running through the MWS. The only way to debug `simprocess()` through MWS was line by line. Finally the problem was found; the MWS did not work if the functions used for simulations contained outputs to the command window in Matlab.

It is necessary that all files used for simulation are located in a hard drive on the same computer as MWS. To have the files in a network demands special settings and these settings requires a high administrator level out of our authority.

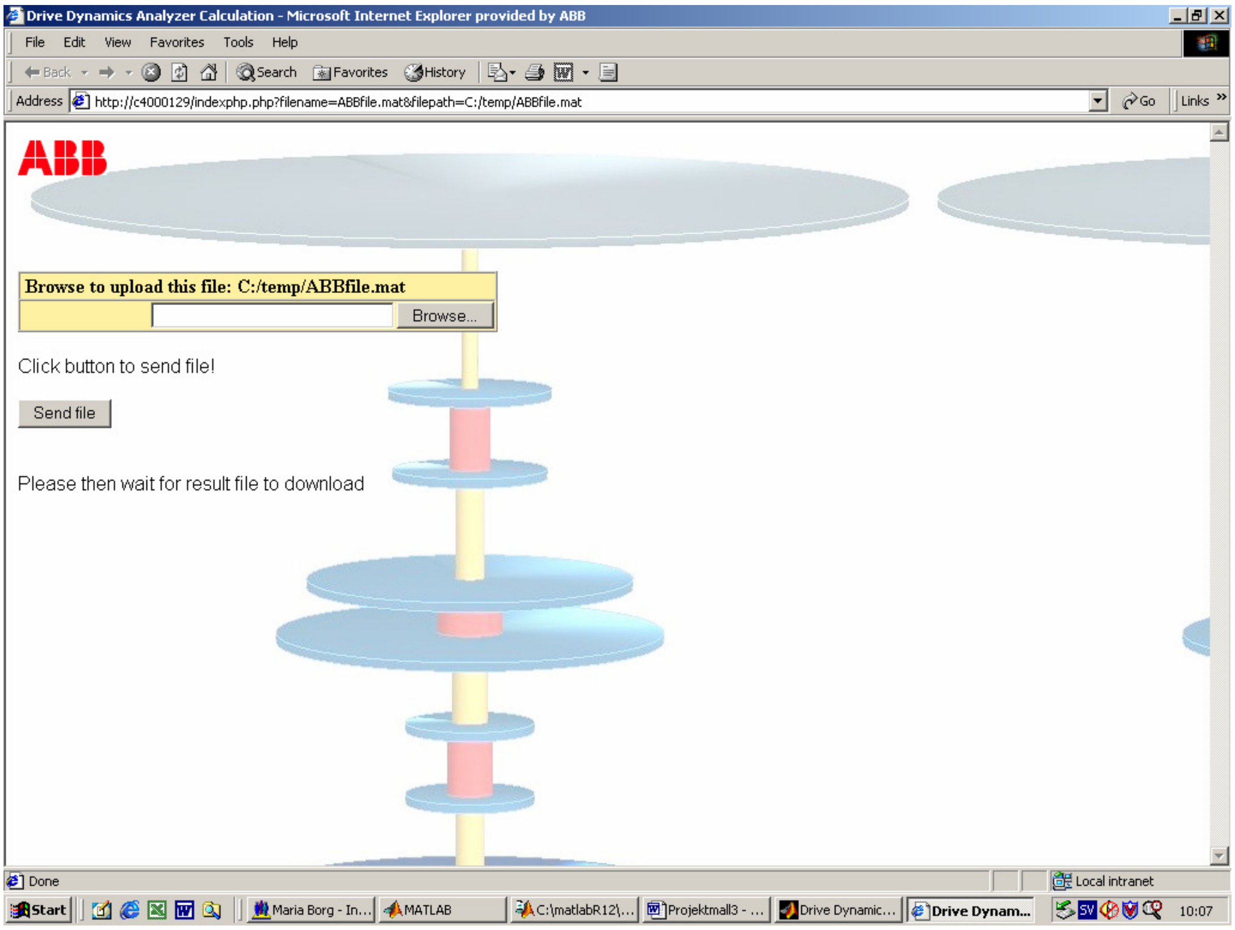
4.4 Usability

Usability is an important part in all applications having a user interface.

4.4.1 Input and Output HTML documents

Forming the client side interface, the input and output HTML documents, high usability was needed in order to minimise error occurring and make the interface as easy to use as possible.

For this purpose default color used in ABBs homepage was chosen for link colors and logotype color. A picture from the DDA application was also inserted as a background into the input document in order for the client to get recognition (see figure 6). To minimise the number of errors in the file upload from the client side the filename is written out in the browse table in the input document. Choosing another file will result in an error message.


Drive Dynamics Analyzer Calculation - Microsoft Internet Explorer provided by ABB

Address http://c4000129/indexphp.php?filename=ABBfile.mat&filepath=C:/temp/ABBfile.mat

Browse to upload this file: C:/temp/ABBfile.mat

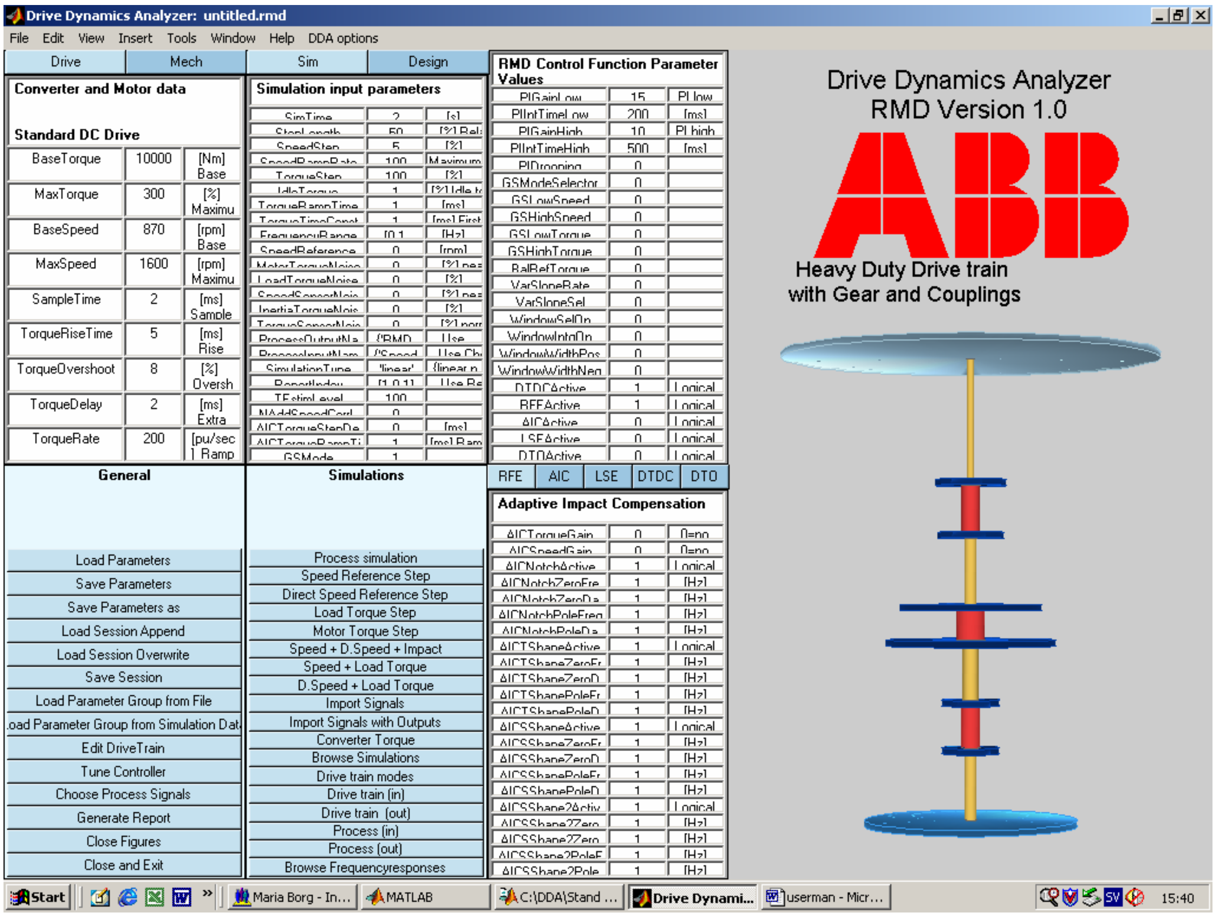
Click button to send file!

Please then wait for result file to download

Figure 6. The input and output HTML document view

4.4.2 DDA application

The DDA application is not yet compiled to be a stand-alone application, meaning customers all over the world have not been in contact with the application. Outside the specification of this project a proposal is made for increasing the usability of the DDA application. Colors on buttons, table headings, and figures have been changed to better match the ABB overall color policy, and to give a calmer and more solid impression (see figure 7).



Converter and Motor data

Standard DC Drive

BaseTorque	10000	[Nm] Base
MaxTorque	300	[%] Maximu
BaseSpeed	870	[rpm] Base
MaxSpeed	1600	[rpm] Maximu
SampleTime	2	[ms] Sample
TorqueRiseTime	5	[ms] Rise
TorqueOvershoot	8	[%] Oversh
TorqueDelay	2	[ms] Extra
TorqueRate	200	[pu/sec] Ramp

Simulation input parameters

SimTime	2	[s]
StepLength	50	[%] Rel
SpeedStep	5	[%]
SpeedRampRate	100	Maximum
TorqueStep	100	[%]
IdleTorque	1	[%] Idle t
TorqueRampTime	1	[ms]
TorqueTimeConst	1	[ms] First
FrequencyRange	[0.1	[Hz]
SpeedReference	0	[rpm]
MotorTorqueNoise	0	[%] nea
LoadTorqueNoise	0	[%]
SpeedSensorNois	0	[%] nea
InertiaTorqueNois	0	[%]
TorqueSensorNois	0	[%] norr
ProcessOutputNa	{'RMD	Use
ProcessInputNam	{'Speed	Use Ch
SimulationType	'linear'	{linear n
ReportIndex	[1 0 1]	Use Re
TEstimLevel	100	
NAddSpeedCorl	0	
AICTorqueStepDe	0	[ms]
AICTorqueRampTi	1	[ms] Ram
GSMode	1	

RMD Control Function Parameter Values

PIGainLow	15	PI low
PIIntTimeLow	200	[ms]
PIGainHigh	10	PI high
PIIntTimeHigh	500	[ms]
PIDrooping	0	
GSModeSelector	0	
GSLowSpeed	0	
GSHighSpeed	0	
GSLowTorque	0	
GSHighTorque	0	
RalRefTorque	0	
VarSlopeRate	0	
VarSlopeSel	0	
WindowSelOn	0	
WindowIntgOn	0	
WindowWidthPos	0	
WindowWidthNeg	0	
DTDCActive	1	Logical
RFEActive	1	Logical
AICActive	0	Logical
LSEActive	0	Logical
DTOActive	0	Logical

Adaptive Impact Compensation

AICTorqueGain	0	0=no
AICSpeedGain	0	0=no
AICNotchActive	1	Logical
AICNotchZeroFre	1	[Hz]
AICNotchZeroDa	1	[Hz]
AICNotchPoleFreq	1	[Hz]
AICNotchPoleDa	1	[Hz]
AICTShapeActive	1	Logical
AICTShapeZeroFr	1	[Hz]
AICTShapeZeroD	1	[Hz]
AICTShapePoleFr	1	[Hz]
AICTShapePoleD	1	[Hz]
AICSShapeActive	1	Logical
AICSShapeZeroFr	1	[Hz]
AICSShapeZeroD	1	[Hz]
AICSShapePoleFr	1	[Hz]
AICSShapePoleD	1	[Hz]
AICSShape2Activ	1	Logical
AICSShape2Zero	1	[Hz]
AICSShape2Zero	1	[Hz]
AICSShape2PoleF	1	[Hz]
AICSShape2Pole	1	[Hz]

Figure 7. DDA application start view

4.5 Test

Function test is already performed using prototyping as process model. Instead focus was on doing a usability evaluation, since the number of mouse clicks has been a big question during the development of the system.

4.5.1 Usability Evaluation

Evaluation of a hifi prototype or an implementation can be performed as a scientific experiment, an observation, and interview or with a questionnaire. Observation performed as a cooperative evaluation was chosen for this MWS application. In a cooperative evaluation the user thinks aloud and informs the evaluator about what his intentions are. He also continuously gives information about what he sees on the screen and his apprehension of the application. It is possible for both the user and the evaluator to ask questions. This makes the user feel comfortable and is encouraged to criticize the application. The data from the observation is entered into a protocol. The protocol can be performed in different ways; Recording, filming, paper and pencil or the user doing his own notes. For this protocol paper and pencil was used.

Test persons for this usability evaluation were chosen for their knowledge of the DDA and are also future possible users of the application. Both were already familiar with the DDA application. They did not get any instructions or manual to read before the test, because the intention was to see how self instructed the web site was.

Anders Hansson, engineer at ABB PI/FVA

Anders was already familiar with the DDA application. He noticed the changes of the colors immediately and found them better and more peaceful for the eyes. He had some thoughts about the table of parameters, but that part was not included in the application. In the input document at the local web server he was at first a bit confused what to do. The instructions how to browse the file was not easy to see and the line, which caught his attention, was not the one which he was supposed to seen first.
After doing the file transmission once there was no need for further instructions. Generally he found the web site simple and easy to use.

After the first test person had finished some changes were made to the input document; changing the color in the browse table and from bold to normal in an instruction line.

Per Lennart Eriksson, engineer at ABB PI/FVT

Per Lennart thought that the new colors of the DDA application gave a calmer impression and was less tiring to look at. At the first look at the new part of the DDA application, the input document, he was a bit unsure what to do. His opinion was that the web site should be as plain as possible, but the text lines could be a little bit more instructive.

The input document was changed again after the last test person; the text lines were changed to a more instructive content.

5 Result

A schematic overview of the result is shown in figure 5 followed by descriptions of the functions.

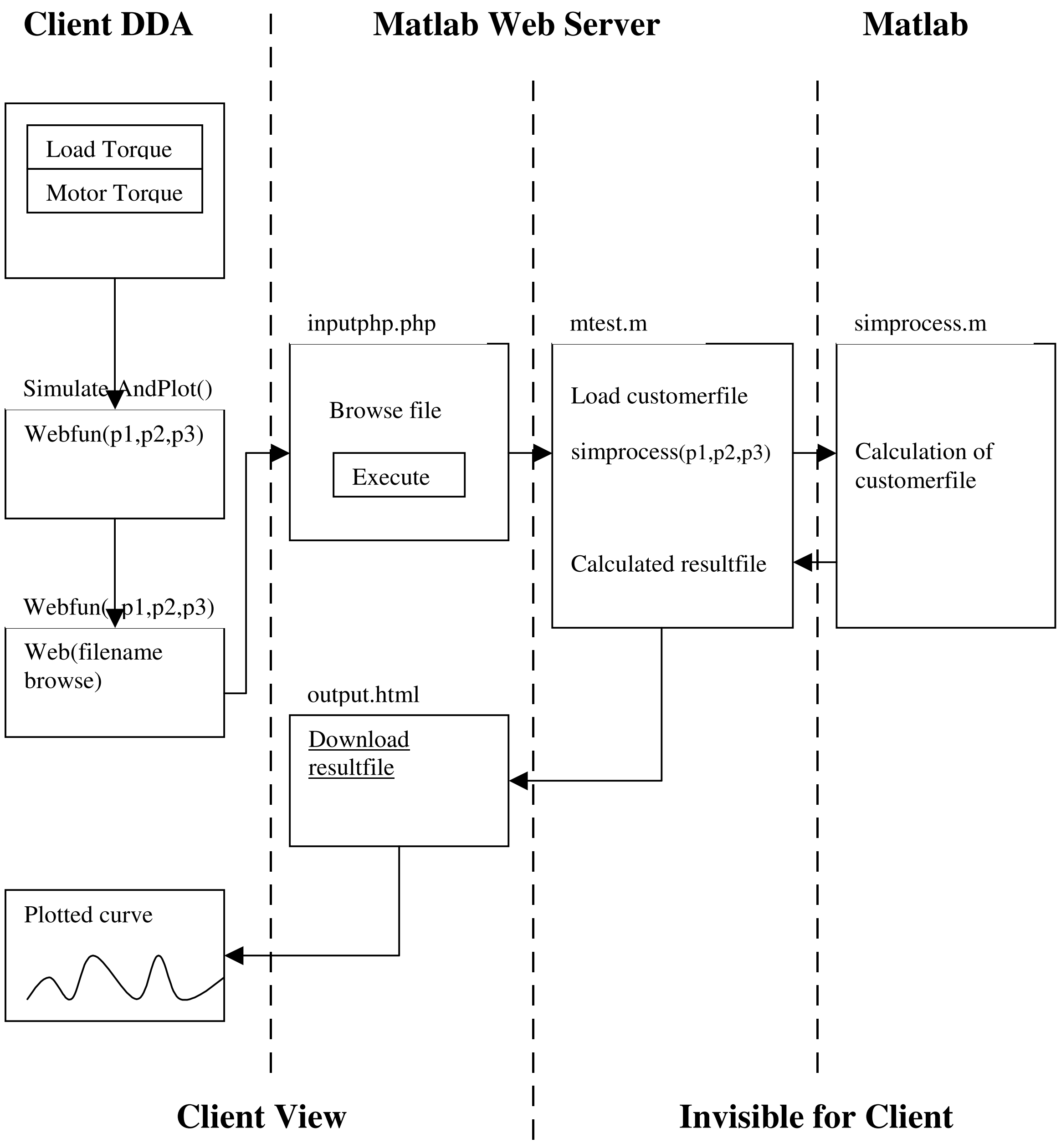


Figure 8. Flowchart diagram of file transmission

Simulateandplot(ParList, menuchoice, fileandpath) is called when a client makes a choice, in the simulation menu in the DDA application, in order to do a simulation. Depending on the settings of the parameters Webfun() is called for advanced simulations over MWS. Before

Webfun() is called a string holding the name of the result file is initialised. The string is one of the parameters to Webfun() and its purpose is to name the result file coming from the MWS. The advantage with this is that whilst the simulation is running over the MWS, a timer is set to poll for the result file to come back. When the result file is downloaded to the client's computer it is opened and plotted directly, without any effort from the client.

Webfun(ParList, menuchoice, menuitems, resultfilename) is the link between the DDA application and the MWS application. It is called from the DDA function SimulateandPlot() if the simulation requires calculations through MWS. The three first parameters (ParList, menuchoice, menuitems) are data required by MWS for processing correct simulation and the fourth (returnfilename) is holding customer identification in order to give the result file a name recognized by the DDA application. The parameters are saved in a .mat file with a predestinated name in order to avoid the client to choose wrong file doing the file upload to the MWS. Webfun() calls the Matlab function web(), which starts up MWS local web server (Apache) and shows the input document. The three parameters of function web() are the http-address to the local server, a string holding the predestinated name of the .mat file and the file path to the directory where the .mat file is destinated.

indexphp.php is the input HTML document for data submission to the MWS. The client must browse the given filename from the given path name and push the button labeled 'Send file'. This will start a php-script; which checks if the uploaded file is the right one. The file gets an unique identification through an php-function called uniqid(). If the file does not already exists the function move_uploaded_file() tries to move the file to the MWS directory. If that succeeds the function virtual() will call matweb.exe, which transfer the information to MWS, which in turn runs the m-file specified.

mtest.m(h, outfile) is the m-file, specified in the HTML input document, that receives the data for simulation from indexphp.php. The working directory is set by the variable h.mldir, which is automatically provided to all MWS applications. The data from the instruct h, is saved in a local variable and loaded into the Matlab working space. mtest.m analyses the data and generates the requested output data through calls to the functions simprocess() and datahistory(). The output data is then saved into a result file named by the fourth parameter in the input data. Finally the output data is placed into a Matlab structure and a call is made to the Matlab function htmlrep(s, path) to place the output data into an HTML output document (e.g output.html).

output.html is the MWS output HTML document providing the output data to the customer. Output data is a result file (.mat-file) ready to be plotted in the DDA application. In output.html the result file is given in a link to the client ready to be downloaded. When the result file is downloaded to the client's computer the plotting is shown directly in the DDA application.

5.1 Result Analysis

To start with there was some problems with the installation and configuration of the MWS, partly due to our lack of Matlab experience.

The test program, which was started up, should handle file transmission. A suggestion to a solution was to use Java sockets. This solution was abandoned due to Matlab Compilers inability to read files containing Java. Instead of Java sockets PHP script seemed to be a better choice. This worked out well but another problem that pursued us throughout the implementation occurred; the function virtual(). The local server used in the beginning (OmniHTTPD) could not run virtual() and it turned out to be an Apache server specific function. virtual() was the best solution for calling MWS and post the parameters from the input document; according to this the local server was changed to Apache. Since Apache and PHP are freeware products the possibility to get support has been quite difficult. With the PHP installation some more problems occurred. PHP can be installed in to ways, as CGI or as a module. To make virtual() work PHP should be installed as a module and the php.ini file should be configured in the right way.

One of the other problems has been the difficulty to debug. The error messages gives inexact information about where and what the errors are and debugging the .m-files, which runs under MWS, have not worked at all.

Requirements have been changed and new ones have appeared during the work. That is one of the advantages with using the prototyping model. One desire that appeared was the ability to identify the owner of the result simulation files. If the customer has any problem with the result file ABB can easily find corresponding result file in the MWS directory and be able to look into the problem. Having an extra parameter, holding an identification of the customer, in the .mat file sent for simulation, solved this. The result file gets its name from this parameter. In this way it is also possible for ABB to see which customer are using the MWS application.

The “dream vision” was that the file transmission between ABB and the client should pass without the client “noticing” anything, meaning that the input and output HTML documents are not needed more than as an informative purpose (e.g. Please wait for simulations to be done). The “dream vision” was not reached. Our solution is in the scope of standard procedure for how to make an application using the MWS.

6 Recommendation, Future work

The focus has been on making a MWS application work well together with the existing DDA application. The most important issue has been to get it all work; consequently there are improvements to make.

It would have been desirable if the .mat-file, which is going to be uploaded to the MWS, in some way can be uploaded automatically or be shown as default in the browser field. Another improvement would be if the result file could automatically be downloaded back to the customer’s computer.

How the customer file identification is going to be set is not yet decided. At the present it is hard coded and sent in a file as a parameter.

The solution of where the customer retrieves and downloads the file can be further developed. Now the input document gives the information about which directory to choose.

In the present situation it is impossible to do a sharp test, run a compiled version of the DDA application, though it is still under development. It will be ready any day. According to this the MWS application needs to run under Matlab and in ABBs internal network.

7 Conclusions

Our task was to enable an on-line connection between ABB and, in first hand, their customers through the MWS in order for the customers and engineers/sellers at ABB who are out on jobs, to do advanced calculations and simulations, without having Matlab installed on their computers. This should be done with as little alteration as possible in the existing DDA.

The requirement was that this connection should be made easy for the customer. The user interface should contain as few mouse clicks as possible to ensure high usability and as little effort as possible for the customer.

The task also covered to install MWS, configure it in a suitable way for the task and adjust the parts affected of the file transmission. The task has been solved using prototyping since it was a suitable way of learning and understand the MWS, due to the fact that the structure of MWS is making it easy to build a test program. It also allows developer, user and customer to have a common understanding both of what is needed and what is proposed.

To summarize our thesis, we consider that we have reached our goals to enable advanced web services, with a MWS application, for a client server communication. Finally we hope that our work will be useful for ABB now and in the future.

8 References

[1] Chapman, Stephen J., (2000), *MATLAB Programming for Engineers*, Brooks/Cole, Pacific Grove

[2] (2000), *Using Matlab Version 6 – The Language of Technical Computing*, The MathWorks, Inc., Natick

[3] (2000), *Matlab Web Server Version 1.2 – The Language of Technical Computing*, The MathWorks, Inc., Natick

[4] The Apache Software Foundation, http://www.apache.org

[5] The PHP Group, http://www.php.net

[6] Comsol, http://www.comsol.se

9 Appendix

Code and Configuration files
MWS Installation and Configuration guide
User manual