ISSN 0280–5316
ISRN LUTFD2/TFRT--5639--SE

# Communication between Matlab/Simulink and ABB Advant Control Builder

Peter Davidsson
Fredrik Hansson

Department of Automatic Control
Lund Institute of Technology
March 2000

**Department of Automatic Control**
**Lund Institute of Technology**
**Box 118**
**SE-221 00 Lund Sweden**

*Document name*
MASTER THESIS

*Date of issue*
March 2000

*Document Number*
ISRN LUTFD2/TFRT--5639--SE

*Author(s)*
Peter Davidsson, Fredrik Hansson

*Supervisor*
Patrik Svensson, Per-Inge Tallberg
ABB Automation Products
Tore Hägglund, Reglerteknik

*Sponsoring organisation*

*Title and subtitle*
Communication between Matlab/Simulink and ABB Advant Control Builder
(Kommunikation mellan Matlab/Simulink och ABB Advant Control Builder)

*Abstract*

Possibilities to simulate control systems can be very useful. During the development of a control system, simulations can verify that the implemented algorithms work as expected. Smaller parts of the implementation can be tested before they are put together to a complete control system. In an existing plant simulation can be used for optimization of the control system and control parameters can be tuned without affecting the real process.

The goal of this thesis has been to enable communication between Advant Control Builder and Matlab/Simulink. Advant Control Builder is a product developed by ABB, used for design and development of control systems. Matlab is a program for mathematical computation,and Simulink is a toolbox to Matlab that enables simulation of mathematical systems. Matlab and Simulink are developed by Mathworks. The communication is to be used for simulation of control systems and the processe s that the system controls. The control system is created in Advant Control Builder and the process is modeled in Simulink.

A way to access data in the Advant Control Builder is through an OPC server, where OPC stands for OLE for Process Control. To enable communication between the OPC server and Matlab we have implemented a gateway. The gateway is implemented in C++ and can be seen as a link between the OPC server and Matlab. The communication between the gateway and the OPC server is built on COM (Component Object Model), and the communication between the gateway and Matlab is built on ActiveX. For exchange of data between Matlab and Simulink, we have implemented functions in Simulink that writes and reads data from Matlab workspace.

*Key words*

*Classification system and/or index terms (if any)*

*Supplementary bibliographical information*

*ISSN and key title*
0280–5316

*ISBN*

*Language*
English

*Number of pages*
48

*Recipient's notes*

*Security classification*

# Abstract

Possibilities to simulate control systems can be very useful. During the development of a control system, simulations can verify that the implemented algorithms work as expected. Smaller parts of the implementation can be tested before they are put together to a complete control system. In an existing plant simulation can be used for optimization of the control system and control parameters can be tuned without affecting the real process.

The goal of this thesis has been to enable communication between Advant Control Builder and Matlab/Simulink. Advant Control Builder is a product developed by ABB, used for design and development of control systems. Matlab is a program for mathematical computation, and Simulink is a toolbox to Matlab that enables simulation of mathematical systems. Matlab and Simulink are developed by Mathworks. The communication is to be used for simulation of control systems and the processes that the system controls. The control system is created in Advant Control Builder and the process is modeled in Simulink.

A way to access data in the Advant Control Builder is through an OPC server, where OPC stands for OLE for Process Control. To enable communication between the OPC server and Matlab we have implemented a gateway. The gateway is implemented in C++ and can be seen as a link between the OPC server and Matlab. The communication between the gateway and the OPC server is built on COM (Component Object Model), and the communication between the gateway and Matlab is built on ActiveX. For exchange of data between Matlab and Simulink, we have implemented functions in Simulink that writes and reads data from Matlab workspace.

# Table of contents

# Preface

This report presents the Master thesis “Communication between Advant Control Builder and Matlab/Simulink”. The thesis has been performed at ABB Automation AB in Malmö and the Department of Automatic Control, Lund Institute of Technology. The thesis has been a cooperation between the two divisions Automation Products and Automation Systems at ABB.

# Acknowledgements

We would like to thank our supervisors Tore Hägglund at the Department of Automatic Control, Lund Institute of Technology, Per-Inge Tallberg at ABB Automation Systems AB and Patrik Svensson at ABB Automation Products AB.

# 1 Introduction

The main goal of this master thesis is to establish communication between Advant Control Builder and Matlab/Simulink. Advant Control Builder is a design environment for control systems, developed and used by ABB. Matlab is a language for technical computing developed by MathWorks. Simulink is a toolbox to Matlab that facilitates simulation. With Simulink it is quite easy to build up systems to be simulated by click and drag operations.

Advant Control Builder communicates with other applications through an OPC Server. ABB has developed its own OPC Server that we used. To establish communication between the OPC Server and Matlab we have implemented a Gateway. The Gateway exchanges data with the Advant Control Builder by a connection to the OPC-servers I/O-interface. We found this the natural way because a real process will connect to the OPC-server that way. The communication between the Gateway and Matlab is done with the Matlab Engine, which is a set of routines that makes it possible to establish communication to Matlab from another program. We established communication between Matlab and Simulink by implementing two functions in Simulink (S-functions), one to read data from Matlab and one to write data to Matlab.

The communication between Advant Control Builder and Matlab/Simulink will make it possible to simulate control systems before they are set up in an actual plant. It is desirable to simulate a control system e.g. because it could show some unexpected behavior and it could be possible to tune the control system before it is set up on to the real process. Another possibility with simulation is to optimize already existing plants. Hazard situations of the control system can also be avoided. Simulation often saves time and money.

## 1.1 Thesis Outline

The second chapter will further discuss the reasons why it is desirable to use simulation before a control system is set up in an actual plant. In the third chapter we will describe the different parts involved in the communication between Advant Control Builder and Simulink. The forth chapter describes the different types of communication that is needed for the parts to communicate with each other. In chapter five we explain how a user will see the communication. The sixth chapter describes the functions used for the communication between Simulink and Matlab. In the seventh chapter the functionality and the implementation of the Gateway program are described. We have also investigated the performance of our implementations and the obtained results will be presented in chapter eight. Chapter nine describes how a new system is created in Simulink. The tenth chapter consists of a User Manual for the Gateway. In chapter eleven some conclusions are drawn and in chapter twelve we will describe future possibilities and extensions of our master thesis.

# 2 Reasons for Simulation

The possibility to simulate a control system gives a lot of advantages. For the purpose of ABB it can also be a benefit to be able to use a well-known simulation tool as Simulink.

There are a lot of benefits with simulations and some of them are:

- Possibility to test a control system and parts of control systems during the development phase. This can be to a big advantage if an error is detected. Often is it easier and less expensive to correct errors if they are detected in an early phase of the development.

- A lot of behaviors of a completely implemented control system can be tested on a process model to see if the control system behaves as expected. Some situations that not could be tested on the actual process can be tested on the model to see if the control system works properly.

- Simulations can give the possibility to show customers how part solutions will behave. This can justify choices of solutions for the control system. Optimization of an existing control system can also be shown to customers.

- Simulation also gives the opportunity to get an idea about the values of a controller's parameters. This can be useful when the controller is installed and connected to a real process.

- At last simulations can give the possibility to introduce and train operators on the control system. The operators can also train hazard situation on the model that never can be possible on the actual plant.

# 3 Communication Structure

The communication between Advant Control Builder and Simulink is divided into communication between different blocks. We found it appropriate to define the following blocks, Advant Control Builder, OPC-server, Gateway, Matlab and Simulink, see figure 3.1. The communication between the blocks is mainly built on COM (Component Object Model) and OPC (OLE for Process Control). We have implemented a Gateway that enables information exchange between the OPC server and Matlab. To deal with the communication between Matlab and Simulink we have implemented functions for information exchange in Simulink. Below follows a detailed description of the different blocks.

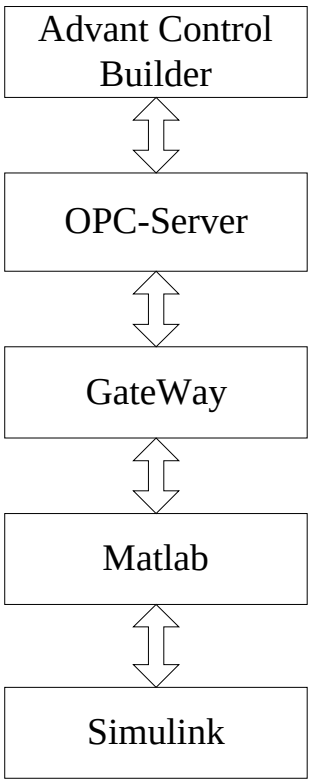


**Figure 3.1** Block-scheme of the communication.

The versions of the programs that is used throughout the thesis are:

- Advant Control Builder α-3
- AdvaControl OPC Data Access Server α-3
- Matlab 5.3.0
- Simulink 3.0

The computers that are used is one Pentium 2, 200 MHz and one Pentium 2, 166 MHz, both with a Windows NT 4.0 platform.

## 3.1 Advant Control Builder

Advant Control Builder is a real-time distributed control system. It includes control units and operator stations that are connected via a network. Advant Control Builder is a fully integrated Windows application for configuration and programming of the ABB products Advant Controller 210, Advant Controller 250, Advant Controller 800 and Advant Soft Controller [4].

Advant Control Builder uses the five programming languages Instruction List (IL), Structured Text (ST), Sequential Function Chart (SFC), Function Block Diagram (FBD) and Ladder Diagram (LD) according to the standard IEC 61131-3. Variables and data types can be boolean, integers, floating point numbers, strings, time, date, etc. all according to IEC 61131-3. A program written in Advant Control Builder is called an application. Applications are built from modules containing variables, code and graphics. An application in Advant Control Builder is build in a project. A project handles one application, which is divided in at the most three programs, fast, normal and slow. The hierarchy in Advant Control Builder is shown in figure 3.2.

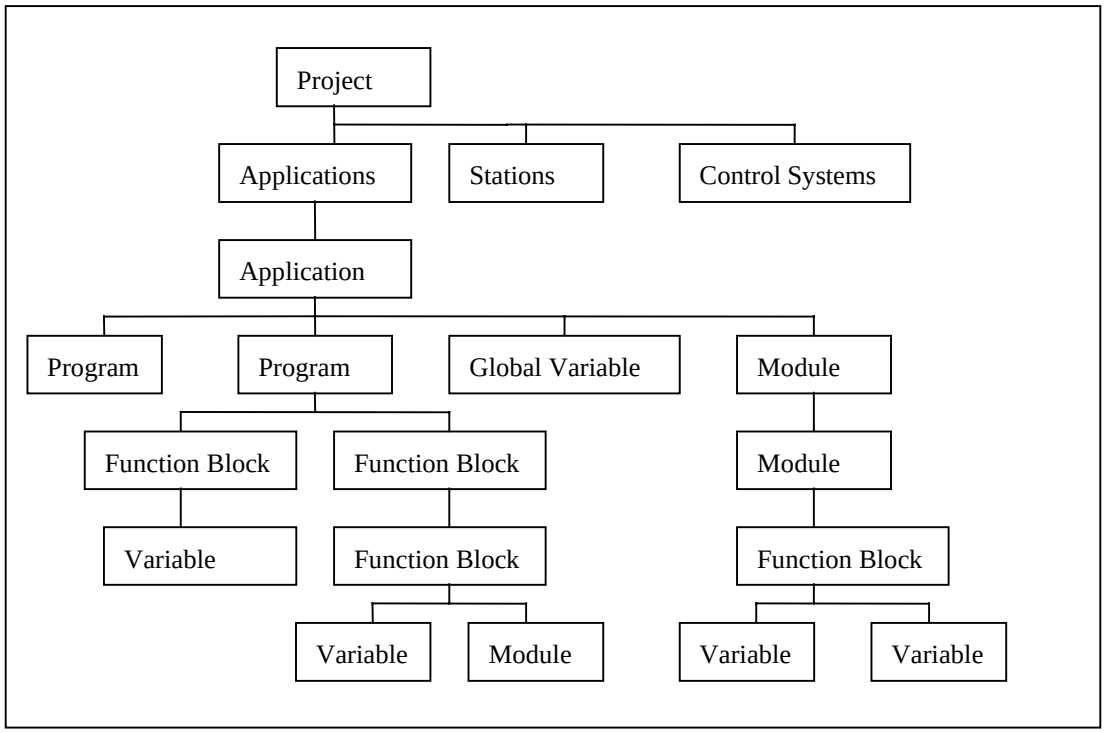


**Figure 3.2** The hierarchy in Advant Control Builder.

A Project Explorer is included in Advant Control Builder. This makes it possible to navigate, create and modify a project. When using the Project Explorer all objects, functions, function block types and module types can be selected and displayed. The controller software and hardware are also configured in the Project Explorer.

In the Advant Control Builder one library is included, this is the System library. Several other libraries can be added e.g. Logic function library, Communication library, Control libraries and Alarm library. The libraries contain data types, functions, function blocks and modules that can be used in a project. It is also possible for the user to make new libraries and add them to the Advant Control Builder.

## 3.2 Advant Soft Controller

Advant Soft Controller is a real-time software technology that runs in a PC. With the Advant Soft Controller the PC becomes a process controller. The Advant Control Builder is used to program the Advant Soft Controller. The communication between these two is achieved via an Advant MMS (Manufacturing Message Specification) server on Ethernet or MMS on a serial port.

The Advant Soft Controller and the Advant Control Builder can be placed in the same PC or in two different PCs. If they are placed in different PCs the communication between the two PCs is handled using an Ethernet network or a serial COM port.

## 3.3 COM

COM (Component Object Model) is a technology developed by Microsoft Corporation. The main idea behind the development was to enable reusability of software components. An application that supports the COM standard makes it possible for other developers to use the application, or parts of it, in their own application. The standard also enables different software components to connect to and communicate with each other. COM is based on server-client interaction through sets of functions called interfaces.

COM is a binary standard, which means that the server and the client can be implemented in different languages e.g. C/C++, Java or Visual Basic. This is the greatest advantage with COM, since the developers do not have to know what language the server is implemented in when implementing a client. All they have to know is which interfaces the server exposes.

An extension of COM is Distributed COM (DCOM) which makes it possible for the server to run on another computer connected to a client through a network.

Most of the information in this chapter is taken from “Inside COM” [1].

### 3.3.1 COM Objects and Interfaces

COM is based on objects. To clarify, in COM an object is some piece of compiled code, and is not to be confused with objects that are used in object-orientated programming. COM objects are well encapsulated. This means that you cannot get access to the internal implementation of the object. The only way to access an object is through an interface. The objects are contained by components.

An interface is a description of a collection of methods and can be seen as a contract between a component and a client. The client can communicate with the COM objects only through an interface. It is important to realize that the interface does not contain any implementation at all. The functionality of the methods is implemented in a COM object. The interface is defined in a language called IDL (Interface Definition Language). It is important to observe that it is only a definition, the implementation of the methods can be made in any language. This IDL definition is what makes the standard binary.

All COM objects and interfaces must have a unique identification number, GUID (Globally Unique Identifier). The GUID is a 128-bit number and its purpose is to make sure that there is only one interface in the world with this specific ID. To generate a GUID you can run either of the programs Guidgen.exe or Uuidgen.exe. For a COM object class the GUID is called CLSID (Class Identifier), and for an interface it is called IID (Interface Identifier). These IDs are stored in the Windows registry and are used by the client when it connects to the object.

If a client would like to use one of the methods provided by an interface, it has to contact the server. The server, who contains the implementation of the methods, then constructs a COM object containing the methods of the interface. The client is then given an interface pointer to the object and can thereby access it. More than one client can access the same interface at the same time, if this is the case each client gets its own object. When the client does not want to use the object anymore, the object is released and the server removes it.

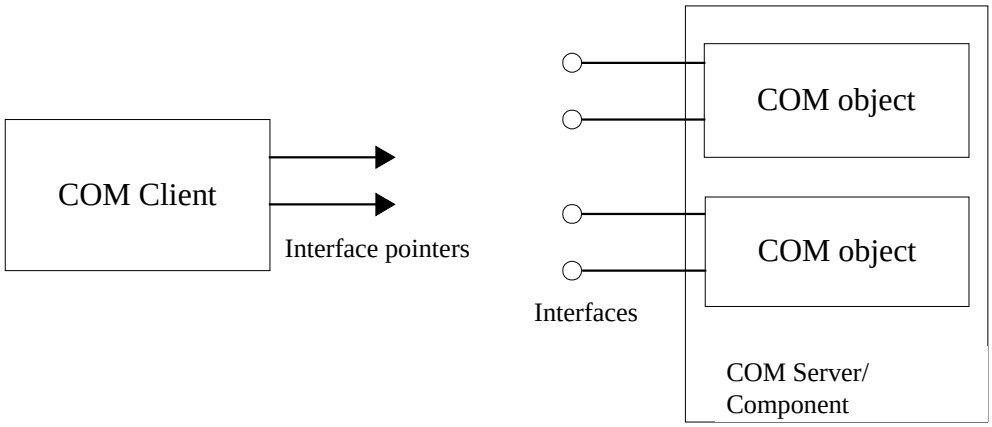


**Figure 3.3** COM client with interface pointers, and COM server with objects and interfaces.

A COM server can contain many objects where each object exposes several interfaces. Often related methods are grouped together in one interface.

Once an interface is defined and its contract is published to other users, it is not allowed to change the functionality of its methods. The reason for this is that other components are depending on the contract and would not work properly. It is allowed to improve the internal functionality as long as the contract is followed. If a change is made to an interface, it has to be given another name. This makes it possible for new clients to use the new modified interface, and the older clients can still use the old unchanged interface.

One interface, IUnknown, is common to all COM objects. The IUnknown interface contains three methods:

- QueryInterface
- AddRef
- Release

The first method, QueryInterface, is used by a client to get access to one of the interfaces exposed by the object. It takes an IID as in parameter and returns an interface pointer to the corresponding interface. AddRef is used to increase the number of used COM objects, and Release is used to decrease the number of used COM objects.

### 3.3.2 Marshaling

Since a client and a component run in different processes they also have different address spaces. When a client calls methods in a component its parameters have to be passed from the address space of the client to the address space of the component. This is called marshaling. Instead of direct communication between the client and the component, the client communicates with a DLL that acts like the component. This DLL is called a *proxy*. It is used to pack and transform the method call, and to do a RPC (Remote Procedure Call). On the server side another DLL, called *stub*, is used to unpack the information and to call the requested method. The stub also marshals the return values from the method by doing a RPC to the proxy.

This marshaling method makes it possible for a client to call a method in another process, on the same or on a different computer, the same way it calls a method in its own process.

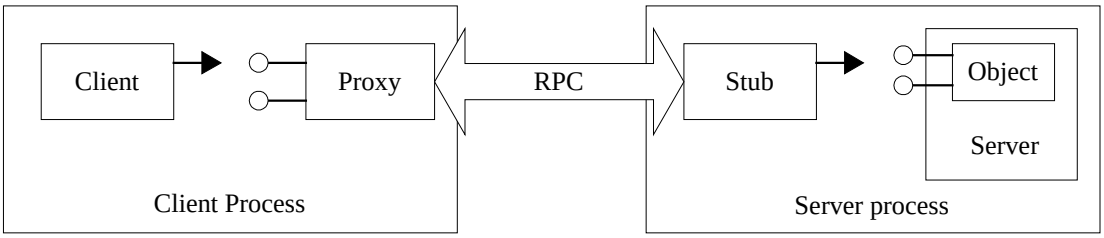


**Figure 3.4** Communication between a client and a server through a proxy and a stub.

## 3.4 OPC

OLE for Process Control (OPC) is a new standard for communicating with different data sources, e.g. devices or controllers. The goal for the development is to standardize communication between different parts of a control system and its surroundings. The OPC standard specifies an interface for communication between client applications and servers. All OPC servers contain the same basic interfaces. This means that any OPC client should be able to communicate with any OPC server. OPC is built on COM, and this makes it possible for clients and servers from different vendors to be used together.

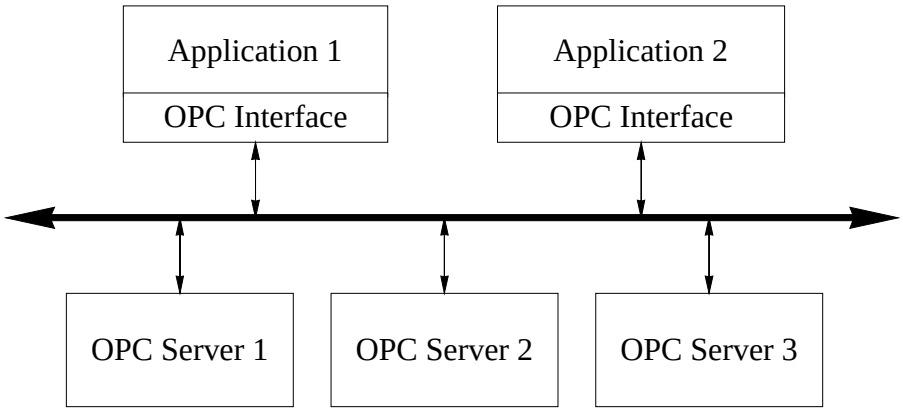


**Figure 3.5** Applications working with several OPC Servers.

### 3.4.1 OPC Server Architecture

The OPC specifications describe three different OPC servers, where each of the servers can work on their own.

- Data Access
- Alarms and Events
- Historical Data Access

OPC Data Access [2,3] defines interfaces for reading and writing information to and from the control system and it makes it possible to browse the system for information. Alarm and Events contains functions for handling alarms and events in the system. An alarm is an abnormal condition in the system and an event is some kind of system change or system error. Historical Data Access is used for handling historical process data, e.g. saving process states in a database.

The OPC standard specifies two sets of interfaces for communication between a client application and a server; Custom interfaces and Automation interfaces. All interfaces specified by OPC are COM interfaces.

The Custom interface is designed for high throughput applications and is used by applications developed in C++. The Automation interface is provided primarily for Visual Basic applications.

### 3.4.2 OPC Data Access Server

#### 3.4.2.1 Architecture

The OPC Data Access server is the part that has been used for the communication and is therefore the one that will be described.

The standard defines three COM objects that all OPC servers must support, OPC Server, OPC Group and OPC Item. These objects are arranged in three levels. At the top level is the OPC Server object, which is the object that the client first connects to. It keeps information about the server and serves as a container for OPC Group objects.

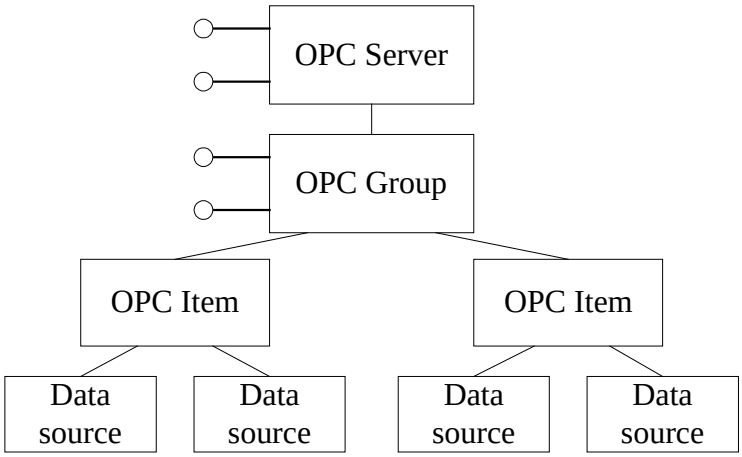


**Figure 3.6** OPC object structure and relation to data sources in controllers.

The OPC Group object provides a way for the clients to organize data. For example, a client may want to group all process temperature values into one group and all process air pressure values into another group. There are two types of groups, public and local. A public group can be accessed by any client while a local group only can be accessed by the client that created it. All groups are local at creation, but can then be publicized by the client.

The OPC item object represents a connection to a data source within the server. This is where to get the actual value received from the process control device. The items do not have any custom interfaces. Access to an OPC item has to go via interfaces for the OPC Group that contains the item.

Associated with each item is a value, of type VARIANT, quality and a time stamp. The client specifies the item to the server through an item ID. The item ID is server specific and it uniquely identifies to the server how to locate the value in the process control device.

It should be noted that the items are not the data sources, they are just connections to them. The OPC Item should be thought of as specifying the address to the data and not as the actual physical source of the data.

#### 3.4.2.2 Functionality

There are three ways to access data in a Data Access server:

- Synchronous read and write
- Asynchronous read and write
- Subscription

The data to be read is from the cache or from the device. The data in the cache represent the latest value of the data value, the quality and the time stamp. The time stamp gives the time when the value was obtained from the device.

## 3.5 Matlab/Simulink

The name Matlab stands for matrix laboratory. Matlab is a language for technical computing. It performs large numerical calculations and matrix operations. Matlab basic data type is an array and it does not require dimensioning. This makes matrix and array calculation quite easy. There are a lot of toolboxes that can be used together with Matlab. One of them is Simulink, which is used for simulation of systems built on mathematical models.

### 3.5.1 Matlab

The Matlab system consists of five main parts [5]:

- The Matlab languages
- The Matlab working environment
- Handle graphics
- The Matlab mathematical function library
- The Matlab Application Program Interface.

The Matlab language is used for accessing and manipulating scalars, vectors, matrices and user-defined data structures and objects. The language that is used in Matlab workspace is the same that the one used for Matlab programs. This makes it easier to convert and reuse code from the workspace to a program. The Matlab program does not require either variable declarations or array dimensioning.

The Matlab working environment is where you work with Matlab as a user or a programmer. You use the workspace for mathematical calculations, importing and exporting data.

Handle graphics is Matlab's graphic system. It is possible to do two-dimensional and three-dimensional data visualizations, image processing and animations. It is also possible to manipulate the lines, surfaces and other graphic elements with the graphical routines. Handle graphics can be used from Matlab workspace or from M-files. It is also possible to do personalized graphic functions in the M-files.

The Matlab mathematical function library is a large collection of algorithms like sine and cosine. It also includes larger functions like matrix inverse and matrix eigenvalues. The mathematical function is expressed in M-files. Explanation about how the functions are to be used is often placed in the M-files and can be read from Matlab workspace. This is done by the command Help, followed by the name of the function.

C and Fortran subroutines can be called from Matlab as if they were built-in functions. This is done with the mex command that recompile the C or Fortran routines as a Matlab executable file. They become mex-files to be run in Matlab. One advantage with mex-files is that large pre-existing programs can be run in Matlab without rewrite them as M-files [6].

All types in Matlab like variables, including scalars, matrices, strings, cell arrays, structures, and objects are declared as a single object type: the Matlab array. The Matlab array can also be declared in a C program and the name is then mxArray. The mxArray structure includes

- Its type
- Its dimension
- The data associated with this array
- If numeric, whether the variable is real or complex
- If sparse, its indices and nonzero maximum elements
- If a structure or object, the number of fields and fieldnames

#### 3.5.1.1 Matlab Engine

With the Matlab Application Program Interface [6] the possibilities with Matlab expands. Matlab can then be called from external programs, it is possible to call Matlab from C or Fortran programs and data can be exported and imported from the Matlab environment. This makes it for example possible to use Matlab as a computation engine from other applications. The Matlab Engine library is used for interprocess communication. The engine library for C-programs consists of the functions in table 3.1.

| **Function** | **Purpose** |
|---|---|
| engOpen | Start the Matlab engine |
| engClose | Shut down the Matlab engine |
| engGetArray | Get a Matlab array from Matlab workspace |
| engPutArray | Send a Matlab array to Matlab workspace |
| engEvalString | Execute a chosen Matlab command |
| engOutputBuffer | Create a buffer to store Matlab text output |

**Table 3.1** Matlab C engine functions.

On a Windows platform Matlab uses ActiveX for the communication. Matlab Engine may be used if you want to do large mathematical calculations in your C-program or if you want to exchange data with Matlab.

The routine engOpen starts a Matlab process, establishes a connection, and returns a unique engine identifier. The engine identifier is then used by the other Matlab engine routines so that they will be addressed to the correct Matlab process. To quit a Matlab engine session the routine engClosed is used. It sends a quit command to the Matlab engine session and closes the connection.

The engGetArray routine is used to copy a variable from Matlab workspace. engGetArray reads the desired mxArray in the workspace and returns a pointer to a newly allocated mxArray structure. The routine engPutArray writes an mxArray to Matlab workspace. If the mxArray does not exist in the workspace, it is created. If an mxArray with the same name already exists in the workspace, the existing mxArray is replaced with the new mxArray.

The routine engEvalString puts a string to Matlab workspace. The string can be a mathematical expression that is to be evaluated by Matlab or a command that is to be executed in Matlab workspace. The string that is sent to Matlab is not shown on the screen, to show the string the routine engOutPutBuffer can be used.

### 3.5.2 Simulink

Simulink is a software package to be run from Matlab, and it is used for simulating, modeling and analyzing dynamical systems. It is possible to work with linear and nonlinear systems, modeled in continuous time, sampled time or a mixture of them. Simulink uses graphical user interface for building up models to be simulated [7].

A library of blocks is included in Simulink and the click-and-drag mouse operations make it possible to build complete models. The block library consists of e.g. sinks, sources, linear and nonlinear components and connectors. It is also possible to write own functions, so called S-functions [8], in C code or in Matlab code. To build up a model in Simulink you choose suitable blocks and connect them with graphical connections.

When a complete model is built it is possible to simulate it. Settings in Simulink can be changed in Simulinks menus or by entering commands in Matlab's command window. The Matlab command for change settings in Simulink is set_param [7]. With different parameters for this function it is possible to control Simulink from Matlab workspace. Examples of settings can be the simulation time and if you want an ongoing simulation you choose inf in the time window. Parameters in Simulink may also be changed during the simulation. By using scopes and other display blocks, you can see the results while running the simulation. The results can be sent to a file. After the simulation you can load the file in Matlab workspace e.g. for plotting and analyzing.

S-function (system function) is a block for extended functions in Simulink. An S-function is a computer language description of a dynamic system. The S-function is written in C or Matlab. To use the S-function in Simulink it has to be compiled as an MEX-file (Matlab Executable file). S-functions written in C are divided in three major parts, one to be executed in the initial part, one to be executed during running simulation and one to be executed in the termination part of the simulation. In the initial part of the S-function you declare start performance, number of inputs and outputs and the sample time for the block. In the next part of the S-function the runtime performance is declared. At last all terminations are implemented [8].

# 4 Communication

The goal has been to enable communication between Advant Control Builder and Simulink. The communication will be used for simulation of control systems and the processes that the system controls. The control system is created in Advant Control Builder and the process is built up in Simulink. The types of data that are sent between the two applications are number values.

Since Advant Control Builder and Matlab are two different processes they do not share memory space. The problem was to find ways for accessing variables in the two processes. There is an existing automation server in Advant Control Builder, which makes it possible for other applications to read and write variables in the system. The interfaces in Advant Control Builder are designed for Advant Control Builder and to be able to use them a client has to have deep knowledge about the functionality of the interfaces. Another way to get access to Advant Control Builder is through an OPC Data Access server. Since the OPC standard defines the interfaces in a more consistent way this seemed like a good idea.

To get access to variables in Simulink we found it reasonable to use the Matlab engine interface. The engine interface is implemented by a server in Matlab and contains routines for communication between a stand-alone program and Matlab workspace. This means that we do not really got any access to the values in Simulink. But since Simulink and Matlab are so tightly integrated it is no problem to make Simulink read data from and write data to Matlab workspace. What we did was to implement s-functions which only purposes were to access variables in Matlab workspace, one kind of s-function for reading and one for writing. This means that Simulink runs as a process by its own, reading and writing variables in Matlab workspace, independently of how and when the values were put there. This part of the communication link is described further down in chapter 6.

To enable communication between the OPC server and Matlab workspace we have implemented a Gateway. The Gateway is a separate process implemented in C++ and can be seen as a link between the OPC server and Matlab. Parts of it are implemented as a COM/OPC client and parts of it use the Matlab Engine.

## 4.1 The Communication Between the Different Blocks

The total communication between Advant Control Builder and Matlab/Simulink can be seen as communication between different blocks as described in chapter three, (see figure 4.1). Here follows a description of what types of communication that is needed between the different blocks.

### 4.1.1 Advant Control Builder – OPC Server

When an application is created in the Advant Control Builder it can be tested without being connected to an actual process. This is done in the mode simulate, a mode where it is possible to simulate the control system. Simulate can be used to test and debug the application during the development of the control system. When it is time to run the control system against a real process the control application is downloaded from the workstation to a control unit. When the application is downloaded, the controller takes control over the control algorithms and the workstation becomes an operator station. To be able to communicate with the controller from the workstation, Advant Control Builder must be in the mode online.

The OPC Data Access server is used to provide data from the controller. During the compilation of the control algorithms Advant Control Builder saves information about applications, hardware configuration and access variables to file. Those files are sent to the OPC server, which needs the files to build its internal structures.

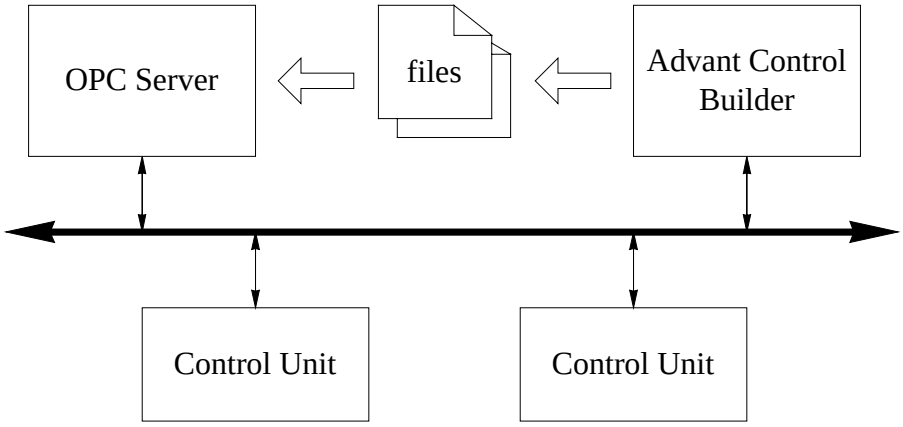


**Figure 4.1** The OPC Server needs files generated by Advant Control Builder to provide data from controllers

Once the control application is downloaded to the controller, the only thing that the OPC server need to work properly is the files generated by Advant Control Builder. The OPC server does not require that Advant Control Builder is in the mode online, or that it is even running.

It is possible to have several OPC servers on different machines connected to the same controllers, i.e. basis for redundancy.

### 4.1.2 OPC Server – Gateway

The part of the Gateway that communicates with the OPC server is implemented as an OPC/COM client.

The first thing that has to be done is that the client contacts the server. There are different ways of doing this depending on what kind of server you want to access. ABB:s OPC server is an out-of-process server and can be run as a local server or as a remote server. In both cases the server needs to be configured before usage, i.e. the server must be informed about what controllers it should provide data from.

Local means that the server and the client are on the same machine. If the server is not running when the client is started, COM will start up the server and connect the client to it. This means that the server is not connected to any controllers and needs to be configured. If the server is running when starting a client, another client may have configured the server and COM will connect the client to the running instance of it.

A remote server is located on a different machine from the client, and DCOM is used for communication. The implementation of a DCOM client is a little bit different from a COM client.

There are two ways to configure the server. One way is to use the graphical user interface that is provided with the OPC server. The user interface makes it easy to select the controllers you want the OPC DA server to provide data from. The other way is to let the client configure the

server directly through COM interfaces. This makes it possible for a specially written OPC/COM client to first use the COM interface to perform the configuration and then use the OPC interfaces to access the server. The Gateway that we implemented does not provide the opportunity to configure the OPC server, the configuration has to be done through the server's user interface.

The Gateway needs to be configured as well, this means that the user has to provide certain information that the program needs to work. The user gives this information in the first instance of the program. After that the OPC client part of the program contacts the OPC server and connects to it.

The Gateway makes it possible to connect both to local and remote servers. The user has to specify what kind of server, local or remote, to be used as a part of the configuration. This is necessary for the program to know since the methods for connecting by COM and DCOM are different.

### 4.1.3 Gateway – Matlab

To get access to Matlab we found it reasonable to use the Matlab Engine interface [6]. On Microsoft Windows, Matlab engine programs communicate with a separate Matlab process via ActiveX. The engine interface is implemented by a server in Matlab and contains methods for communication between a stand-alone program and Matlab workspace. The methods provided by the interface allow you to start and end the Matlab process, send data to and from Matlab, and send commands to be processed in Matlab.

According to Mathworks, the company behind Matlab, the engine routines are to be used by stand-alone programs implemented in C or Fortran. We found that the routines worked well for C++ programs also. This suited us well since we wanted to implement the OPC client part of the Gateway in C++.

### 4.1.4 Matlab – Simulink

The communication between Simulink and Matlab is handled with two S-functions, one to get values from Matlab workspace and one to put values to the workspace. The S-functions are named inputblock1 and outputblock1 with increasing numbers depending on how many connections the user wants to have. The S-functions are written in C.

The S-functions have to be compiled with the mex command to be able to run in Simulink. The mex command makes a dll-file of the c-file. That file could then be declared in an S-function block in Simulink without extension. These blocks are at the beginning drawn with one in- and one output. When Simulink gets the name of the S-function it redraws the S-function block to fit the predefinitions.

The regular user of Simulink may wonder why the predefined Simulink-block to-workspace and from-workspace could not be used. That is because this blocks save the value to be sent in an internal array and after the simulation has stopped it puts the array to the workspace. This is not what we wanted to do because we needed the values continuously.

# 5 How the User Sees the Communication

The main goal with the communication between the Advant Control Builder and Matlab/Simulink is to build a control loop in the Advant Control Builder and a model of the process to be controlled in Simulink.

An example of how a working Simulink and Advant Control Builder system looks like is shown in figure 5.1. It is a simulation of a system with two tanks on top of each other. The outflow from the upper tank flows into the lower tank. The level in the lower tank should be controlled from the inflow in the upper tank. The controllers are placed in Advant Control Builder and the models of the tanks are placed in Simulink. The tanks are made as S-functions in Simulink, see Appendix A for the C code of the S-function. The level from both tanks can be shown in a graph in Simulink. The graph becomes visible by a double click on the scope icon in the simulation window.

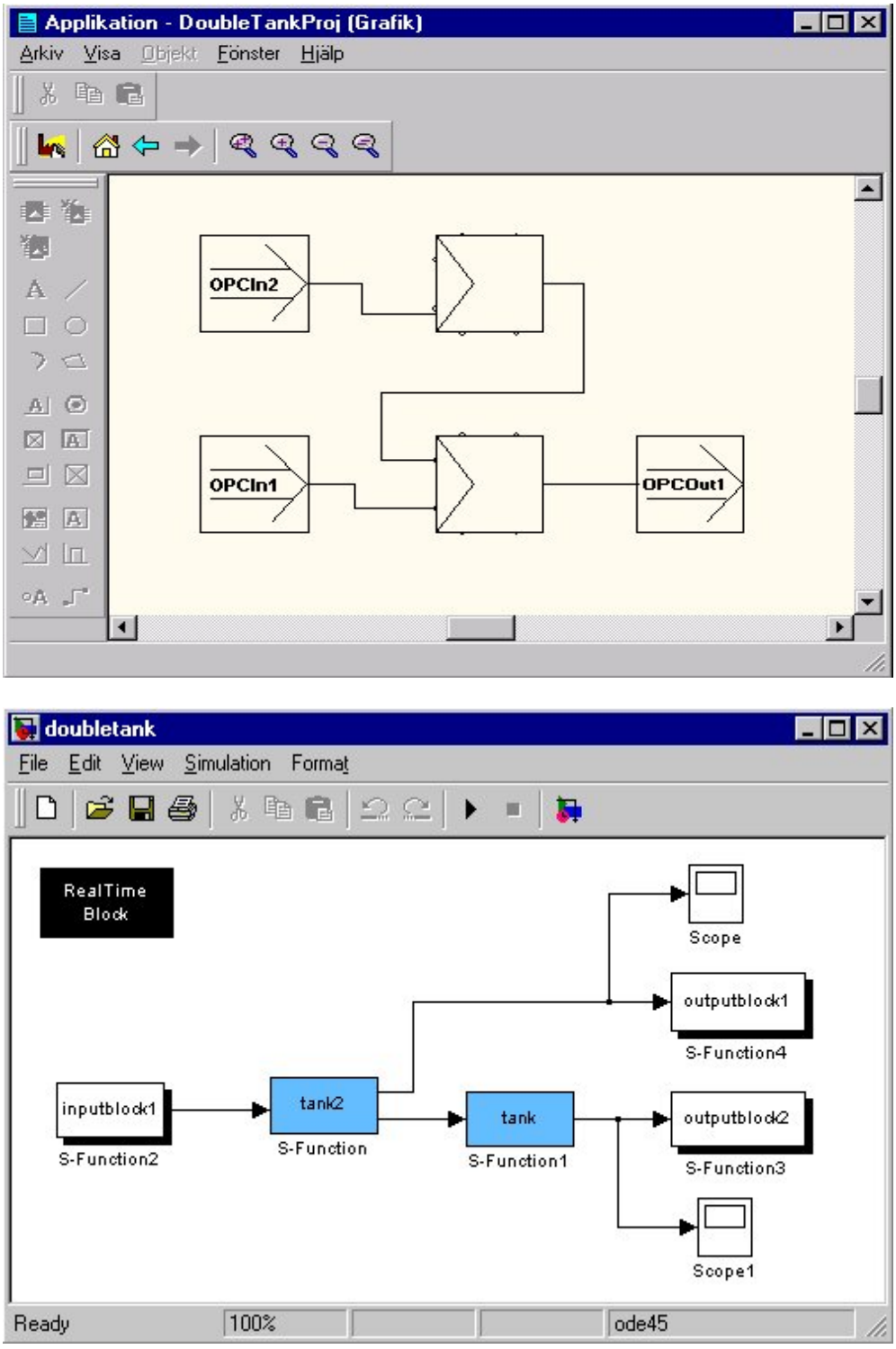


**Figure 5.1** Example of a complete simulation system. The upper window is from Advant Control Builder and the lower one from Simulink.

In the Advant Control Builder should mainly the controllers be placed. The values from the process in Simulink are taken from the OPC server to an OPC-In block. The values from the controller are connected to the OPC-Out block and then sent to Simulink via the OPC server. The OPC-In and OPC-Out block are implemented only for this purpose. Five OPC-In and five

OPC-Out blocks have been implemented, they are named OPCIn1 to OPCIn5 and OPCOut1 to OPCOut5.

By clicking on the OPCIn- or OPCOutblocks in the Advant Control Builder the window in figure 5.2 will be visible. This window shows the actual values that are connected to the OPC-In and OPC-Out blocks. A graph with history values and a pile diagram of the actual value can also be shown from the icons in the upper right corner in that window.

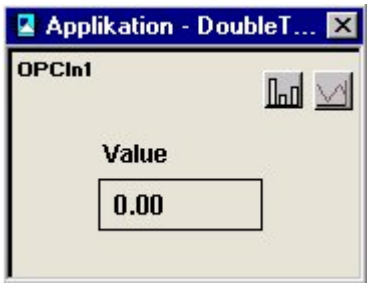


**Figure 5.2** Window from the in- and outputs-block in the Advant Control Builder.

A model of the process to be simulated should be placed in a Simulink system. The values to the process model are brought to Simulink from the OPC server via an inputblock in Simulink, and the values from Simulink to the OPC server are sent via an outputblock in Simulink. The input- and outputblocks are S-functions implemented in C. They are named inputblock1–inputblock5 and outputblock1–outpublock5.

The same numbered OPC-Out block and inputblock makes a connection between Advant Control Builder and Simulink. A connection like this is used to send values from Advant Control Builder to Simulink i.e. OPCOut1 is connected to inputblock1. Also the same numbered outputblock in Simulink and OPC-In block in Advant Control Builder makes a connection from Simulink to the Advant Control Builder, i.e. outputblock1 is connected to OPCIn1. This is necessary for a value to be addressed to the right place in both directions.

# 6 Communication S-Functions

To be able to use a Simulink system together with the communication there are three different kinds of S-functions that have to be used beside the process model. These S-functions are:

- inputblock
- outputblock
- realtimer

The S-functions are implemented in C. Their main implementation follows the standard framework of different sub functions for an S-function. The functions used are listed in table 6.1 and the calling sequence is shown in figure 6.1.

| Function | Purpose |
|---|---|
| mdlInitializeSizes | Define the number of input and output ports, and the sizes of the ports |
| mInitializeSampleTimes | Define the sample time of the S-function |
| mdlStart | Used for one time initialization |
| mdlOutputs | Calculation of outputs |
| mdlUpdate | Perform tasks that only take place once per integration step |
| mdlTerminate | Perform tasks at end of simulation |

**Table 6.1** Functions in the S-functions.

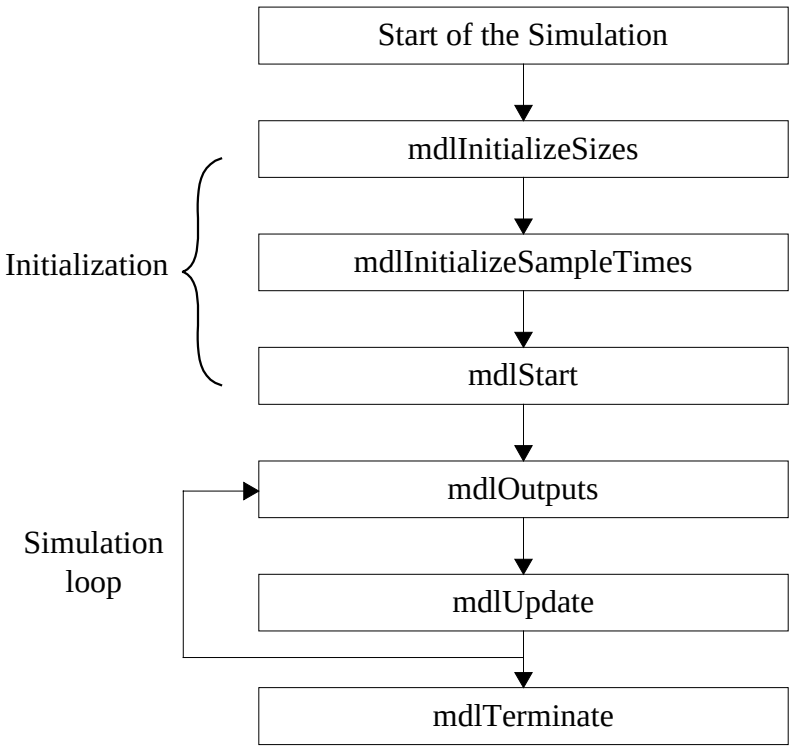


**Figure 6.1** The calling sequence for the S-Functions

Before the sub functions in the S-functions some definitions have to be made. One important definition is the name of the S-function that later is to be written in the S-function block in Simulink [8].

## 6.1 Inputblock and Outputblock

The purpose of the S-function inputblock is simply to get a variable from Matlab workspace and put it to the block's output port. This variable contains the value that is sent from the Gateway to Matlab workspace. The purpose of outputblock is to pass the variable from its input

port to Matlab workspace. This variable is then fetched from Matlab workspace by the Gateway.

In the mdlInitializeSize function the number of function-parameters, the number of inputs and the number of outputs are declared. In the inputblocks there is one function parameter, no inputs and one output. The function parameter is the sample time for the S-function, which is provided by the user. The outputblocks also need the sample time parameter, but one input and no output.

The mdlInitializeSampleTimes function is to declare the sample time for the S-function. This is the sample time specified by the user.

The mdlStart function is used for initialization code in the S-function. We only used mdlStart in the outputblock. In mdlStart the variables that are to be sent from Simulink to Matlab workspace are declared. They are named Fromsim1−Fromsim5. Fromsim1 belongs to outputblock1 and so on. The creation of Fromsim1−Fromsim5 is done by the command mxCreateDoubleMatrix [9].

In the function mdlOutputs the code to be running during simulation is placed. In the inputblocks the array ToSim is collected from Matlab workspace by the command mexGetArray [9]. Then depending on which block it is, one value is picked from the right place in the array. For example in inputblock1 the first element in the array is collected. Then at last this value is put to the output of the S-function. In the outputblocks the value from the simulation is collected and then copied to the parameters Fromsim1-Fromsim5 depending on which S-function it is. The parameters are then put to Matlab workspace by the command mexPutArray [9].

The function mdlTerminate contains termination code used before the S-function is shot down. This function is only used in the outputblocks. The function is used to destroy the variables Fromsim1−Fromsim5. The command used for this is mxDestroyArray [9].

## 6.2 Realtimer

The sample time specified in an S-function sets the sample time at which a discrete block's states are updated. In a Simulink model it is possible to have blocks with different sample times, multirate systems. In a multirate system the different block's states are updated at different times.

Simulations in Simulink are optimized to go as fast as possible and the sample times are only used to keep track of the different update rates of the block's, thus the specified sample time has nothing to do with the real time. This cause problems if it is desired to run the simulation in real time. The toolbox Simulink does not provide any possibilities to simulate a model in real time. One possible way to solve this problem is to use the toolboxes Real-Time Workshop and Real-Time Windows.

To solve the real time problem we have implemented an S-function that slows down the speed of the simulation, this S-function is called realtimer. The realtimer block has two purposes, to let the S-function loop run once for every sample period, and to show the jitter, i.e. the lag in the loop.

To slow down the simulation loop we have used two Windows functions, GetCurrentTime, which gives the time of the day in milliseconds and Sleep which suspends the execution of the

current thread for a specified interval. The user provides the desired sample time of the function.

In mdlStart the current time is measured. When it is time for mdlUpdate to execute, a new time is measured. The difference between the new and the old time is determined, and this gives the execution time, i.e. how long time that has gone so far. Then the difference between the sample time and the execution time gives the wait time, i.e. the time that the function should sleep. When the thread wakes up mdlOutputs is executed.

The jitter is calculated in mdlOutputs and put to the output port of the realtimer. The jitter is calculated as: 1-( wait time )/(sample time). This is a value that lies between zero and one, where a low value indicates good sampling accuracy. Values close to one indicates that the sampling period should be increased.

Since Matlab workspace and Simulink belong to the same thread, Matlab workspace is also suspended when realtimer goes into the Sleep function. This causes a problem when the Gateway wants to write to or read from Matlab workspace. When Sleep is executed the communication between the Gateway and Matlab is blocked, and this makes the communication to be slower than we want to. This problem is further discussed in chapter 7.5.

# 7 Gateway

The task of the Gateway can be divided into three parts:

- Access of the OPC server
- Access of Matlab workspace
- Provide a user interface

The Gateway needs to be configured, this means that the user has to provide certain information that the program needs to be able to work. The user gives this information in the first instance of the program. After that the OPC client part of the program contacts the OPC server and connects to it. When the client has got access to all the OPC interfaces that it needs the program establishes a contact with Matlab, Simulink starts and the Gateway is ready to begin the communication. When the communication is started the program goes into a loop where it sends data between the OPC server and Matlab.

## 7.1 Access of the OPC Server

Access of the server includes a number of tasks that have to be fulfilled. The client must be able to establish contact with the server. It also has to provide methods to get access to all the different interfaces that the server contains. The interfaces are then used to exchange information between the client and the server. For example the interfaces are used to build the structure of the server, e.g. groups and items. The interfaces are also used for things like browsing the server for information, reading data and writing data. The OPC Data Access server provides a large number of interfaces, but since the main task of the Gateway's client-server interaction is to exchange number values, only some of the interfaces are used.

We have written functions that contain the necessary implementation for communication between the Gateway and the OPC server. Table 7.1 lists all the functions and their purposes.

| Function | Purpose |
|---|---|
| InitOPC | Establish contact between a client and a local OPC server |
| InitOPCRemote | Establish contact between a client and a remote OPC server |
| CreateGroup | Add a group to the structure of the OPC server |
| DefineItems | Define one or more items to be added to a group |
| CreateItem | Add an item to a group |
| ReadOPC | Read from a data source associated with an item |
| WriteOPC | Write to a data source associated with an item |
| KillItem | Remove an item from a group |
| KillGroup | Remove a group from the OPC server |
| CloseOPC | Disconnect the client from an OPC server |

**Table 7.1** Gateway OPC functions.

### 7.1.1 InitOPC and InitOPCRemote

A process that wants to use COM must call the function CoInitialize [1] to initialize the COM library. After a call to CoInitialize the process is allowed to use any of the library's methods. Since this has to be done both for COM and DCOM, CoInitialize is called in InitOPC as well as in InitOPCRemote. When the process is finished with the COM library, it must call CoUninitialize [1]. This is done in the function CloseOPC. The COM library can only be initialized once per process.

If the server and the client are on the same machine, the security level of the information exchange between them is high. To get the same security level for a distributed system the client can call CoInitializeSecurity [1]. This call is made in InitOPCRemote.

A function in a component often allocates a block of memory. This block of memory is returned via an out parameter to the client. Since the client and the component can be written in different languages and be running in different processes, problems may occur when the client wants to free the memory. To avoid this problem COM's task memory allocator is used. By using the task memory allocator a component can provide the client with a block of memory that the client can delete.

To get access to the task memory allocator, InitOPC and InitOPCRemote call the function CoGetMalloc [1]. The task memory allocator is then used via an interface pointer returned by CoGetMalloc. This interface contains a method that makes it possible for a client to free memory allocated by a component.

To initialize contact to the OPC server object, InitOPC uses the COM method CoGetClassObject [1]. This method takes a CLSID and returns a pointer to an interface belonging to the class factory for that CLSID. The interface contains a method called

CreateInstance [1], which is used by the client to get an interface pointer to the OPC server objects IUnknown interface. Through this IUnknown interface the client gets a pointer to the IOPCServer [2] interface, which is the main interface of the OPC server. IOPCServer makes it possible for the client to get access to all the interfaces that the Gateway needs to be able to communicate with the server, i.e. group interfaces and item interfaces. To get a pointer to an interface the COM function QueryInterface [1] is used. QueryInterface is called every time the Gateway program wants to get access to a new interface, together with an IID for the desired interface.

In InitOPCRemote a call to CoCreateInstanceEx [1] is made instead for CreateInstance. This makes it possible for the client to get a pointer to a remote server's IUnknown interface.

### 7.1.2 CreateGroup

To add a group to the server, the IOPCServer method AddGroup [2] is used. AddGroup returns a pointer to the OPCGroup object's IUnknown interface. Through IUnknown it is possible for the client to gain access to all the interfaces that the group object exposes.

A call to AddGroup makes it possible for the client to set a number of parameters, such as the name of the group and the requested update rate. The update rate is the fastest rate at which a value associated with an item in this group is updated. The Gateway sets the update rate parameter to zero, which means that the server should use the fastest practical rate.

### 7.1.3 DefineItems

Before an item can be added to a group, the item has to be defined. The definition of an item is an array with variables of the type OPCITEMDEF [2]. These OPCITEMDEF:s tell the server everything it needs to know about the item including the access path, definition and requested data type.

Since the items represent the data sources in the downloaded Advant Control Builder project, one of the important things that the server needs to know is where to find the data source. The ItemID [2] defines the location of this data source. The ItemID is one of the OPCITEMDEF:s and it consists of a string that uniquely identifies an OPC data item. The syntax of the ItemID is server specific. For the OPC server that we have used the ItemID can look something like "Applications.SimulinkProj.SimuLinkMod.OPCOut1.ValueOut". This is the access path to a variable in the Advant Control Builder where SimulinkProj is the application, SimuLinkMod is the module and OPCOut1 is the function block. The last string, ValueOut, is the name of the variable that represents the data source. This is the variable that the client gets access to when it adds an item to a group.

Another OPCITEMDEF is the data type requested by the client. The Gateway program requests real values when it defines the items, but it is possible for a client to request other data types, such as integers, strings, words and boolean variables.

Ten items are defined in DefineItems. These items represent the values associated with the ten function blocks OPCOut1-OPCOut5 and OPCIn1-OPCIn5. For each item definition a call to CreateItem is made. CreateItem returns a handle for the defined item.

### 7.1.4 CreateItem

This function has two purposes, to determine if an item defined in DefineItem is valid (could it be added without error), and if it is, add the item to a group. An item is valid if the user of the Advant Control Builder have used the function block whose value is associated with that item, i.e. if the function blocks OPCOut1 and OPCIn1 are used, only the corresponding items are added.

To validate the item, the OPCGroup interface IOPCItemMgt [2] is used. The interface contains a method, ValidateItems, which takes an OPCITEMDEF and returns a result that tells if the item was successfully validated.

If the item is valid, the IOPCItemMgt method AddItems is called. AddItems takes an OPCITEMDEF and adds the defined item to the group. When the item is added, the server got all the information it needs about the data that are to be exchanged between the client and the server.

CreateItem takes an itemID as a parameter, and if the item defined by that itemID is added it returns an OPCHANDLE for the item. The OPCHANDLE is then used to refer to this item. The handles of the added items are put into arrays, one array for OPCOut items and one for OPCIn items. These arrays are in the Gateway called OPCOutItems and OPCInItems.

### 7.1.5 ReadOPC

The OPC interface IOPCSyncIO [2] allows a client to perform synchronous read and write operations to the server. ReadOPC calls the IOPCSyncIO method Read. This method reads the value for one or more items in a group. The data can be read from cache in which case the cache is updated by the rate that was set for the group in CreateGroup, or it can be read from device. The Gateway program reads from cache.

When calling Read, the client has to tell if it wants to read from cache or device, how many items to be read, and the names of the OPCHANDLES for the items to be read. Read then returns the values represented by the items.

In ReadOPC the method Read is called together with the OPCHANDLE array OPCOutItems and the number of added OPCOut items. The data source values returned by Read are put into an array that later on is used in a function called WriteMatlab.

ReadOPC takes a pointer to an array of OPCHANDLES and returns an array of values associated with the items that are read. This means that by calling ReadOPC, all the OPCOut blocks in the Advant Control Builder can be read in one single operation.

### 7.1.6 WriteOPC

This function calls the IOPCSyncIO method Write to write values to the items in the group. The values are automatically written to devise, and Write does not return until it has verified that the device has actually accepted the data.

Write takes the values, the number of items and the names of the OPCHANDLES for the items to be written. It returns a result that indicates success of the item writes.

In WriteOPC the method write is called together with the OPCHANDLE array OPCInItems, the number of added OPCIn items and the values that are to be written.

WriteOPC takes a pointer to an array of the values to be written and a pointer to an array of OPCHANDLES. This means that by calling WriteOPC, all the OPCIn blocks in the Advant Control Builder can be written in one operation.

### 7.1.7 KillItems and KillGroup

KillItems calls the method RemoveItems of the IOPCItemMgt interface to remove the items from the group. This is done when data exchange between Advant Control Builder and Simulink is stopped, e.g. when the user wants to reconfigure the Gateway program.

KillGroup releases the pointers that points to the interfaces exposed by the OPC Group. After that it calls the method RemoveGroup of the IOPCServer interface, which results in the group being deleted.

### 7.1.8 CloseOPC

To disconnect the client from the server, all the used interfaces have to be released. In CloseOPC the pointer to the IOPCServer interface is released. After that the pointer to the server's IUnknown interface is released. The memory allocated by the IOPCSyncIO method Read is set free by the task memory allocator and the pointer to the memory allocator is released. Since the COM library no longer is needed, CloseOPC ends with a call to CoUninitialize.

## 7.2 Access of Matlab Workspace

As mentioned above the Gateway uses the Matlab engine routines for communication with Matlab. A program that uses Matlab engine needs to have access to the Matlab files libeng.def, libmat.def, and libmx.def. This is done either by linking against the files in their original Matlab directory or by importing them to your program project directory, and link against them there. The header file engine.h, provided by Matlab, also has to be included in the program.

For our purpose Matlab workspace only serves as a temporary storage for the values to be sent to and brought from Simulink and the Gateway. Since Simulink is a toolbox to Matlab it is necessary to run Matlab to be able to simulate in Simulink. Matlab is started externally by the Gateway program. That is done with the use of Matlab Engine. It is also possible to bring up the right Simulink system directly from the Gateway. When the simulation is finished the Gateway terminates Matlab.

We have implemented functions that deal with the communication between the Gateway and Matlab workspace. Table 7.2 lists the functions and their purposes.

| Function | Purpose |
|---|---|
| InitMatlab | Starts Matlab and initialize variables to Matlab workspace |
| WriteMatlab | Writes data to Matlab workspace |
| ReadMatlab | Reads data from Matlab workspace |
| CloseMatlab | Deletes the Matlab variables and closes Matlab |

**Table 7.2** Gateway Matlab functions.

### 7.2.1 InitMatlab

InitMatlab is the function to do initializations to be sent to Matlab, start Matlab and start the right Simulinksystem.

To be able to use engEvalString [6] to change directory and open the Simulinksystem in Matlab we have to make a compatible string to the command engEvalString. We add cd to the path that the user enters to change to the right directory and open for the entered Simulinksystem.

Matlab then starts from InitMatlab. This is done with the engine routine engOpen [6]. The Gateway uses predefined variables in Matlab to send and collect data that the variables have. This variable has to be defined in InitMatlab and then sent to Matlab. To be able to put the variables to Matlab they have to be placed in a Matlab compatible array. These arrays are created with the command mxCreateDoubleMatrix [9]. The variables are initialized to zero so that the simulation always starts at a well-defined value. It is also necessary that the values exist in Matlab workspace before the system starts in Simulink. If the value does not exist then the output- and input-block would not have values to send and collect data from with the consequence that Matlab will crash.

At last InitMatlab put the started Matlab in the right directory, send the value and bring up the Simulink system. This is performed with the engine routine engEvalString and engPutArray [6].

### 7.2.2 WriteMatlab

WriteMatlab copies the array with the values that are read from the OPC-server to a Matlab convertible array. Memcpy is the routine used for the operation. This array is then sent to Matlab workspace with the command engPutArray [6].

### 7.2.3 ReadMatlab

ReadMatlab first copies the single values, in Matlab workspace, from the different outputblocks in Simulink to an array. This array is then brought to the Gateway with the routine engGetArray [6]. This array can now be sent further to the OPC-server. The ReadMatlab function returns a pointer to the array from Matlab to the main program. The return is done with the mxGetPr [9] command.

### 7.2.4 CloseMatlab

CloseMatlab first deletes the parameters used to send and collect data-values to and from Matlab. The function mxDestroyArray deletes the values and sets the memory that the parameters occupied free. At last CloseMatlab closes the Simulink system and the Matlab command window. Matlab is shut down with the engine routine engClose.

## 7.3 The User Interface

To make it easy to use the Gateway we have implemented a user interface. The user interface consists of a main window and a couple of dialog windows. The main window is visible all the time when the Gateway is running. The small dialog windows are for the user to make different choices for the Gateway. The dialog windows are just visible when the specific choices are to be done. The information that has to be provided by the user through the dialog windows is:

- The name of the OPC server
- The name of the computer that the server is installed on
- The path to the system in Advant Control Builder
- Which of the possible connections that are to be active
- The path to the system in Simulink
- The name of the system in Simulink

The user interface is built up with MFC (Microsoft Foundation Classes). In MFC a window has its own class that handles all the necessary implementation for that window. The class takes care of the construction and the destruction of the window and it also implements the window's functionality, e.g. data exchange between a dialog window and the rest of the application. Table 7.3 lists the window classes and their purposes. The windows and their classes are described below.

| **Class** | **Purpose** |
|---|---|
| MainFrm | Handles the main window |
| ServerConDlg | Handles the Connect Server dialog window |
| AcbProjDlg | Handles the Choose ACB-Project dialog window |
| AcbSimConDlg | Handles the Connections dialog window |
| SimSysDlg | Handles the Choose Simulink System dialog window |
| MainConDlg | Handles the Connect dialog window |

**Table 7.3** User interface classes.

Pictures of the windows can be seen in chapter 10, Users Manual.

### 7.3.1 MainFrm

MainFrm is the class that handles the main window. The main window is a standard MFC window with some extensions in the pop up menus for our purpose. The menus are only enabled when it is possible to make the specific choice. This means for example that if the user cancels a dialog window the only way to continue is to bring up the same dialog window or to start over from the beginning. Under the Configure pop down menu there are tabs that are used to make a complete connection between Simulink and the Advant Control Builder. Under the file menu three more tabs are added. Each tab is connected to a function that is executed when the user clicks on the tab. The main window is shown in figure 10.1. Table 7.4 lists the tabs under the Configure menu, the tabs under the File menu and their belonging functions.

| **Tab** | **Function** |
|---|---|
| Connect Server | OnConfigureConnectServer |
| Choose ACB-Project | OnConfigureChooseAcbProj |
| Used Connections | OnConfigureUsedConnections |
| Choose Simulink System | OnConfigureChooseSimulinkSystem |
| Connect | OnFileConnect |
| Disconnect | OnFileDisconnect |
| Reconfigure | OnFileReconfigure |

**Table 7.4** Tabs and belonging functions under the Configure menu and the File menu.

The Gateway program is event based. This means that the execution of the program is controlled by certain events. The different events occur when the user clicks on the tabs described above. It is these events that make the functions in table 7.4 to execute. This means that all the functions described in 7.1 and 7.2 have to be called from within one of the functions in table 7.4. How this is done is discussed later in the part 7.4, Execution of the Gateway Program.

### 7.3.2 ServerConDlg

ServerConDlg is the class that controls the Connect Server dialog shown in figure 10.2. This dialog is shown when Connect Server is chosen in the Configure pop up menu. In the Connect Server dialog the user have to make two choices in the edit boxes. In the first box the type of OPC-server to be used must be declared. The second edit box is for the user to print in the name of the computer where the OPC-server is hosted. If this edit box is left empty the Gateway will look for the OPC-server on the same computer that the Gateway is running on. The code behind this dialog binds the strings in the edit boxes to two variables. The variables are then used in the function InitOPCRemote or InitOPC. When the next button is pressed the Connect Server dialog disappear and the Choose ACB-project dialog is shown.

### 7.3.3 AcbProjDlg

The class AcbProjDlg controls the Choose ACB-Project dialog shown in figure 10.3. This dialog window is shown when Choose ACB-Project is selected in the Configure pop up menu, or when the next button is pressed in the Connect Server dialog. In the Choose ACB-Project dialog window the users must declare the path to the ACB-project they wants to use. The separation between the application and the modules must be with a dot. The path that is declared in this dialog is later to be used in the function DefineItems. When the next button in this dialog window is pressed the Connections dialog window is shown.

### 7.3.4 AcbSimConDlg

When Used Connection is selected in the configure pop up menu or when the next button is pressed in the Choose ACB-project dialog window the Connections dialog window is shown see figure 10.4. In this dialog window the user should select which connections that are to be active. This is done by selecting the wanted connections in the check boxes. Only the possible connections, depending on what ACB-project chosen, are shown in the dialog window. The OPCOut ----->Input column are the connections to be made from ACB to Simulink, and the OPCIn <----- Output column are the connections to be made from Simulink to ACB. The class behind this dialog window is AcbSimConDlg. In this function boolean variables are connected to the checkboxes. The Boolean variables are used to know which connection the user wants to make. When the next button is pressed in the Used Connection dialog window the Choose Simulink System dialog window will turn up.

### 7.3.5 SimSysDlg

SimSysDlg is the class behind the Choose Simulink System dialog window, see figure 10.5. This dialog window is shown when Choose Simulink System is selected under the configure pop up menu or when next is pressed in the Connections dialog window. In the Choose Simulink System dialog window the name of the Simulink system, without the mdl extension, should be declared and the path to the directory in which the system is placed. The s-functions inputblock and outputblock must also be placed in the same directory as the Simulink system. It is possible to write the system name and the path in the edit boxes or to browse to the system and select it. This is done when the browse button is pressed. When the browse dialog window, see figure 10.6, is shown the default file extension is mdl-files. The possibility to choose all file types is given but Simulink demands mdl-files. The path name is later used in the function InitMatlab to set Matlab workspace to the right directory and the system name is to open the Simulink system to be simulated. When the user has made its choice the next button should be pressed and this will causes the Connect dialog window, see figure 10.7, to show.

### 7.3.6 MainConDlg

The MainConDlg class will open the Connect dialog window. The function is started when the Connect button under the File pop up menu is chosen or when the next button in the Choose Simulink System dialog window is pressed. The Connect dialog window declares that the connections between Advant Control Builder and Simulink will be completed when the next button is pressed. It is also always possible to choose the back button and this will show the previous dialog window. The back button is for just checking the previous configurations or to edit them.

## 7.4 Execution of the Gateway Program

All the above described functions are put together to make the Gateway work. To get a picture of how the Gateway program is executed, figure 7.1 may be helpful. The figure shows only one of many possible calling sequences. The shown calling sequence occurs when the user starts by clicking on the pop up menu Configure, chooses Connect Server and then clicks the next button on all of the dialog windows, i.e. goes straight through the configuration. If the user chooses Disconnect when the communication loop is running the loop is stopped and it is possible to activate or deactivate the connections. If the user chooses Reconfigure when the loop is running all the functions that are necessary for disconnection of the server and Matlab workspace are executed. The user can always choose Reconfigure, not just in the communication loop but also during the configuration.

Since the user can move between the different configure and file functions, by clicking back, next or cancel in the dialog windows, there is a lot of possible calling sequences. In the program there are a lot of flags to keep track of what is done and what is not. The program also contains error control so that the user can be informed if something is wrong, e.g. if the program can not find the requested server the user is informed about this by a dialog box. This gives the user a chance to choose another server or to check if he has misspelled the server name.

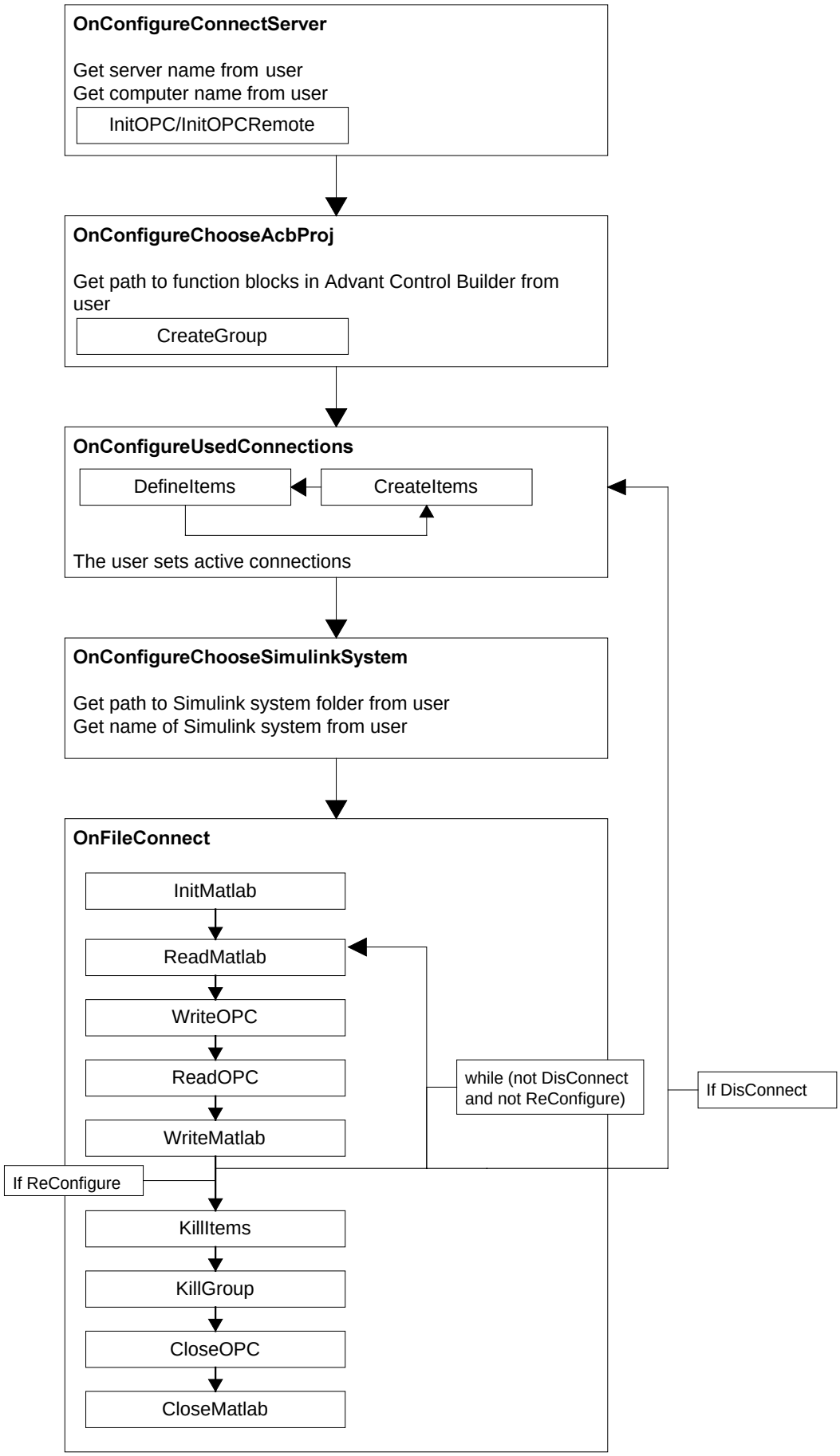


**Figure 7.1** The calling sequence for the Gateway program.

## 7.5 Difficulties with the Communication between the Gateway and Matlab

The idea of the Gateway communication is that the loop where the program reads and writes data should run as fast a possible. This will cause the Gateway to work only as a data exchange link between the OPC server and Matlab workspace. The sample time of the system is to be set in the Simulink model.

Since the Simulink thread has to be suspended for some time every sample period, this to make the simulation run in real time, Matlab workspace is also suspended. When Matlab workspace is suspended it is not possible for the Gateway to read data from, or write data to the workspace. This force the Gateway to wait until the Matlab thread wakes up before it can make any read or write operations. The time that the Gateway has to wait causes an unwanted delay in the communication. It takes some time for the Gateway to write and read to and from the OPC server and this time plus the time the Gateway waits for Matlab is always greater than the sample time specified in the Simulink model. Of course this is not a good behavior for the communication but we did not realize the problem until the final phase of our work.

If it is possible to find a way to access Matlab workspace even when the Simulink system is suspended, there will be no delay and the communication will probably be rather fast. Another way that maybe can solve the problem is to synchronize the Gateway so that it writes to Matlab workspace just before the thread is suspended and reads from the workspace right after the suspension time has run out. This would cause the Gateway to make its operations to the OPC server while Simulink is suspended and its operations to Matlab while Simulink is not suspended.

# 8 The Gateway Performance

The speed of the data exchange between the Advant Control Builder and Matlab is an important factor in the simulation. The ideal would be that when a value in an OPCOut block is updated, the corresponding value in the corresponding inputblock would change at the same time. Of course this is not possible, especially since the communication goes via a sometimes heavily occupied network.

We have investigated the performance of the Gateway program and noticed that there are a lot of factors that have influence on the speed of the communication. The major part of how fast the communication will be is the capacity of the network and how much traffic there is on the network. The speed of the communication can therefor differ depending on the time of the day when the program is running.

The computers that the different parts of the communication chain are placed on have dependency on how fast the communication will be. We have used a Pentium 2, 200 MHz computer where the Advant Control Builder, Matlab/Simulink and the Gateway program were placed and a Pentium 2, 166 MHz computer where the Advant Soft Controller and the OPC server were placed. These were the computers that we had access to during the thesis.

The size of the systems in the Advant Control Builder and Simulink has also some influence on the speed of the Gateway program. That is to say the amount of the OPCIn, OPCOut, inputblock and outputblock that effects the speed for the values to pass between the Advant Control Builder and Simulink.

Because of the problem with the real-time function in Simulink that was described in chapter 7.5 we have not done any actual tests on the speed of the Gateway communication. We are interested in the speed of the Gateway communication but with the problems the actual speed can not be measured. It is also rather difficult to draw some conclusions about which part that influence the communication the most, because there are various parts that effect the communication.

# 9 How to Create a New Simulink System

Simulink is a toolbox to Matlab and to bring up the Simulink library browser you write simulink in the Matlab workspace or you click on the Simulink icon in the Matlab toolbar. To build up a new system you click on the Create-a-new-model button in the Simulink library browser, see figure 9.1.

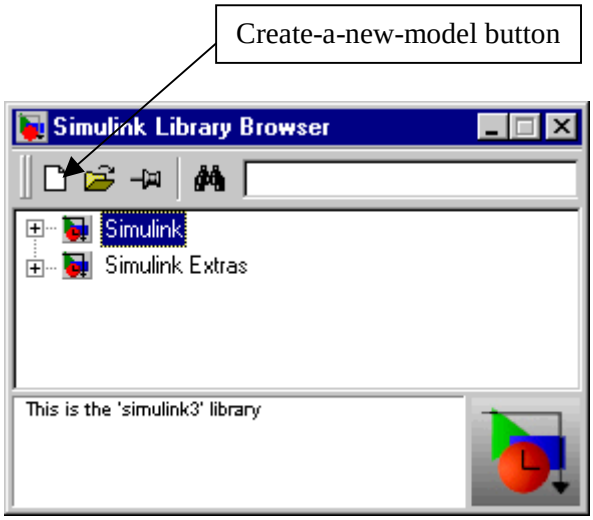


**Figure 9.1** The Simulink library browser.

When a new system is selected there is some important settings to make. In the simulation window you should select parameters under the simulation pop down menu, see figure 9.2.

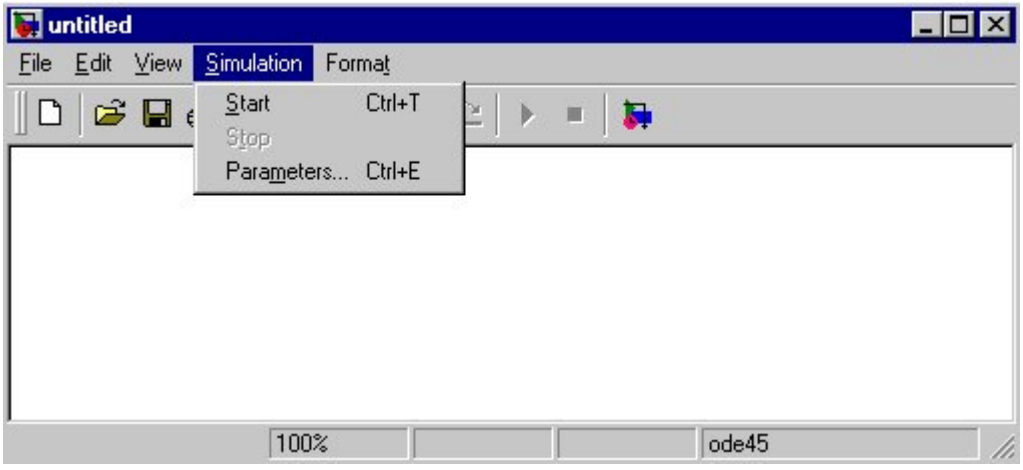


**Figure 9.2** A simulation window in Simulink.

In the new window that now appears there are three tabs, see figure 9.2.

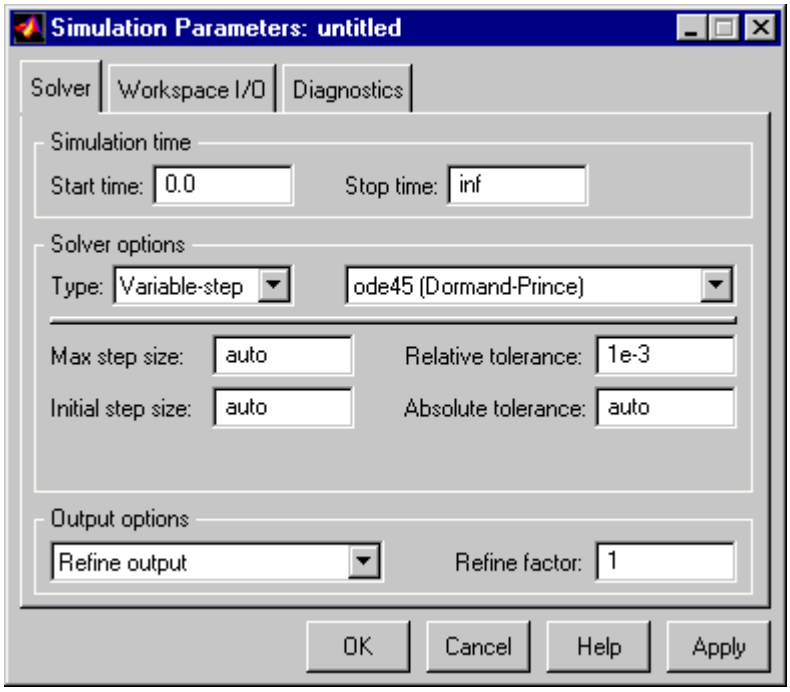


**Figure 9.3** The Simulation Parameters window when the Solver tab is selected.

Under the solver tab, which is selected in figure 9.3, the simulation stop time should be set to inf. This will cause Simulink to run the simulation continuously until the user stops the simulation manually. Under the tab workspace I/O *no* checkboxes should be chosen, se figure 9.4. This is because Simulink can e.g. save time values to workspace but the problem is that simulink saves the values in an internal array and put it to workspace after the simulation. When the simulation is going to run continuously Simulink has to swap to another memory space when the array grows and this causes the simulation to slow down.

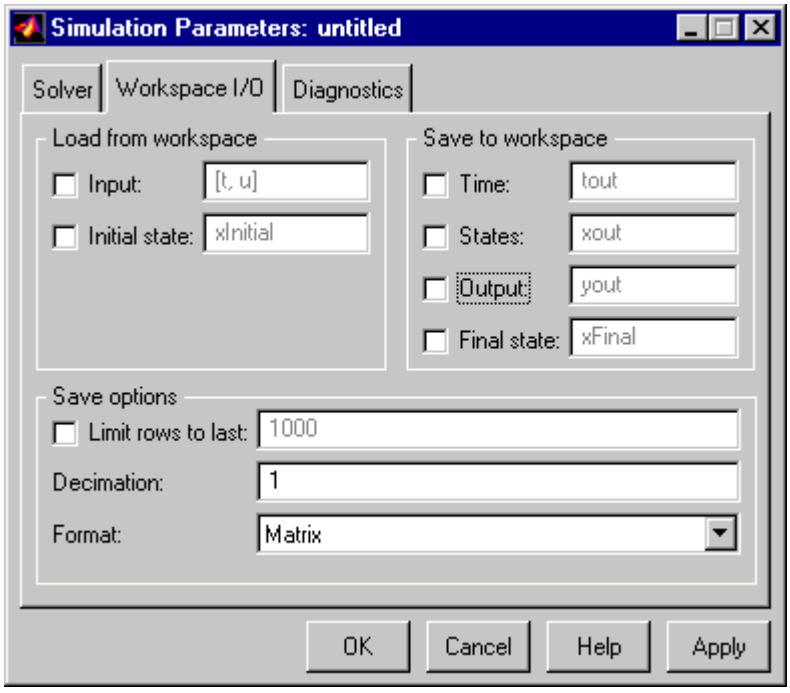


**Figure 9.4** The Simulation Parameters window when the Workspace I/O tab is selected.

When the simulation system is built up the in- and output blocks are created by first choosing an S-Function block under Functions & Tables in the Simulink library browser. The S-functions are named in the dialog window that appears when double clicking on the block, see figure 9.5.

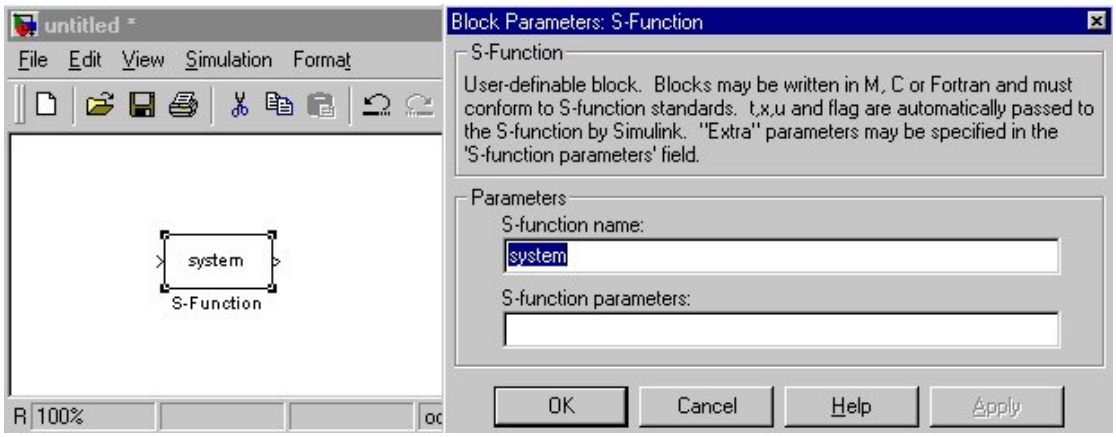


**Figure 9.5** The S-function window.

The in- and output block should be named inputblock1-5 and outputblock1-5. The connection inputblock1 is to be connected with OPC-out1 in Advant Control Builder. Likewise outpublock1 is to be connected to OPC-in1 in Advant Control Builder. The connections should accordingly have the same number in Advant Control Builder and Simulink. The field “S-function parameters” is used to fill in the sample time, in milliseconds, for the function.

Between the in- and output block the model of the process should be placed. The process could be either an S-function or be built up by various blocks in Simulink. It is also possible to see the actual value at any point in the simulation model by connecting a scope to a connection. The scopes are to be found under Sinks in the Simulink library browser. The time range in the scope

is edited under the properties button on the scope and the Y-axis is set under Axis properties menu bar that is shown on a right mouse click in the scope window, see figure 9.6.

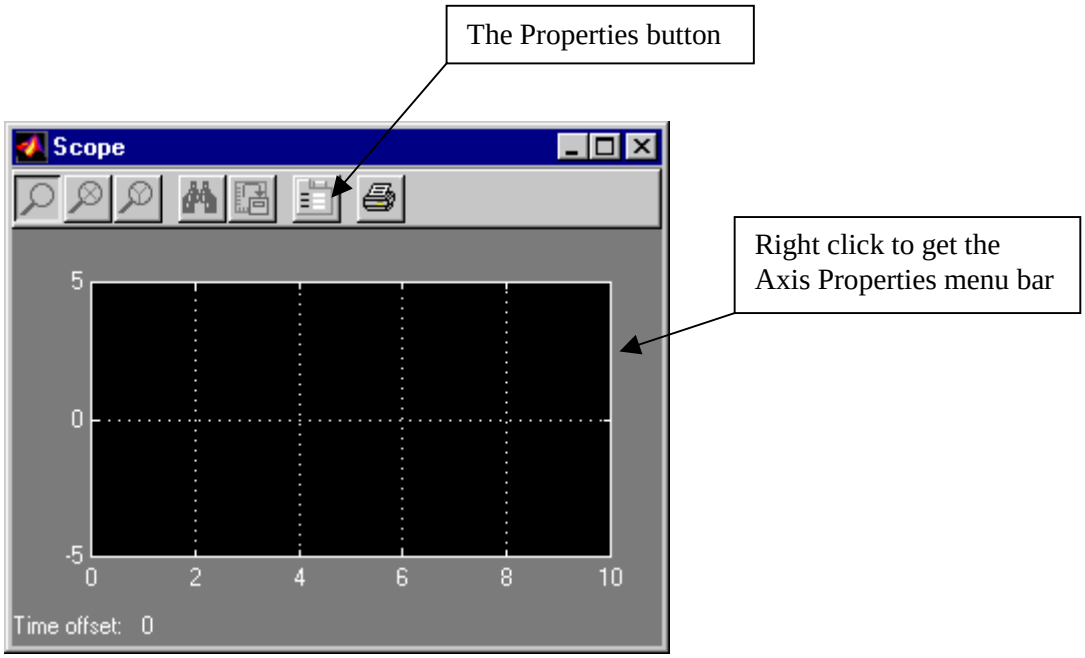


**Figure 9.6** The Simulink Scope window.

At last a real-time block must be placed in the simulation window. This is done by selecting a S-function block in the Simulink library browser and place it anywhere in the simulation window. The S-function must then be named realtimer. This is a function that we implemented to force Simulink to simulate in real-time. In the “S-function parameters” field the desired sample time, in milliseconds, has to be filled in. A scope must also be connected to the S-function block. This scope is used to show the jitter.

# 10 Users Manual

Before the Gateway could be started there are a few things to do. An Advant Soft Controller has to be running on a network connected computer. It could be the same as the Gateway is running on, but preferably on another. The reason for that is that the Advant Soft Controller takes quite a lot of computer power.

An OPC-server has to be running. It could be placed on the same computer as the Advant Soft Controller. The OPC-server also takes a lot of computer power so if possible, it should not run on the same computer as either the Gateway or the Advant Control Builder.

There must exist one working system in Advant Control Builder and one in Simulink. The system in Advant Control Builder has to be compiled and downloaded to the Advant Soft Controller. The OPC server has to be connected to the controller that contains the downloaded system.

When the Gateway program is started the main window is shown, see figure10.1.

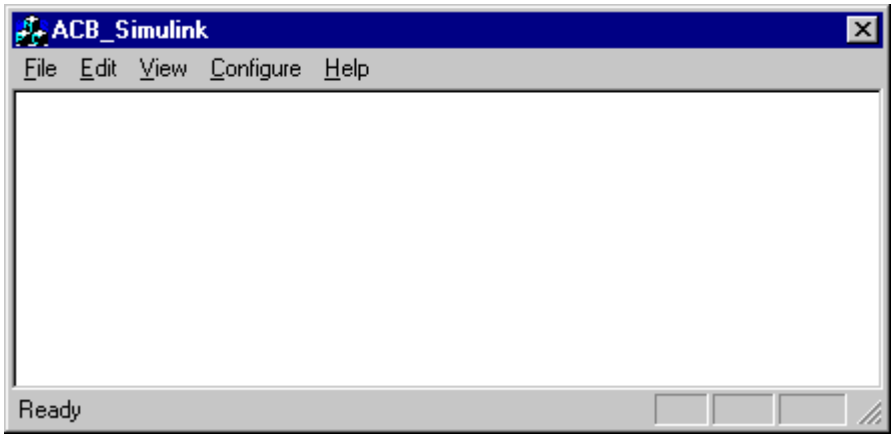


**Figure 10.1** The main window of the Gateway program.

The main window is visible all the time the Gateway is running. To start the configuration to a complete connection between the Advant Control Builder and Matlab/Simulink, the Connect Server tab under the Configure pop up menu has to be chosen. This is the only possible choice under Configure at the beginning. It is also possible to use the accelerator Ctrl+F1. When this choice has been done the Connect Server dialog window becomes visible, see figure10.2.

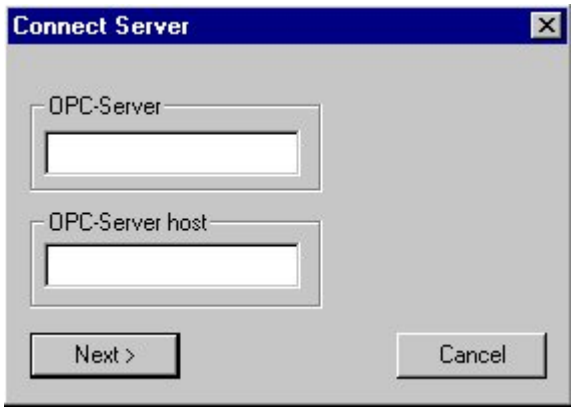


**Figure 10.2** The Connect server dialog window.

In the OPC-server edit box the user should enter what type of OPC-server he wants to use. This is the server ProgId. The server we used was called "AdvaControlOPCServer.OPCDAServer.1". In the OPC server host edit box the name of the computer where the OPC server is running on should be written. If the OPC-server is located on the same computer as the Gateways is running on, the OPC-Server host edit box should be empty.

When the necessary choices have been made the next button should be pressed. It is always possible to press the Cancel button and bring up the same dialog box from the Configure menu in the main window. It is also possible to restart the configuration from the beginning by choosing the Reconfigure choice under the File menu.

A click on the next button in the Connect Server dialog window will bring up the Choose ACB-Project dialog window, se figure 10.3.

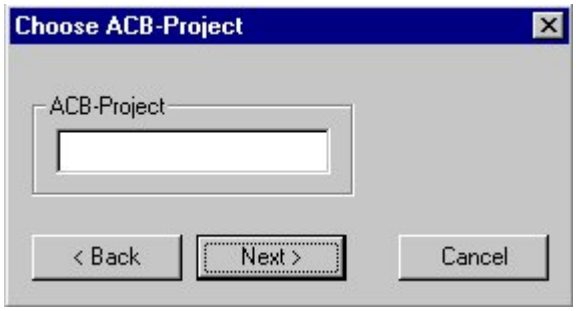


**Figure 10.3** The Choose ACB-Project dialog window.

In the ACB-Project edit box the path to the function block instances in the Application in the Advant Control Builder should be written. The separations between the Applications and the Modules should be marked with a dot. In this dialog window it is also possible to press a back button, that will bring up the Connect Server dialog window again. To continue the connection between the Advant Control Builder and Matlab/Simulink the next button should be pressed. This will cause the Connections dialog window to be shown, see figure 10.4.

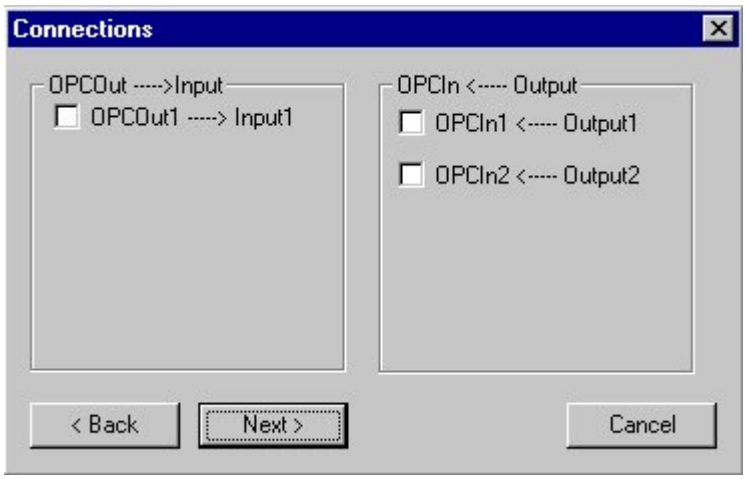


**Figure 10.4** The Connections dialog window with three connections possible.

The Connections dialog window shows all the possible connections that can be made. The column OPCOut----->Input is the connection from the OPC-server to Simulink and OPCIn<-----Output is the connection from Simulink to the OPC-server. It is then possible to choose which connections to be established. The Dialog window show the possible connections but it is not necessery to choose all of them. The next button will now bring up the Choose Simulink System dialog window, see figure 10.5.

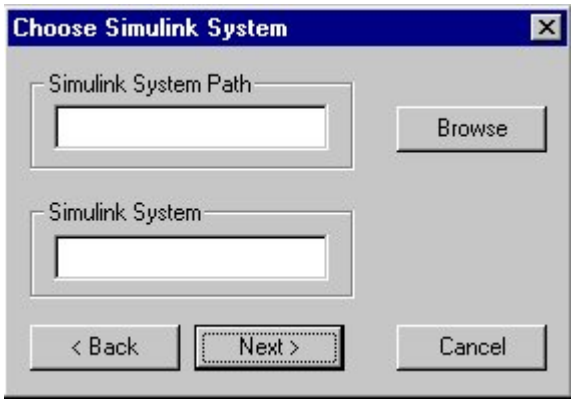


**Figure 10.5** The Choose Simulink System dialog window.

In the Simulink System Path edit box the path to the directory where the Simulink system is placed should be entered. It is also necessery to have the in- and outputblock files in the same directory as the Simulink system. In the Simulink System editbox the name of the Simulink system should be entered. The name should be typed without .mdl extension. It is also possible to use the browse button to find the system. If the brows button is pushed the Choose Simulink System browse dialog window will be shown, see figure 10.6.


**Figure 10.6** The Choose Simulink System browse dialog window.

The Browser window has as default file type simulink files (mdl extension). You may choose all file types but Simulink can only open .mdl files. When the correct file has been found the open button should be pressed. This action will cause the right path and system to be shown in the Choose Simulink System dialog window. To continue the connection between the Advant Control Builder and Simulink the next button should be pressed. Now the last dialog window will be visible. This is the Connect dialog window, see figure 10.7.


**Figure 10.7** The Connect dialog window.

This dialog window is used just to complete the connection. When the finish button is pressed the Gateway will open Matlab and the right Simulink system. The Simulink system is in stop mode when it is shown, but there is already a working connection between Matlab and the Advant Control Builder. When the start button is pressed in the simulation window Simulink starts to send and collect values to and from Matlab workspace.

Now there are a couple of alternatives to end the connection. The quickest way is to choose exit under the File menu or click on the shut down button in the Main frame window. This action will end the connection and turn off the Gateway program. To be able to turn off the connection and shut down Matlab, but still have the Gateway program running the user should choose Reconfigure under the File menu. After this the program is ready to establish a complete new connection. The last way to break the connection between Advant Control Builder and Matlab/Simulink is to choose Disconnect under the File menu. This will cause the communication to break and give the possibility to change the number of connections or to edit in the Simulink system. If Disconnect is chosen the connections dialog window, see figure 10.4, is shown. If the next button is chosen when the Gateway program is in this mode the Connect dialog window, see figure 10.7, will come up. The Choose Simulink System dialog window will be excluded since the Simulink System still is running. Before the connection between Advant Control Builder and Simulink is broken the simulation in Simulink must be stopped. This is done by a click on the stop icon in the simulation window.

# 11 Conclusions

The main goal with this master thesis was to investigate if it was possible to establish communication between the Advant Control Builder and Matlab/Simulink. If that was possible it then were to be implemented in C++ code. We found it possible and have done a Gateway program that establishes the communication between the Advant Control Builder and Matlab/Simulink. To get the values continuously all the way to a running simulation in Simulink we also implemented some S-functions for reading and writing data in Simulink. The communication from the Advant Control Builder is handled with COM/DCOM and an OPC-server. We used an OPC-server developed by ABB. The Gateway that we have implemented handles five connections at each way. It is possible to add more connections but it requires quite a lot of changes in the code. We limited the number of connections to five in the beginning because we needed to have a limitation and thought that it was a good start to test how it worked. The question of how many connections that are possible and realistic did arise but we did not have time to investigate it.

The communication between the Gateway and the OPC-server works satisfactory. Since we had to suspend the simulation in Simulink once every sample period, the communication between the Gateway and Matlab is blocked and this causes the communication to be too slow. A reason why we have not solved this problem is that we did not realize that the Matlab thread was suspended until the final phase of the thesis.

# 12 Future Possibilities

We have made a Gateway with at most ten possible connections between the Advant Control Builder and Matlab/Simulink. The desirable way to expand the Gateway should be to have the possibility to choose as many connections as desired. There are a lot of questions around such Gateway but we think that it is possible. The user interface where for example the connections should be chosen is one problem to solve. The Connections dialog, which shows the number of used connections, must be designed to expand depending on how many connections the user desires. Another interesting question is how many connections that are realistic.

Future development of our Gateway could be the possibility to save all settings made during the configuration of the Gateway to a file. It should then be possible to make the same connection from the saved settings. This file should perhaps have a unique extension for the system to know that it is a correct file. This could also be connected to a question in a dialog window that asks if the user wants to save the used settings before the Gateway is shut down.

To get more use of the Gateway a lot of processes, to be simulated, have to be built in Simulink. This can either be done by connecting different smaller blocks in Simulink or by writing S-functions. For identification of processes Mathworks provides a toolbox to Matlab. This toolbox is called System Identification Toolbox. It provides tools to generate a mathematical model of a dynamical system. The identification is based on observed input/output data.

There are some more desirable extensions to be made to the Gateway program. To show in the main window all the settings that were made during the configuration and what the program has performed is one of them. It should also be possible to print out what is shown in the main window. Another extension should be to add a browser in the Choose ACB-Project dialog window. This will make it easier to find the function block instances in the Application in the Advant Control Builder.

The problem that the communication between the Gateway and Matlab is locked when Matlab is suspended must be solved. Either by making it possible for the Gateway to access Matlab even if Simulink is suspended, or by synchronizing the Gateway so that it is only accessing Matlab when the thread is not suspended.

It is always desirable to get the communication to work faster. The way from the Advant Control Builder to Matlab/Simulink includes a lot of different parts and new development of these parts may do the entire communication faster. New features to our functions in the Gateway can also make the communication faster.

The Advant Control Builder is still under development and more sophisticated versions of Matlab/Simulink is certainly to come. We hope that our Gateway will work together with future versions of both Advant Control Builder and Matlab/Simulink, but some improvements may force our Gateway to be changed.

# 13 References


[1] Dale Rogerson, “Inside COM”, Microsoft press 1997

[2] OPC, OLE for Process Control, “Data Access Custom Interface Standard v2.0”, OPC Foundation October 1998

[3] OPC, OLE for Process Control, “OPC Overview v1.0” OPC Foundation October 1998

[4] Advant Control Builder, “Getting Started Basics and Installation v1.1”, ABB

[5] Matlab, “Using Matlab v5”, The Mathworks Inc.

[6] Matlab, “Matlab Application Program Interface Guide v5”, The Mathworks Inc.

[7] Simulink, “Using Simulink v3”, The Mathworks Inc.

[8] Simulink, “Writing S-Functions v3”, The Mathworks Inc.

[9] Matlab, “Application Program Interface Reference v5”, The Mathworks Inc.

# Appendix A

```
/*
 * File : tank.c
 */

#define S_FUNCTION_NAME  tank
#define S_FUNCTION_LEVEL 2

#include <math.h>
#include "simstruc.h"

static real_T  height=40;
static real_T  radius=3;
static real_T  level;
static real_T  levelOld;
static real_T  FlowIn;
static real_T  FlowInOld;
static real_T  FlowOut;
static real_T  FlowOutOld;
static real_T  Ts=0.01;


/* Function: mdlInitializeSizes ===============================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumSFcnParams(S, 0);
   if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
      return; /* Parameter mismatch will be reported by Simulink */
   }

   if (!ssSetNumInputPorts(S, 1)) return;
   ssSetInputPortWidth(S, 0, DYNAMICALLY_SIZED);
   ssSetInputPortDirectFeedThrough(S, 0, 0);

   if (!ssSetNumOutputPorts(S,1)) return;
   ssSetOutputPortWidth(S, 0, DYNAMICALLY_SIZED);

   ssSetNumSampleTimes(S, 1);

   /* Take care when specifying exception free code - see sfuntmpl.doc */
   ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}

/* Function: mdlInitializeSampleTimes =========================
 * Abstract:
 *    Specifiy that we inherit our sample time from the driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
```

```
   ssSetSampleTime(S, 0, Ts);
   ssSetOffsetTime(S, 0, 0.0);
}

/* Function: mdlStart=======================================*/

#define MDL_START
#if defined(MDL_START)

static void mdlStart(SimStruct *S){

   FlowIn=0.0;
   FlowInOld=0.0;
   FlowOut=0.0;
   FlowOutOld=0.0;
   level=0.0;
   levelOld=0.0;

}
#endif


/* Function: mdlOutputs=====================================
 * Abstract:
 *
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
   int_T            i;
   InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
   real_T           *y    = ssGetOutputPortRealSignal(S,0);
   int_T            width = ssGetOutputPortWidth(S,0);
   real_T           a=5.0;

   FlowIn = *uPtrs[0];
   FlowOut = a*sqrt(levelOld);
   *y = levelOld;
}

/* Function: mdlUpdate======================================*/

#define MDL_UPDATE
#if defined(MDL_UPDATE)

static void mdlUpdate(SimStruct *S, int_T tid){

   level = levelOld + (FlowIn - FlowOut)*Ts/(radius*radius*3.14);
   if(level > height){
      level = height;
   }
   if(level < 0.00001){
      level = 0.00001;
   }
   levelOld = level;
```

```
}
#endif


/* Function: mdlTerminat=====================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
}


#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```