ISSN 0280-5316
ISRN LUTFD2/TFRT--5785--SE

Object Oriented Automation Systems

Maja Arvehammar

Department of Automatic Control
Lund University
February 2007

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

Document name
MASTER THESIS

Date of issue
February 2007

Document Number
ISRNLUTFD2/TFRT--5785--SE

Author(s)
Maja Arvehammar

Supervisor
Mattias Wallinius at Tetra Pak D&E in Lund
Karl-Erik Årzén at Automatic Control in Lund

Sponsoring organization

Title and subtitle
Object Oriented Automation Systems (Objektorienterad automation)

Abstract
This master's thesis is about the implementation and evaluation of a small object-oriented automation system. By using a realtime Java VM from Jamaica, a sheet feeding magazine from the machine Tetra Aptiva Aseptic has been controlled. The expectations included to achieve more structured and safer programming, better documentation through UML and to separate application developers from developers of basic functionality. The work has involved to understand the existing control program, to design an object oriented model in Java and to run it on a test rack. The system worked fine, and most expectations were fulfilled. The performance measurements indicated that the Java was fast but also had a slightly larger jitter.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
70

Recipient's notes

Security classification

Preface

This thesis marks the end of Maja Arvehammar's Master of Science degree in Engineering Physics at the Lund Institute of Technology in Sweden.

The project work was carried out at Tetra Pak D&E in Lund and has taught me plenty. Mostly about managing different software tools and interaction between hardware and software, but also a little about how large companies work and many other things.

I would like to thank my supervisor att Tetra Pak, Mattias Wallinius for support and inspiration and the people at B&R and aicas for helping me through the tricky parts in managing their hardware and software tools. I would also like to thank Karl-Erik Årzén, my examiner at LTH, the co-workers at Tetra Pak and family and friends for supporting me during this master thesis.

Contents

1 Introduction 11
- 1.1 Background . . . 11
- 1.2 Vision . . . 12
- 1.3 Project task and methods . . . 13
- 1.4 Limitations . . . 14

2 Companies involved 16
- 2.1 Tetra Pak . . . 16
- 2.2 aicas . . . 16
- 2.3 B&R . . . 17

3 Theoretical background 18
- 3.1 IEC 61131-3 . . . 18
- 3.2 The Java Native Interface . . . 19
- 3.3 UML and Design Patterns . . . 21

4 Software tools 23
- 4.1 Automation Software . . . 23
- 4.2 Realtime Java . . . 26
- 4.3 Jamaica VM . . . 27
- 4.4 Eclipse . . . 27
- 4.5 ArgoUML . . . 27
- 4.6 Other Programs . . . 28

5 The sheet magazine 29
- 5.1 Hardware . . . 29
- 5.2 Software . . . 29
- 5.3 The Machine Phases . . . 32
- 5.4 The Magazine State Machines . . . 32

6 Design and Implementation 35
6.1 Solution overview 35
6.2 JNI Interface 35
6.3 State Machines 36
6.4 Package Overview 37
6.5 Program Flow 39

7 Performance Testing 44
7.1 Test Program Flow 44
7.2 Calculations 45
7.3 Results 45

8 Discussion 47
8.1 Vision Fulfilling 47
8.2 Performance Testing 49
8.3 Further Work 49

Bibliography 49

Glossary 52

A Additional Figures 54

B Test Results 65

C Implementation Methods and Issues 69

List of Figures

3.1 Example of appearance of a Sequential Function Chart program. The picture is taken from Automation Studio's help file (used with permission) 20
3.2 Example of appearance of a ladder diagram. The picture is taken from Automation Studio's help file (used with permission) 20

4.1 Cam control in history. The picture is taken from Automation Studio's help file (used with permission) 24
4.2 Cam control in software. The picture is taken from Automation Studio's help file (used with permission) 25
4.3 The simplified model of the S88-interface used by mclib and the magazine software. The picture is taken from Automation Studio's help file (used with permission) 26

5.1 The event flow of the magazine. The axis controlled by the magazine software is represented by the rightmost circle. The figure is drawn by Johan Henricson (used with permission). 30
5.2 An UML diagram showing the hardware hierarchy of the magazine. Drawn by Mattias Wallinius (used with permission). 31
5.3 An internal state machine (i.e. switch-case statement) from the function block calculating signals for the cam control. 33
5.4 The state machine of the cam control. Each state (except from state 0) has an own cam curve. 33

6.1 An overview of the solution architecture 36
6.2 State machine for the sheet feeding servo 37
6.3 Shows the state mapping between the magazine and the machine. The leftmost state machine is an excerpt from the one in Appendix A. The middle is represented by an integer stateLevel and the rightmost is the one from Figure 6.2. 38
6.4 Shows an overview of all classes in the magazine application . . . 40

6.5 Shows the thread communication through monitors. RTFStateMachine and FeederThread are threads. RTFMonitor, Magazine and the Axis classes are monitors. 42
6.6 Shows the sequence of awaiting a cam state index. 43

A.1 A minor part of the axis handler state machine. The function block starts in state 0 and the axis is up and running (and awaiting external commands) in state 500. 55
A.2 Machine phases. Continuing in the next figure. 56
A.3 Machine phases. Continuing in the next figure. 57
A.4 The first of the machine phases. 58
A.5 The brasruntime package providing basic motion controlling functionality 59
A.6 The brasruntime.event package providing basic event handling . 60
A.7 The brasruntime.configuration package facilitating axis configuration 61
A.8 The utilities package with a few useful classes. 62
A.9 The com.tetrapak.a5.packagingline.rtf package representing the rest of the machine communicating with the magazine. 62
A.10 The com.tetrapak.a5.packagingline.rtf.magazine package containing classes for magazine control. 63
A.11 The com.tetrapak.a5.packagingline.rtf.magazine.magazineServo package containing axis-classes for the magazine servo. 64

List of Acronyms

ANSI American National Standards Institute

AR Automation Runtime

AS Automation Studio

B&R Bernecker&Rainer

D&E Development & Engineering

GmbH Gesellschaft mit beschränkter Haftung (company with limited liability)

HMI Human Machine Interface

IEC International Electrotechnical Commission

ISA Instrumentation, Systems, and Automation Society

JAR Java ARchive

JNI Java Native Interface

LTH Lunds Tekniska Högskola (Lund Institute of Technology)

OO Object Oriented

PLC Programmable Logic Controller

PVI Process Visualization Interface

SFC Sequential Function Chart

ST Structured Text

SysML Systems Modeling Language

TAA Tetra Aptiva Aseptic

UML Unified Modeling Language

VM Virtual Machine

Chapter 1

Introduction

1.1 Background

The software solutions that Tetra Pak develops for their automats are very much focused on old solutions, consisting of PLC programs written in ladder diagrams and other primitive languages. The standard of PLC programming was mostly developed during the 80s, standardized by the International Electrotechnical Commission (IEC) in 1993 [8] and lacks features like object-orientation and declarative programming. The standard for industrial control programming is called IEC 61131-3 and defines five different language standards, both textual and graphical.

Although several standard libraries for basic functions and motion control have been developed, see PLCopen [13], they are still quite primitive and require much knowledge from the developer. In B&R Automation Studio, a development environment for PLC-programs, a library is defined by a collection of function blocks compiled to a library file.

An object-oriented system has very much to offer when it comes to develop PLC programs. Languages like Java, C++ and C# include extensive standard libraries, inheritance possibilities and ready-made design pattern solutions for common problems. A design pattern is not a natural part of an IEC61131-3 program. Furthermore, there are several documentation standards like UML and SysML which (when used correctly) can contribute enormously to both design, documentation and debugging of applications. They can also make the programs more easy to read and understand.

Several development environments, unit testing and programs for UML-drawing are already common tools in the object-oriented world. Since the tools already exist, why not use them for PLC-programming as well?

It would be interesting to see if, for instance, Java can be used to control a

Tetra Pak machine automat, if it is stable enough, fast enough and at all works for a hard realtime system.

1.2 Vision

This section presents some of the advantages to gain with an object-oriented (OO) automation system.

Structure, test, model and document applications

Large automation programs, large function blocks with many uses are very difficult to completely test and maintain. As the control programs grow larger, this becomes more and more of an issue. OO-methods offer the UML standard to model large programs by using sequence- and activity diagrams to show process flows, state diagrams to show the machine states and not least: class diagrams to show application design. Design solutions for common problems exist in established design patterns to keep programmers from constantly reinventing the wheel. Several tools for unit testing OO-programs already exist for the debugging and validation of applications.

To be able to structure and specify the software solution modeling is a necessary part. A standard for this is UML [7]. Properly used it shortens development times and reduces the number of errors in the code. It also leads to better test specifications and better testing since unit testing is possible. Tools in which you can test and debug at UML diagram level lead to greater productivity for the application developers.

Privacy and reuse of code

OO-libraries with clear hierarchies and well-designed interfaces would increase the possibility of reusing code. Using inheritance and common interfaces, it would be possible to represent mechanical parts of the automat by objects. If a mechanical solution is reused, much of the controlling code could also immediately be reused. One could, for instance, create a standard class library representing basic functionality for a specific mechanical part.

Compared to Automation Studio (AS) function blocks, the OO world also provides greater possibilities to protect code. When calling an AS function block in C, all variables are totally unprotected. It is just as easy to set an input variable as an output- or internal variable (the reason is described in Section 4.1). Using the OO class-definition it is natural to declare how private a function or variable is.

Other common functions like communication standards can also be abstracted with OO libraries and reused. PLC communication is yet at a rather low abstraction level requiring the programmer to know plenty about each way to communicate.

Separating developers

As mentioned in the previous section, OO would simplify development of separate libraries for common functions and basic functionality of mechanical parts. It would be favourable to separate application engineers that are design experts from developers of basic functionality that specialize on the hardware. It would allow one group of hardware experts to focus only on creating basic libraries while the application developers could focus solely on object diagrams, composition, execution sequences, process flows and state diagrams. They would use the basic libraries in their applications, without caring about the code inside.

Graphical programming

Ladder programming is a graphical programming language, looking very much like an electric circuit, see Section 3.1. The Sequential Function Chart language is also graphical, where the objects represent states and transitions. Graphical programming has several advantages and the vision is that the graphical programming methods will remain, but at a much higher abstraction level. The application programmer would be able to create programs by drawing class diagrams and state charts in a ”drag-and-drop”-like way, based on the standard libraries. Tools already exist for this. Instead of drawing coils and relays one could draw axes and sensors, and perhaps assign certain behaviours to them.

1.3 Project task and methods

The task of this thesis project is to run Java on a (test rack for a) sheet magazine from the Tetra Pak automat TAA, Tetra Aptiva Aseptic. The goal is to examine if it is possible to get a Java environment to work on the hardware platform, and see if it meets the hard real-time requirements. The next step is to develop base functionality for motion control and then use UML or SysML tools for the application programming.

Several software and hardware tools have been used to achieve this. The sheet magazine, i.e. the target system, is from B&R (introduced in Section 2.3). It uses the operating system VxWorks with the runtime environment B&R Automation Runtime. The file system on target is FAT32.

A Java VM suitable for the PLC is provided by aicas (presented in Section 2.2) and is called Jamaica VM (which is described in Section 4.3).

The magazine's task is to separate sheets from each other and pass them on at a very high speed. The existing control program for the magazine is written in the IEC 61131-3 languages (presented in Section 3.1) and C. The major task for it is to control an ACOPOS Servo from B&R using Cam control (explained in Section 4.1) and to response to sensor-information. The magazine is described further in Section 5.

Part of the task in this thesis is to rewrite the control program in Java. Since the Java-application could not directly communicate with the ACOPOS, an intervening layer had to be created. A Java Native Interface (see Section 3.2) was the solution, Java calling C-functions to send commands and obtain status information and measurements. With object-oriented programming most of modeling is done upfront, using UML to draw state machines, classes and threads before implementation and testing.

The concrete work can be divided into the following sub-tasks:

- Set up the environment and make the necessary arrangements to make JamaicaVM work with B&R hardware and Automation Runtime (AR). Make sure the standalone Java application can be started from within a program written in Automation Studio.
- Develop a JNI for basic motion control to enable communication with the ACOPOS. Test it with Automation Runtime.
- Develop a standard class library to make the basic functionality available at a higher Java-level.
- Read and understand the sheet magazine control program (described in Section 5.2) in order to create an object-oriented design.
- Create an object-oriented model of the magazine software. This means drawing state machines, class diagrams and illustrate communication between threads.
- Implement the solution. Test it with Automation Runtime and the JNI.
- Test the application and measure the performance.

1.4 Limitations

At the time for this project, it was not possible to develop all the basic functionality using Java, the hardware was too dependent on the Automation Runtime

solution from B&R. It would also be far too much work. Therefore the existing motion control C-library mclib together with cyclic AR tasks, have been used and is called from the Java application using the Java Native Interface. This solution might have a significant negative effect on the magazine performance.

Chapter 2

Companies involved

This section contains a brief description of the companies involved and how they have contributed to the project.

2.1 Tetra Pak

Tetra Pak is one of the leading companies in processing, packaging and distribution of groceries. It was founded by Ruben Rausing and Erik Wallenberg in 1951 and has now more than 20000 employees in more than 160 countries.

Tetra Pak's vision is to make food safe and available everywhere, see Tetra Pak's webpage [14]. To achieve this vision, they work together with their customers to develop intelligent process- and packaging solutions for groceries.

This master's thesis project has been carried out for and at Tetra Pak D&E Automation and Line Integration in Lund, using a magazine from their filling machine Tetra Aptiva Aseptic.

2.2 aicas

T he company aicas GmbH has developed a way to use Java in hard realtime control systems. This includes a virtual machine, object oriented development environment and analysis tools. Their aim is the promotion of modern software development methods in embedded and time-critical control systems, see aicas' webpage [1]. aicas is a growing company that was founded in 2001 in Karlsruhe, Germany and has 16 employees.

In this project, their Java virtual machine Jamaica has been used and tested on the magazine.

2.3 B&R

B&R was founded by Erwin Bernecker and Josef Rainer and their business defining motto Perfection in automation, as their their web page [2] states, has been valid for over 26 years.

The hardware used in this thesis, i.e. the magazine parts, were produced by this company. Their enclosed software, B&R Automation Software, has also been used.

Chapter 3

Theoretical background

To perform the tasks mentioned in Section 1.3 some theoretical studies had do be done. This section presents some facts relevant to the project and where they have been used.

3.1 IEC 61131-3

Since most of the magazine software (see Section 5.2) is written using the IEC 61131-standard, understanding the magazine design requires knowledge of some of the languages. IEC 61131-3 is a global standard for industrial control programming. It defines syntactic and semantic rules for two textual (Structured Text and Instruction List) and two graphical (Ladder and Function Block Diagram) languages. There is also the Sequential Function Chart syntax which is used to structure programs.

The IEC standard defines basic concepts like data types and variables. It also defines some standard functions, for instance basic maths (like ADD, SQRT and SIN). There are also user-defined functions which can be created by programmers. The standard also defined Function Blocks, which are said to be equivalent to Integrated Circuits, ICs, representing a specialized control function, IEC 61131-3: a standard programming resource [12]. Function Blocks may, unlike traditional functions, have several outputs and an internal state which remains between calls.

Below are descriptions of some languages which syntax has affected the magazine (and machine) control program design and for that reason are relevant.

Sequential Function Chart, SFC is basically states connected with transition conditions. Each state (also known as step) has a certain task, executed every time the SFC-block is called (like once every cycle) until the tran-

sition condition to exit is fulfilled. It may also contain an entry and an exit action. The first one is performed only when entering the state and the second just before exiting it. A state which action is being executed is described as active. An action may consist of a piece of code written in any IEC61131-3 language or C. The TAA machine steps through certain phases, and each phase steps through S88 states (Section 4.1). For that reason it is convenient to encapsulate the tasks for each state and phase into elements of SFCs. An example of what an SFC program might look like is shown in Figure 3.1.

Ladder is basically a drawing of an electric circuit. Commands and links are represented by connections lines and symbols like contacts and coils. On the left side there are an imaginary line continuously supplying power. On the right hand side the coils are placed, representing the devices to control (for instance a lamp or a boolean variable). Since state transitions often are determined by boolean variables (whether a not a task execution has succeeded, if an error has occurred, etc.) the ladder syntax is often useful. An example of a ladder diagram is shown in Figure 3.2.

Structured text, ST is, unlike the above ones, textual and very similar to Pascal. An ST program consists of a sequence of instructions, with standard high level statements like IF .. THEN .. ELSE, WHILE .. DO and SWITCH .. CASE. The magazine control program is built out of cyclic tasks (see 5.2 and 4.1), but sometimes it is desirable to execute large pieces of code sequentially. In that case state machines, with many small states, can be built out of SWITCH .. CASE statements, switching on a variable that is increased by 1 every cycle. This is very common behaviour in the magazine software, especially during the start up phase (initialize, configure, enable power and so on).

More information about the IEC61131-3 standard can be found on the PLCOpen webpage, [13].

3.2 The Java Native Interface

The Java Native Interface is a framework that enables a Java Virtual Machine to interact with native code written in for example C or C++. In this section only a few capabilities is presented and focus will be on the C language. The Java Native Interface by Shen Liang [11] is an extensive reference to JNI and contains information about how to implement it.

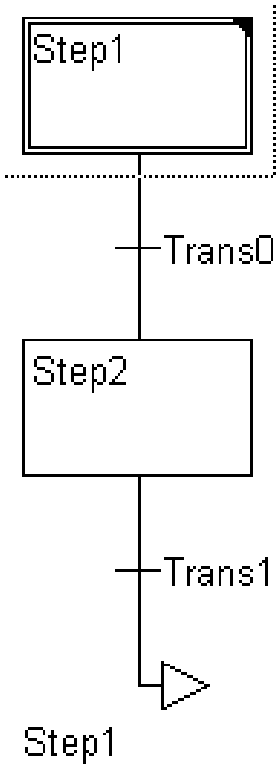


Figure 3.1: Example of appearance of a Sequential Function Chart program. The picture is taken from Automation Studio's help file (used with permission)

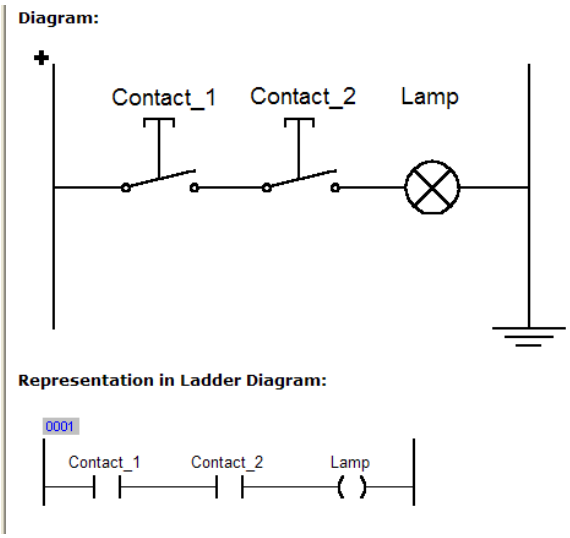


Figure 3.2: Example of appearance of a ladder diagram. The picture is taken from Automation Studio's help file (used with permission)

Calling native functions from Java

One JNI feature is to call methods written in C from a Java application. By declaring methods native in a Java application they can be implemented in a native library. The native methods must be declared in a special, non-trivial way. To obtain the correct declaration of a certain native method, the javah (or in Jamaica, jamaicah) program can be used with the class file as input. It generates a header-file containing declarations of all native methods of the class. The methods can then be implemented and compiled to a library. The library is loaded in the Java application using the System.loadLibrary() function. When the Jamaica builder is used, the library-loading is not necessary; Jamaica links the pre-compiled object files itself.

Mapping of types

The JNI defines a set of C-types that correspond to Java primitive types which can be used in C functions without further thought. When it comes to objects, arrays and strings, special care has to be taken. The JNI Environment provides a large set of functions to perform actions on these types, for instance accessing arrays and object fields.

Callbacks and Constructor invoking

Another possibility is to call Java functions from the C-functions, i.e. to perform callbacks. This is done by providing a special signature describing inputs and return type. The signature can be obtained by using the javap -p -s tool on the class file in which the function is. Constructors are invoked in a similar way. It is also possible to throw exceptions from JNI by using a certain JNI function.

Caching field and method IDs

To access a certain method or field, an ID that is defined by the symbolic name and type descriptor is required. The ID can be obtained by calling certain methods of the JNI environment, but includes a symbolic lookup which might slow the program down if performed often. A possible optimization is to cache the values when the class loads. This can be done by calling a native method in the static initializer of the class.

3.3 UML and Design Patterns

Various ways exist to model and document applications. The one used in this project is the Unified Modeling Language (UML). This includes class diagrams,

state charts and activity diagrams. Examples of these kinds of diagrams are shown in Figures 6.2 and 6.5 . Advantages of UML are for instance that one does not have to look at the source code to understand what a program does. State charts, activity diagrams and use cases make the design transparent.

A design pattern can be a standardized solution to a common problem and a major advantage in OO programming. The solution can be described with a class diagram showing hierarchies, dependencies and methods. Several design patterns have been used in the magazine design. One of the most common design patterns is the Observer pattern, where objects may observe each other and are notified when a state changes. The observer pattern has been used when implementing events, consisting of the IBRAxisEventNotification, BRAxisEvent and Mailbox classes in Figure A.6.

For more information about design patterns, look at the web page [5]. More about UML can be found on the Object Management Group webpage, [7].

Chapter 4

Software tools

This chapter presents some programs that were used in the project. The intention is to briefly describe their main purpose and what they were used for.

4.1 Automation Software

Automation Software is the environment for developing, testing and running control applications on B&R hardware. It consists of three different parts: Automation Runtime, Automation Studio and PVI transfer. Automation Runtime (AR) is the runtime environment and PVI-transfer the tool to transfer programs to the target (in this project that is the magazine cpu flash disk).

Automation studio

In Automation Studio (AS) the control programs are written, compiled, run and tested. The programs are built out of cyclic tasks, written in either Automation Basic[1], Ansi C, or one of the IEC 61131 languages. Libraries of function blocks can be created and some libraries for basic functionality (for example parsing strings) are included in the installation. Languages can be mixed freely inside a program, so that a Structured Text program can easily call a function block written in C.

Automation Studio also contains tools for configuring the hardware and change and watch process variables. A few cyclic tasks have been created using Automation Studio, to load and start the Java program and to manage mclib (further described in Chapter 6).

As mentioned before (in Section 3.1), calling a function block from C-language requires some caution. The function block input consists of a structure

[1] A language invented by B&R, [15]

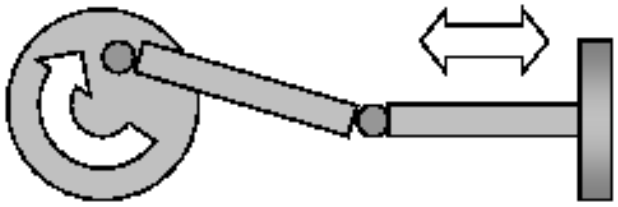


Figure 4.1: Cam control in history. The picture is taken from Automation Studio's help file (used with permission)

containing all variables, both input output and internal ones. The reason for that is that a function block has an internal state that has to endure during several calls.

Motion control library

Due to the limitations of this project an existing motion control library written in C has been used to control the ACOPOS servo (see Section 5.1). This library is called mclib and is created by B&R. It contains for instance functions to retrieve measured values, moving and power controlling functions. Many of the functions uses the PLC open interface, see Section 4.1, and need to be called several times before the execution is finished.

Cam control

Cam control originates from mechanical solutions where typically a rotating axis was connected to a pole to achieve a linear movement, as in Figure 4.1. The same result can be obtained using cam control, but with much more flexibility.

The principle is the same: a slave axis' movement depends on a master axis position. The slave axis follows a cam curve $y = f(x)$, where x is the master position, the principle is shown in Figure 4.2. In the magazine software (5.2), the master axis is a virtual axis rotating at a constant speed. Depending on sensor and trigger values (like indicating sheet position and sensing double sheets), the slave axis (i.e. the ACOPOS servo) follows one out of six different cam curves.

mclib cam control

The motion control library contains functions to perform the cam control. A large data structure containing all cam information is given as input. It contains up to 14 cam states, where each state contains a cam curve (except from state 0 which is empty). A cam state also has up to 5 events that trigger state changes.

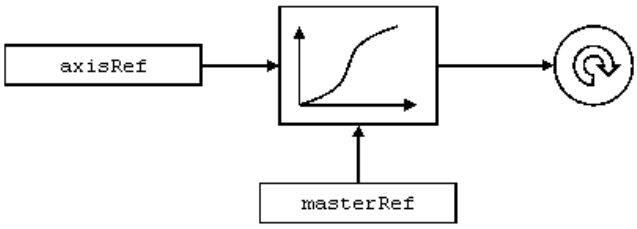


Figure 4.2: Cam control in software. The picture is taken from Automation Studio's help file (used with permission)

This enables simple and fast switching between different cam curves in order to achieve certain movement patterns.

PLC open interface

In the PLC motion control libraries, a common method is to start a function and then keep it active. An interface enabling this behaviour is the PLC Open interface. The function blocks have enable/active or execute/done variables. An example is the mc˙moveAbsolute function block in mclib. At a positive edge of the execute input, the function block can start communicating with the drive. As soon as the function is successfully completed, the done flag is set. The function block needs to be called several times before the execution is finished, due to the implementation of the function blocks. Function blocks may also abort or block each others execution since some has higher priority than others (for instance, mc˙stop has higher priority than mc˙moveAbsolute). If a function block is aborted, the commandAborted flag is set on the output. If an error has occurred, the output error is set, and the status integer output holds the error number.

A function block which execution does not finish has instead of execute/done the enable/active variables. An example of such a function block is mc˙actValues which reads measured values from the servo, for instance the axis position and if it is moving.

S88 States

S88 (also known as ANSI/ISA-88) was standardized by ISA 1995 and adopted by IEC in 1997. It provides models and terminology for batch control and is said to make the execution of automation projects more efficient. The standard defines several things, see for instance [4] or [9] but mclib uses a very simplified model of the S88 state and command matrix in their cam control. The same interface is used for the magazine software states (see Section 5.2). A picture

of the defined states and transition commands is shown in Figure 4.1. In the interlocked and ready state the application basically waits for commands to proceed to ready and run respectively. When a start command is recieved in the ready state a transition to run is performed. In the run state the ordinary operation actions are executed. When the running is finish a transition to done is performed. In the held state the equipment is placed in a safe state. A transition to held happens either when an exception from ordinary operations has occurred or when it is issued by the operator. Also in the aborted state the equipment is in a safe state. The transition to aborted is issued by the operator.

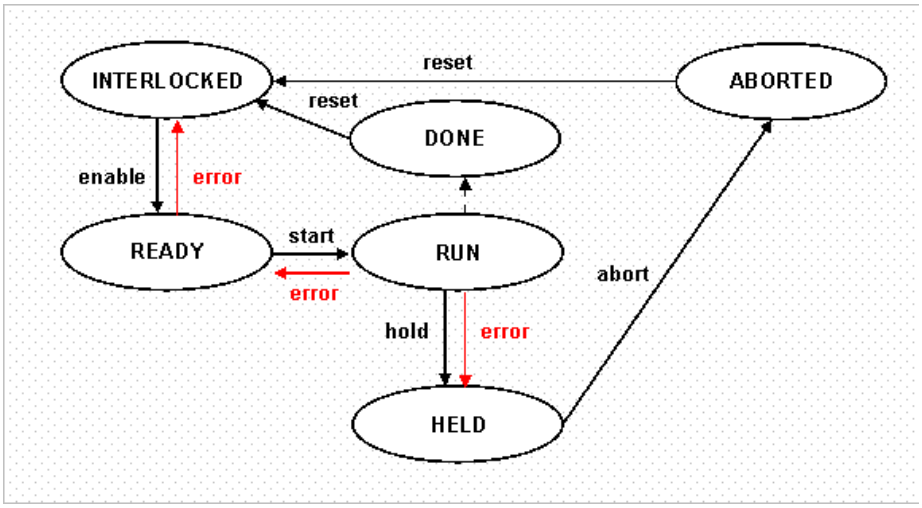


Figure 4.3: The simplified model of the S88-interface used by mclib and the magazine software. The picture is taken from Automation Studio's help file (used with permission)

4.2 Realtime Java

A common Java runtime environment uses a technique called just-in-time compiling. That usually means that bytecode is compiled to native machine code at runtime. Another technique is the ahead-of-time compiling, where the compilation to native machine code takes place before the program execution starts. An example of such a compiler is the Jamaica builder (see next section). The reason for ahead-of-time compiling is often to optimize the application on speed and memory requirements.

The unpredictable garbage collection has always been an obstacle when it comes to writing hard realtime applications in Java. There are several projects working on solutions to this. Jamaica, which is presented in the next section, is

one of them. Another one is RTSJ.

The Real Time Specification for Java (RTSJ) is an application programming interface to control the Java Virtual Machine behaviour in certain time-critical parts of the application. By letting the programmer manually managing the memory in certain memory areas it is kind of a work-around to the ordinary garbage collector. For more information, see for instance [16] and [3].

4.3 Jamaica VM

Jamaica Virtual Machine is a Java VM for realtime systems, designed to run under hard realtime conditions on realtime platforms. It features deterministic and efficient garbage collection, priority inheritance and other necessary real-time functionality, see [1]. It was created by Aicas (see 2.2). JamaicaVM also supports many of the Java standard libraries[2] and the Real Time Specification for Java[3].

In this project, the Jamaica VM was used to run Java on the VxWorks operating system, on B&R hardware. The tool consists both of a virtual machine and a builder, which translates the Java into C-code and creates a standalone application out of it (that is, ahead-of-time compiling). Jamaica also supports the Java Native Interface, which has been used to call the motion controlling C-functions from Java (see Section 4.1).

4.4 Eclipse

Eclipse is among other things an environment for developing, testing and debugging Java applications. It was developed by the Eclipse foundation, which is an open-source community (see [6]).

Many plugins exist for additional functionality. In this project a plugin for Jamaica was used. The plugin (developed by aicas) contains tools for configuring and running the Jamaica builder.

4.5 ArgoUML

ArgoUML is a tool for drawing UML-diagrams, like class- and activity diagrams and state charts. A useful feature is the export-functionality which enables projects to be exported to xmi (XML (Extensive Markup Language) Metadata Interchange) or source code for use in other contexts. It is also possible to export diagrams as graphics.

[2]Jamaica API at http://www.aicas.com/jamaica/doc/jamaica_api/index.html, [1]
[3]For further reading: https://rtsj.dev.java.net/

4.6 Other Programs

This section contains brief descriptions of other programs that have been useful, but has not taken especially large place in the project.

VMWare is a program that acts as a virtual PC. It was very useful when running programs that were not so stable as they ought to be.

Tornado, a development environment for VxWorks applications. It was used while examining the standard libraries in VxWorks, and Tornado's compiler was used by JamaicaVM to build the standalone application.

SVN, subversion is an open source version control system[4]. There is also a plugin for Eclipse which has been used in the project. It is called subclipse.

[4]The management of multiple revisions of the same project. Information on http://en.wikipedia.org/wiki/Revision_control and http://subversion.tigris.org/, [17]

Chapter 5

The sheet magazine

The hardware to be controlled is, as mentioned in the introduction chapter, a sheet magazine from Tetra Aptiva Aseptic (TAA). This section contains a description of the magazine hardware, functionality and the existing software. The software part is focused on internal state machines and does neither contain details on managing sensors nor any of the error handling.

5.1 Hardware

The sheet feeding process consists basically of the steps showed in Figure 5.1.

The magazine consists of several parts, optical sensors and digital I/O. An UML diagram showing the hardware hierarchy is shown in Figure 5.2.

5.2 Software

Programs developed in Automation Studio are normally built out of cyclic tasks (see Section 4.1). The application controlling the magazine consists roughly out of three cyclic tasks (several parts concerning error handling, sensor-checking and communication with other machine software parts have been left out). A major part of the application consists of SWITCH..CASE statements which have been interpreted as state machines. All state machines presented in this section that are implemented in Structured Text are actually such statements. The cyclic tasks controlling the magazine are:

- A fast task which has a cycle time of 2ms and is responsible for the automatic cam control (see Section 4.1). First it waits until the axis is active and then starts calling three function blocks. The first one is for calculating the cam curves, the second is for calculating the signals and the

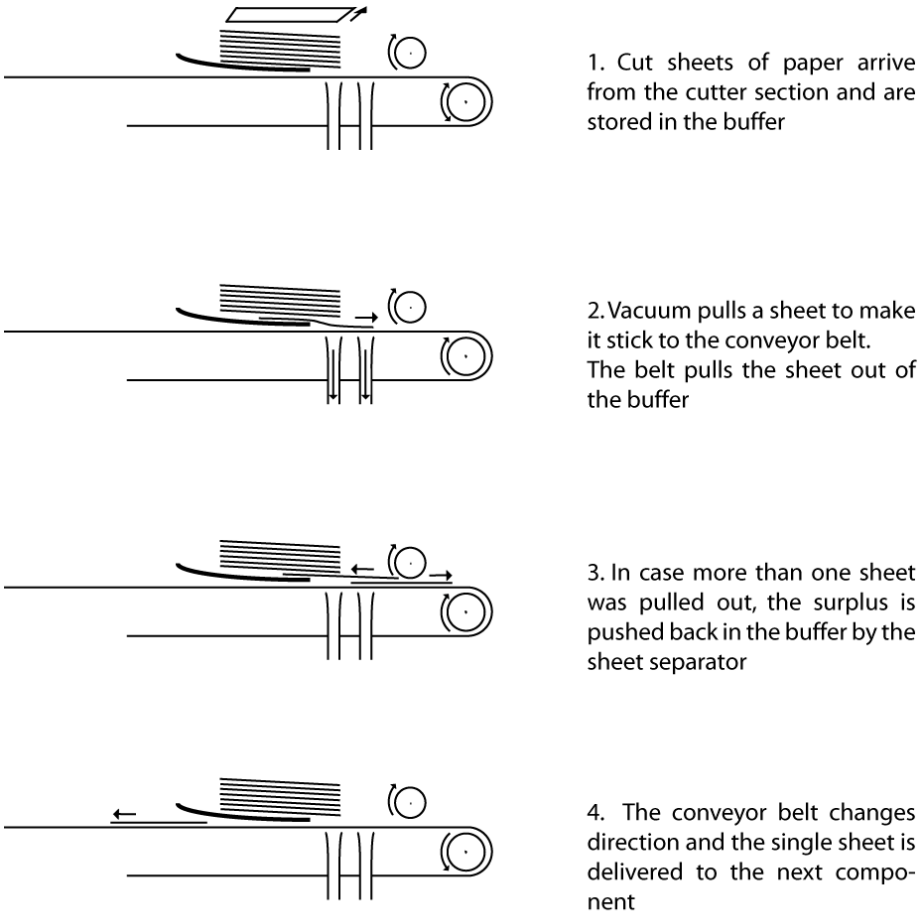


Figure 5.1: The event flow of the magazine. The axis controlled by the magazine software is represented by the rightmost circle. The figure is drawn by Johan Henricson (used with permission).

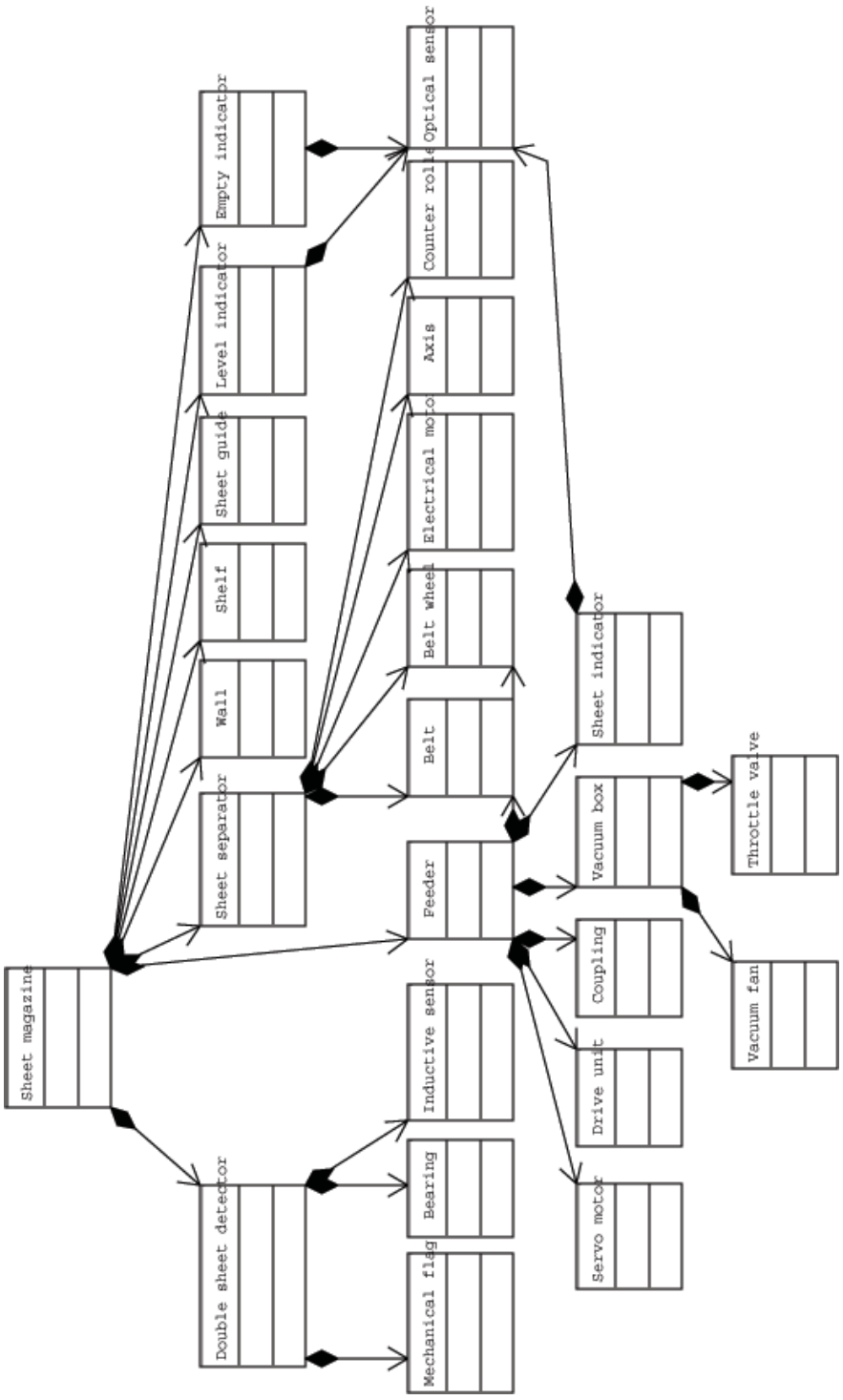


Figure 5.2: An UML diagram showing the hardware hierarchy of the magazine. Drawn by Mattias Wallinius (used with permission).

third is the mclib cam control function. The signals are calculated in an internal state machine supervising the cam states. A visualization of the state machine can be seen in Figure 5.3, it is implemented in ST. The cam control itself has an own state machine, controlled within the mclib function block, described in Section 4.1.

- A task controlling the physical axis. It uses an extremely large axis handling function block to start and supervise it. The axis handling function block contains eight internal state machines controlling various things. The function block is written in ST and all state machines are represented by SWITCH..CASE-statements. The largest one is the main case controller for the axis which steps through all the initializing functions and executes motion control commands. A minor part of the main case controller state machine is illustrated in Appendix A.

  External commands trigger state switches in the main case controller and the function block reacts to the S88-state of the axis, which is part of the output from the cam control. Depending on the S88 state and the internal state machines, the function block produces an S88-command as part of the output, which is passed as input to the cam control.

- A task controlling the virtual axis which serves as master in the cam control. It is started, powered up and then set to spin at a constant speed. The real axis' position depends on the virtual axis' position.

5.3 The Machine Phases

The machine program is designed so that it steps through certain phases, each phase containing S88-states. Figures A.2,A.3 and A.4 in Appendix A show the phases and inner states of TAA.

5.4 The Magazine State Machines

From the list in the previous section it is clear that the magazine software contains many state machines. The topmost one consists of the machine phases. Each phase has an S88 state. Furthermore, the axis handler has eight internal state machines (in the SWICH..CASE fashion), and the task controlling the master axis also has one. The cam control has at least one and even the function block calculating the cam curves has one. It can be mentioned that the cam control's S88-state has no connection to the s88-state of the machine phase. One of the ambitions with designing the magazine software in an object-oriented

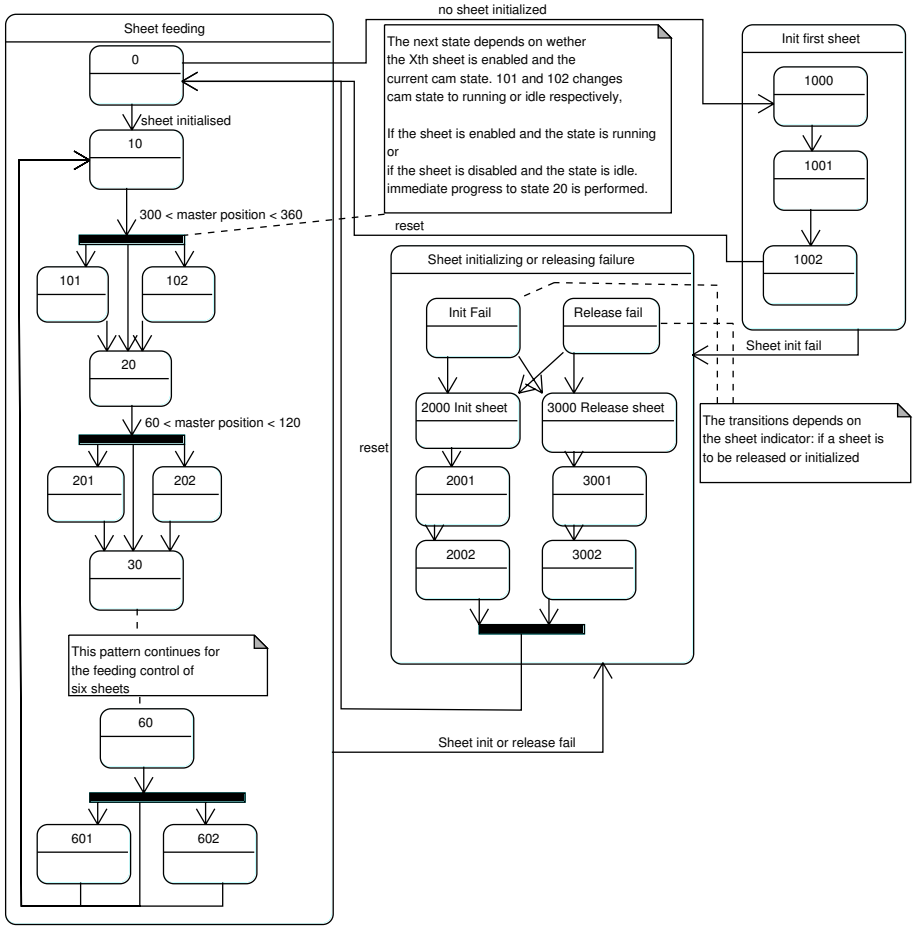


Figure 5.3: An internal state machine (i.e. switch-case statement) from the function block calculating signals for the cam control.

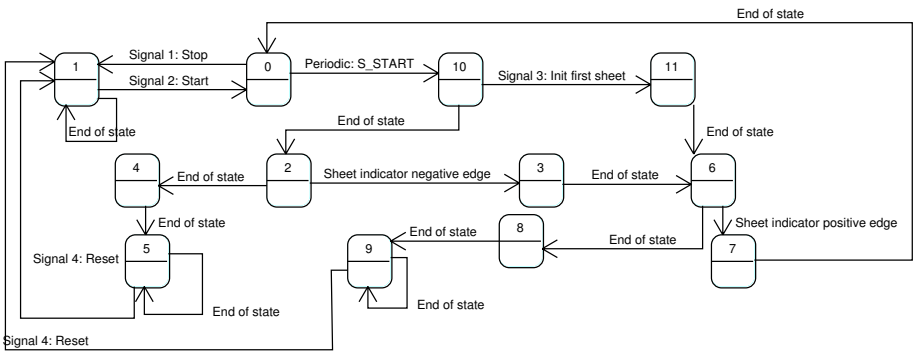


Figure 5.4: The state machine of the cam control. Each state (except from state 0) has an own cam curve.

way has been to simplify the state machines, in some cases by making the states larger and fewer. Another goal is to try and choose the states from an axis-specific and magazine-specific point of view, i.e. avoid adjusting them to standardized interfaces which means nothing to the functionality of the device. The hardware-specific states could then be mapped to the S88-interface, so that the common states still exist to an external observer. The resulting object-oriented design is presented in Chapter 6.

Chapter 6

Design and Implementation

In this section some interesting aspects of the implementation are described and some some benefits of OO-design are demonstrated. It also contains an overview of the packages and most important classes in the magazine model.

Figures of the written Java packages and all classes are enclosed in Appendix A.

The resulting implementation's functionality corresponds roughly to the program described in Section 5.2, leaving out all error handling in the OO-version.

6.1 Solution overview

As explained in Section 1.3, an existing motion control library had to be used in the solution and the connection was made using a Java Native Interface. Figure 6.1 shows an overview of the solution.

A few cyclic tasks were written in Automation Studio, one was to load and start the Java program. Another one with 2ms cycle time took care of mclib function calls.

The communication between the fast cyclic task and the Java program was carried out by a function pointer, which was passed to the JNI-interface. By using that pointer, mclib commands were pushed to the cyclic tasks and by using common structures protected by a semaphore the results could be obtained.

6.2 JNI Interface

The native interface is basically a wrapper around mclib (see Section 4.1). A minor issue was that the Java-implementation was not cyclic and the mclib functions could only be called once a cycle.

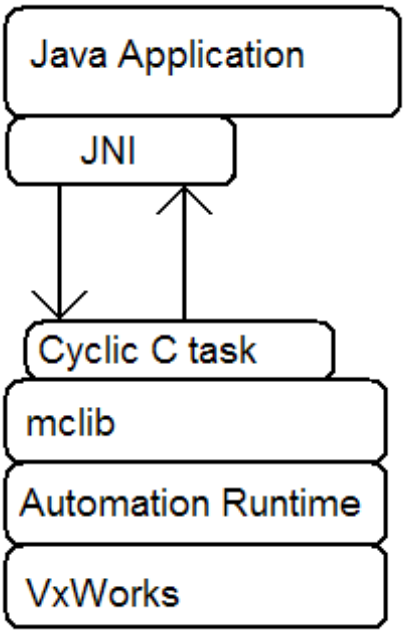


Figure 6.1: An overview of the solution architecture

When a function is called, for instance moveAbsolute, a command is pushed to a fast cyclic task on automation runtime. The cyclic task then calls the mclib-function once every 2ms cycle until the execution is done or an error occurs. The axis' poll thread (explained in Section 6.5) continuously polls the result structure, keeping the thread that called the function waiting. The native structure is protected by a mutex-semaphore[1], using the VxWorks standard library semLib. If an error occurs, or the command is aborted, an exception is thrown to the Java application.

6.3 State Machines

The magazine has an internal state machine, reflecting the state of the ACOPOS servo. It is shown in Figure 6.2. Each state is represented by a Java class, in CamAutControlRun the internal states are represented by internal classes. The classes are part of the package com.tetrapak.a5.rtf.magazine and can be seen in Figure A.10.

In order to correctly map the internal states to the S88-interface, another internal state machine was also implemented, consisting only of an integer, called stateLevel in the Magazine class. The possible values are declared in the

[1] A semaphore that ensures mutual exclusion

MagazineState-class and ranges from AXIS˙OFF to RUN, with an AXIS˙ERROR state as an exception. An UML diagram of how the states are mapped is shown in Figure 6.3.

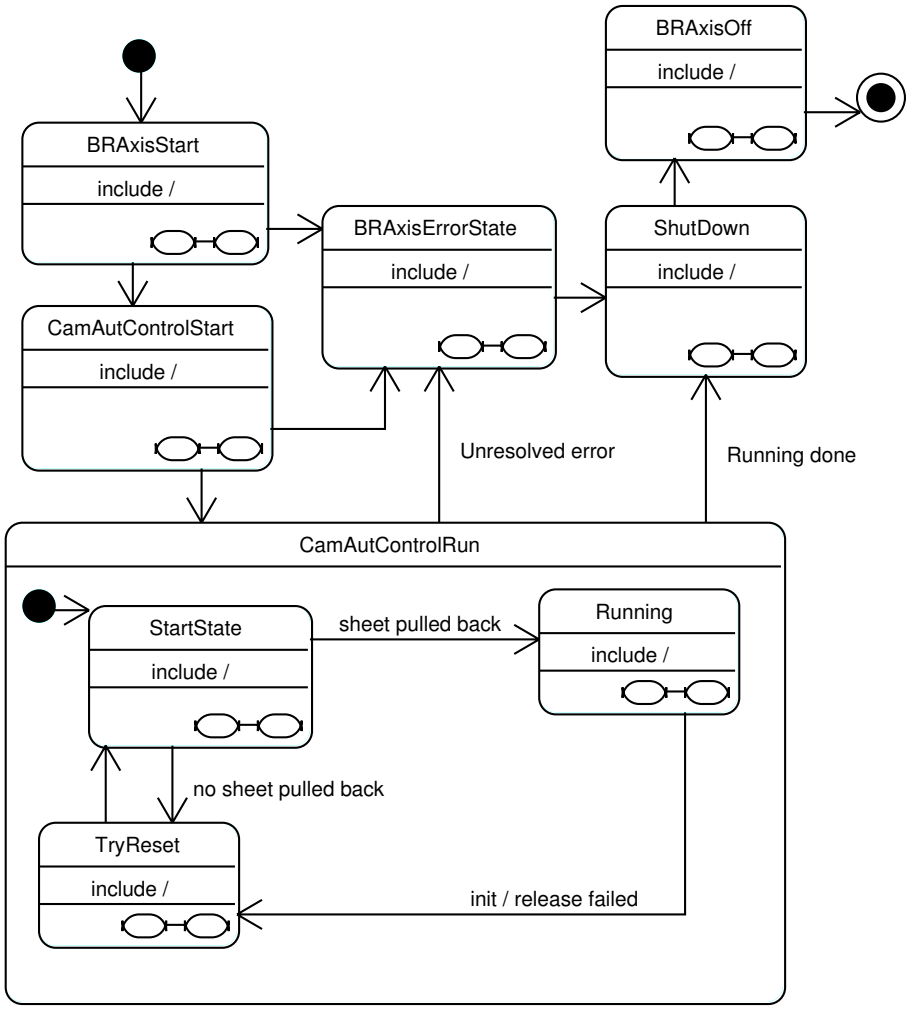


Figure 6.2: State machine for the sheet feeding servo

6.4 Package Overview

An overview of all packages and classes is shown in Figure 6.4. More detailed class diagrams, with methods and attributes, can be seen in Appendix A.

brasruntime contains the classes related to the native code. The class BRASruntime contains a native method to obtain axes references. From the axis addresses instances of the base class BRAxis are created.

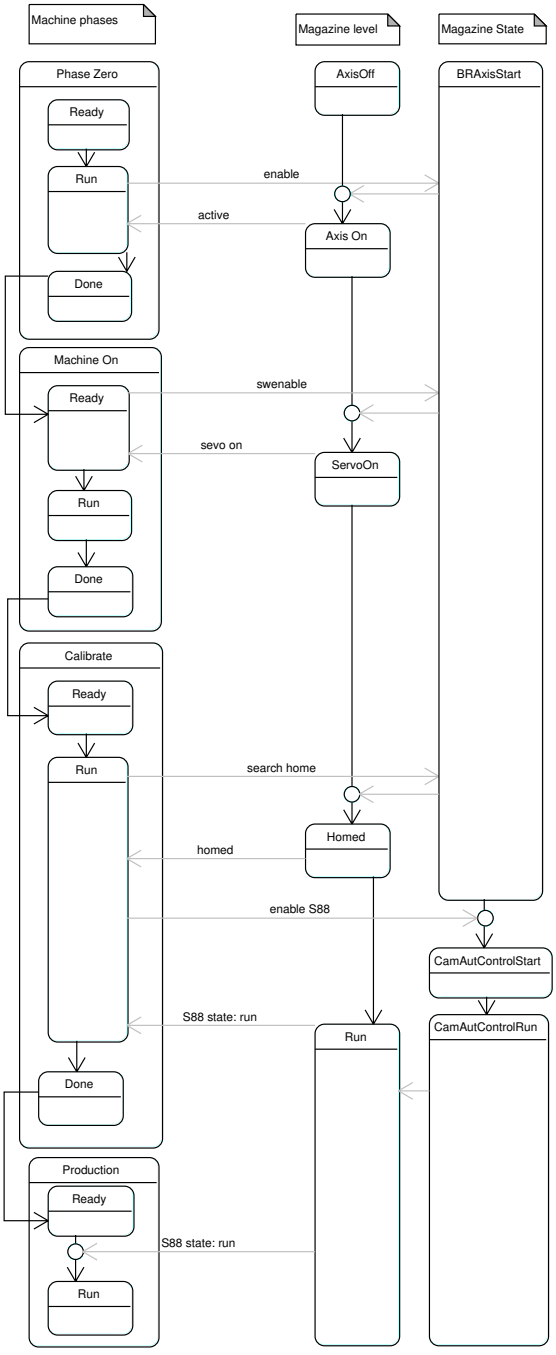


Figure 6.3: Shows the state mapping between the magazine and the machine. The leftmost state machine is an excerpt from the one in Appendix A. The middle is represented by an integer stateLevel and the rightmost is the one from Figure 6.2.

brasruntime.event provides basic functionality for handling events. The Mailbox class acts an observer of any object that fires instances of BRAxisEvent and holds one event at a time.

brasruntime.configuration contains structure-like classes for holding input to motion controlling native functions that often requires many parameters. The class BRAxisConfiguration contains the input to functions that are usually called only once, mostly for axis initializing and configuring. The intention is that the class shall be inherited, and the configuration hard coded into the subclass.

utilities contains a few general classes. The Recorder class writes to a log-file on the PLC flash disk.

com.tetrapak.a5.packagingline.rtf consists of the classes representing the external communication with the magazine. It is simply a matter of starting, enabling functionality, observing and stopping the magazine, although the stopping mechanism is not implemented. The magazine stops itself after a certain amount of time. This package contains the main class RTFStateMachine, and the monitor RTFMonitor is used to communicate with the magazine.

com.tetrapak.a5.packagingline.rtf.magazine This package contains the magazine controlling software, which includes all magazine states and the FeederThread class to execute them. The Magazine class acts as a monitor for communication between the RTFStateMachine and the magazine states.

com.tetrapak.a5.packagingline.rtf.magazineServo is the package containing the axis-classes required to control the magazine. It consists of a virtual MasterAxis and a real CamAxis as slave and their configuration classes. Both axis classes inherit from BRAxis and the configurations from BRAxisConfiguration. There are also a few axis-specific events, which can be fired when the axes are observed.

6.5 Program Flow

The final program flows according to the following list unless an error occurs, in which case the BRAxisErrorState is immediately entered and the magazine shut down.

- The main class is called RTFStateMachine and is part of the rtf package. It creates an instance of the Magazine class and calls startFeeder().

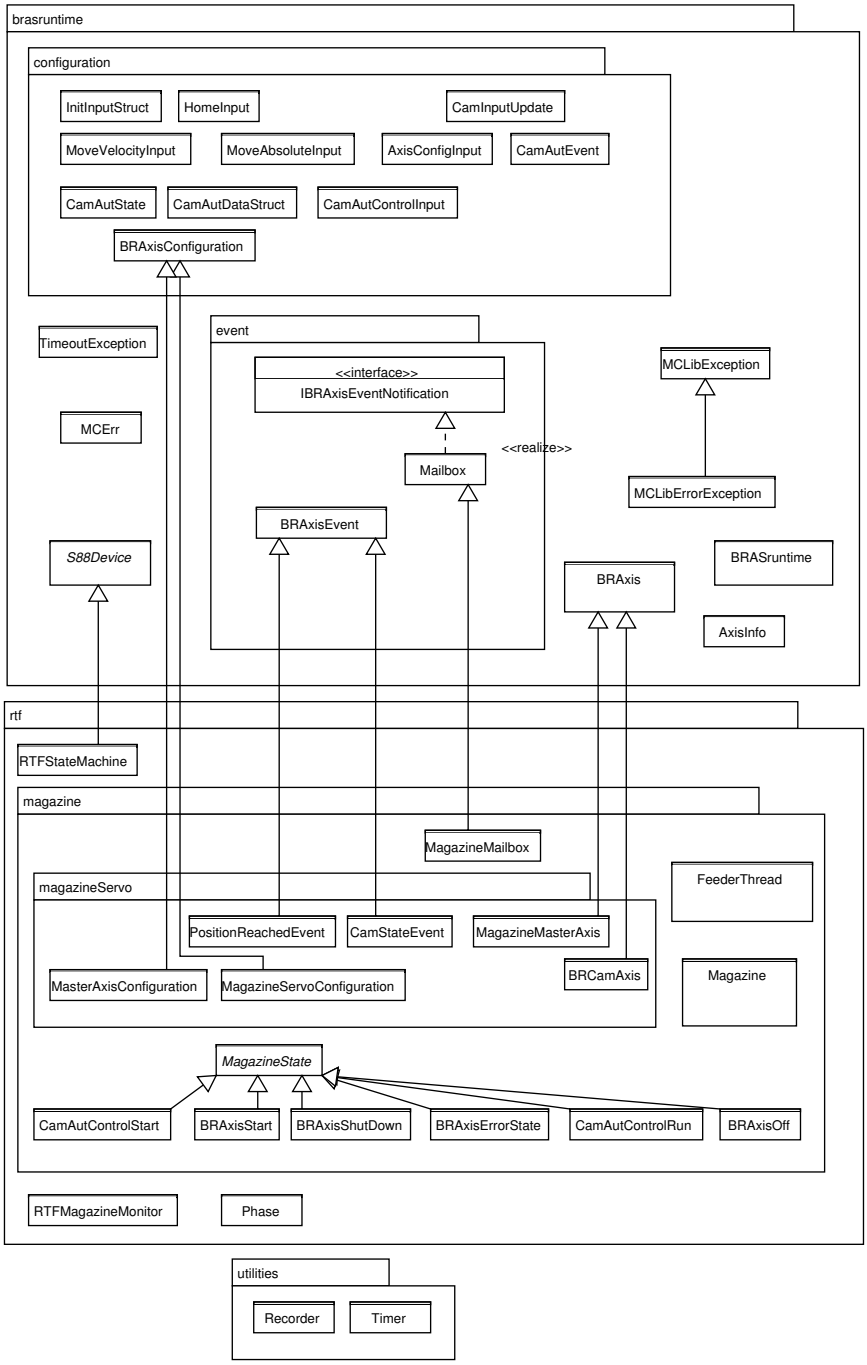


Figure 6.4: Shows an overview of all classes in the magazine application

- The magazine starts a FeederThread which contains a loop to execute the states. The instance of FeederThread first retrieves the axes by calling the getAxes method of the class BRASruntime. Then it creates a CamAxis and a MasterAxis and also the initial state, of type StartState.
- The state execution flows according to Figure 6.2. CamAutControlRun continues for a certain amount of time (if no errors occur) and then transits to the ShutDown state. The FeederThread stops when the MagazineState is BRAxiOff.
- When the magazine is started, the RTFStateMachine simulates the machine behaviour, stepping through the phases described in Section 5.3. In some of the states, it communicates with the magazine, enabling certain values allowing the magazine to proceed and awaiting certain magazine levels. A Figure of the communication in different states is shown in Figure 6.3.

Threads and monitors

There are two controlling threads, plus one polling thread for each axis updating the properties. That makes a total of four threads in the whole application. The threads only communicate through monitors, carefully designed to avoid deadlocks and unnecessary locking. Figure 6.5 shows the monitor communication.

Event communication

During the cam control it is necessary to await certain axis positions and cam state indices (according to the flow in Figure 5.3), and react to changes fast. This behaviour is obtained by using events and awaiting certain conditions to become true. For this the MagazineMailbox, CamStateEvent and PositionReachedEvent classes are used. A sequence diagram of the process awaiting a cam state index is shown in Figure 6.6.

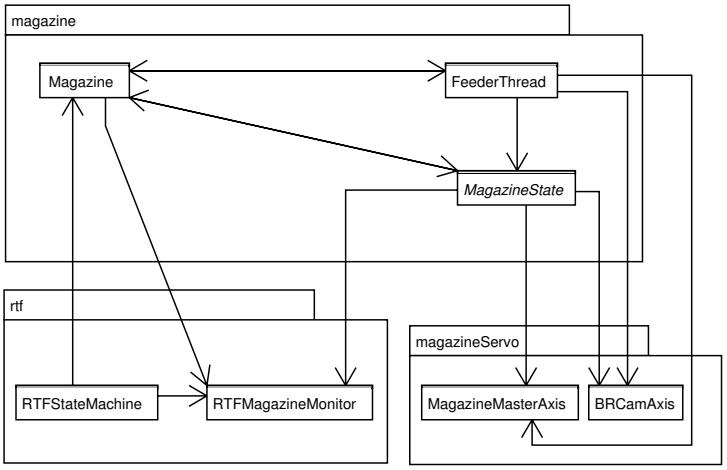


Figure 6.5: Shows the thread communication through monitors. RTFStateMachine and FeederThread are threads. RTFMonitor, Magazine and the Axis classes are monitors.

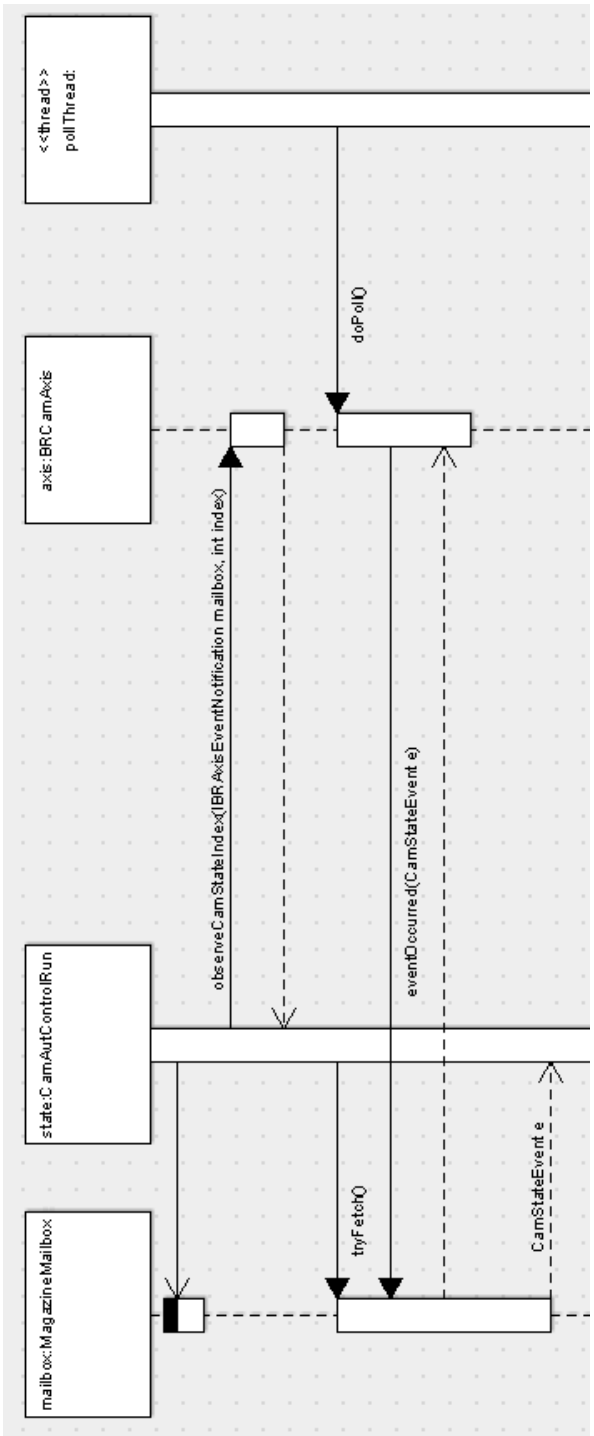


Figure 6.6: Shows the sequence of awaiting a cam state index.

Chapter 7

Performance Testing

7.1 Test Program Flow

The reaction speed of the program was measured by running the following sequence for several speeds:

- The axis starts and initializes by running BRAxisStart.
- The Magazine transits to a TestState which sets the axis to spin at a constant speed.
- A measureStopDistance-function is called in the BRCamAxis class. It awaits a trigger signal, registers the current position, stops the axis, registers the axis position again and returns the distance d_{stop}.
- The distances for each speed are written to the log file.

Because of the PLC open interface (from Section 4.1), the execution of the stop command is performed in several steps. Some of those steps could be performed before the real execution (i.e. stopping of the axis) started. In order to minimize the reaction time those steps were done before, requiring a small modification of the JNI and BRAxis class. Since it was about four steps, four extra calls of the stop function were performed once every $2ms$ cycle, it shortened the reaction time with about $8ms$.

In addition to the ordinary test, a test trying extra allocation of object was performed in order to stress the garbage collector. In this test, 5 new Time objects were created and dereferenced between each test speed. A test started by measuring the stop distance for the lowest speed, then for the next, finishing with the highest speed. Therefore, more garbage memory exist when testing the highest speed.

Table 7.1: Jitter and standard deviations. The notation SS means that the mean value was calculated for the same speeds as for the Cyclic Task (3 different speeds instead of 7). GC means that the test was performed stressing the garbage collector. σ means standard deviation

	Java	Java SS	Java GC	Java GC SS	Cyclic AR task
Mean (ms)	8.8	9.0	8.9	9.1	9.2
σ (ms)	1.13	1.13	1.07	1.13	0.68

Table 7.2: Mean values for certain speeds in ms. Speeds (rows) are measured in units/s.

	Java	Java GC	PLC
300	8.5	8.7	9.1
500	9.3	9.2	9.2
800	9.1	9.4	9.3

A cyclic task was also written to perform the same test directly from within Automation Runtime. However, for this test only 3 different speeds were tested (since it required more work).

7.2 Calculations

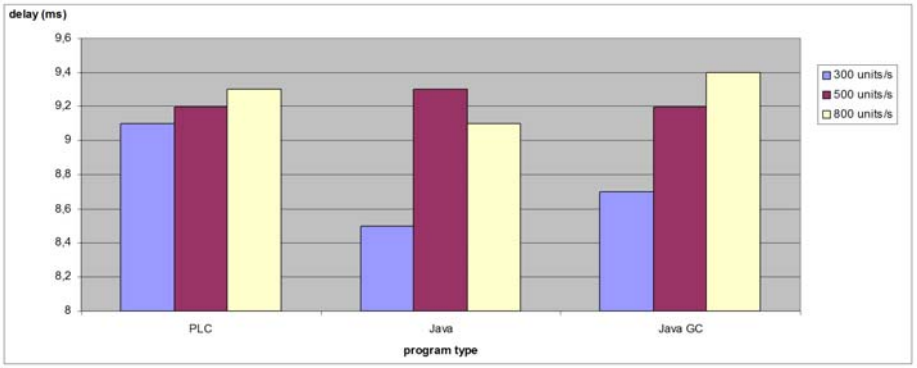
The deceleration distance was calculated and subtracted from the stop distance with the formula for constant acceleration $d_{react} = d_{stop} - \frac{v^2}{2 \cdot a_{dec}}$ where d_{react} is the reaction distance, v the initial speed and a_{dec} the deceleration. d_{react} is the distance that the axis advances before the deceleration starts. The corresponding time is calculated by $t_{react} = \frac{d_{react}}{v}$.

The reaction times were calculated for 7 different speeds: 200, 300, 400, 500, 600 and 700units/s, where one revolution is 360 units. The deceleration was 36000$units/s$. 20 different measurements were performed. The axis position was given as a number of units.

7.3 Results

Table 7.1 shows mean values and standard deviations for the Java applications and the cyclic task in Automation Runtime. Mean values for different speeds are shown in Table 7.2. The mean values for different speeds may also be viewed in Figure 7.3.

Tables of all measurements are enclosed in Appendix B.


delay (ms)
9,6
9,4
9,2
9
8,8
8,6
8,4
8,2
8
PLC
Java
Java GC
program type
300 units/s
500 units/s
800 units/s

Chapter 8

Discussion

In this section the results are discussed and analyzed.

8.1 Vision Fulfilling

In this section the vision from Chapter 1 is compared to the results to see what has been achieved.

Structure, test, model and document applications

During the development of the software the connection between model and implementation has been very strong. At first a model was built, process flows, class diagrams with threads and communication. Since the testing environment became available late in the project, the initial modeling was verbose.

When the implementation began the model changed much. Writing the actual code while rethinking the design generated several improvements. During the third step, explaining the model and presenting the UML-diagrams, even more redesigning took place. The last redesign was mostly done by redrawing the UML, but it was tested directly in the code. Traveling easily between code and schematic pictures made one think of the design and functionality from more than one point of view. Another advantage was that it more or less documented itself - no program code was needed to explain the application.

No unit testing has been done, but the drawback of having a long delay from writing to running made testing a quite painful process. It should be considered though, that this is a first step towards OO-automation and the tools could be fit to suit this kind of development with help from the hardware vendors. The Eclipse remote debugging showed some bugs itself (described in Appendix C), but worked fine most of the time, which made testing very much easier.

Privacy and reuse of code - facilitating portability

Since the software part of this thesis was fairly small the contingencies of reusing code is not totally clear. However, several classes from the Java API were used, in many places in the code, and that can serve as an evidence of reusability. Also, there is the obvious reusing of code in inheritance, for instance in the subclasses of BRAxis.

Another point of view is the portability possibility. If the vendor were to be replaced and another servo were to be used in the same magazine with the same functionality, basically the only classes that would need replacement would be the BRAxis and BRASruntime classes with the JNI. Assuming of course that the cam control works the same way in the new servo.

The protection of methods and attributes in Java was also useful. An example is the RTFStateMachine and RTFMonitor communicating with the Magazine. By declaring the methods in Magazine protected, they are only visible to other classes in the same package (magazine) and to subclasses. That means that the RTFStateMachine is only able to do what it is allowed to do (i.e. start the magazine and observe it's states, not change them) - protecting the programmer from performing illegal variable assignments and method calls. Comparing to the data types of IEC-61131 and the structures of C, this is a great advantage and probably shortens development time.

Separating developers

The separating issue appears to be quite straight forward. One developer could (for instance) write the brasruntime package while another one did the magazine. It is all about communication and package dependencies, designed by the developers. A package may contain many complicated classes and methods - not visible to any class outside the package. It is not necessary to distribute the source code either since the packages can be distributed as JAR-files[1] with neat auto-generated API's attached to it[2]. If Java were to be supported by vendors they could themselves construct basic hardware related packages.

Graphical programming

The figures in this report demonstrate some of the UML benefits when it comes to graphical programming. For example there are class diagrams, as in Figure A.9, state charts as Figure 6.2 and sequence diagrams as Figure 6.6. Tools exist for generating code out of UML-diagrams, and also to create diagrams

[1] Java ARchive, http://en.wikipedia.org/wiki/JAR_%28file_format%29, 2007-02-08

[2] Auto generation of API from Java source: http://en.wikipedia.org/wiki/Javadoc, 2007-02-08

out of existing code (known as reverse engineering). In this project mostly generating code from class diagrams and generating class diagrams out of code have been done, using the program argoUML. Still, that is just for creating skeleton-classes. The method bodies still have to be implemented by writing code.

8.2 Performance Testing

The most time critical parts were performed by the mclib-functions themself and appeared to work fine. Since the performance testing did not contain very much data (due to time limitations in measuring), safe conclusions can not be drawn. However, indications and tendencies can be seen.

It seems like the Java program is increasing the jitter (Table 7.1) and also that the stressing of the garbage collection affected the delay (as seen in Figure 7.3). The overall results can still be considered good though, since the Java was neither especially slow nor significantly increased the jitter. The Java also seemed a little faster than the plc, which was unexpected.

8.3 Further Work

This was a first test of an object oriented automation system. The main issue is that it is not currently supported by hardware vendors. It would be good if vendors would adjust their platforms to support Java and provide standard classes to control their hardware.

A further step would be to connect the HMI (Human Machine Interface) to the Java control application. A thesis performed at the same time at Tetra Pak D&E by Johanna Byrlind and Kajsa Karlsson [10], is about the creation of a new web based HMI. The idea of their thesis is to separate the programmers from designers by using the MVC (Model View Controller) model. This involves techniques such as JSP and the use of tag libraries.

With a web based HMI using applets or servlets, communicating with a Java control application would be rather simple using for instance RMI (Remote Method Invocation).

Bibliography

[1] Aicas. Jamaica VM, 2006-10-01. http://www.aicas.com.

[2] B&R. B&R - Perfection in Automation, 2006-10-01. http://www.br-automation.com.

[3] Don Busch. RTSJ - The Real-Time Specification for Java. Java News Brief, (May), 2006. http://www.ociweb.com/jnb/jnbMay2006.html.

[4] Todd Ham Christie Deizh and Steve Murray. Writing a functional specification for an s88 batch project. Chemical Engineering, (August 01), 2003. Available at http://www.finessephotonics.com/pdfs/CES88.pdf.

[5] Data & Object Factory. Design Patterns in C# and VB.NET - Gang of Four, 2006-09-11. http://www.dofactory.com/Patterns/Patterns.aspx.

[6] The Eclipse Foundation. Eclipse - an open development platform, 2007-01-02. http://www.eclipse.org.

[7] Object Management Group. Unified modelling language, 2007-02-18. http://www.uml.org.

[8] International Electrotechnical Commission. International Standards and conformity assessment, 2007-02-18. http://www.iec.ch.

[9] ISA, Instrumentation, Systems, and Automation Society. S88 for engineers, 2007-02-18. S88 White Paper - Engineers.pdf.

[10] Kajsa Karlsson Johanna Byrlind. New web based hmi portal for tetra pak equipment. Master's thesis, Lund Institute of Technology, 2007.

[11] Sheng Liang. The Java Native Interface, programmers guide and speification. Addison Wesley Longman, Inc, first edition, 1999.

[12] Plcopen organization. IEC 61131-3: A standard programming resource, 2004. http://www.plcopen.org/TC1/Intro˙IEC˙March04.doc.

[13] PLCopen organization. Plcopen - for efficiency in automation, 2006-06-12. http://www.plcopen.org.

[14] Tetra Pak. Tetra Pak. About processing, packaging and aseptic technology., 2006-10-01. http://www.tetrapak.com.

[15] B & R. B&R Automation Software Help, 2002. Automation Studio 2.4.0.9.

[16] The Real-Time for Java Expert Group. rtsj: Real-Time Specification for Java, 2007-02-18. https://rtsj.dev.java.net/.

[17] Tigris.org. subversion.tigris.org, 2007-02-08. http://subversion.tigris.org/.

Glossary

A

ACOPOS Servo drive from B&R.

aicas The company that developed Jamaica. See Section 2.2.

Automation Software Software tools and runtime environment developed by B&R. See Section 4.1.

Automation Studio The development environment for PLC programs to B&R hardware. See Section 4.1.

Axis A rotating device, like the one in figure 5.1.

B

B&R Bernecker & Rainer. The company producing the hardware used in this project. See Section 2.3.

I

IEC 61131-3 A PLC programming standard defining 5 languages. See Section 3.1.

J

Jamaica VM The program used for running Java on the plc. See Section 4.3.

L

Ladder One of the IEC61131-3 programming languages. See Section 3.1.

M

mclib A C-library for motion control written by B&R See Section 4.1.

P

PLC open interface See Section 4.1 .

S

S88 See Section 4.1.

Sequential Function Chart, SFC A programming syntax defined in IEC61131-3. See Section 3.1.

Structured Text One of the IEC61131-3 programming languages. See Section 3.1.

Appendix A

Additional Figures

Existing Magazine Software

axisHandler

Figure A.1 shows a minor part of the function block axisHandler's state machine. Representing SWITCH..CASE statements in this way is not very helpful, and these programs can indeed be fairly difficult to document, other than with the code itself.

Machine Phases

The machine phases are shown in Figures A.2, A.3 and A.4.

Object Oriented Implementation

This section presents UML diagrams of all classes created during this project using Eclipse and Jamaica. The complete software consists of the packages illustrated in the Figures A.5 and forward.

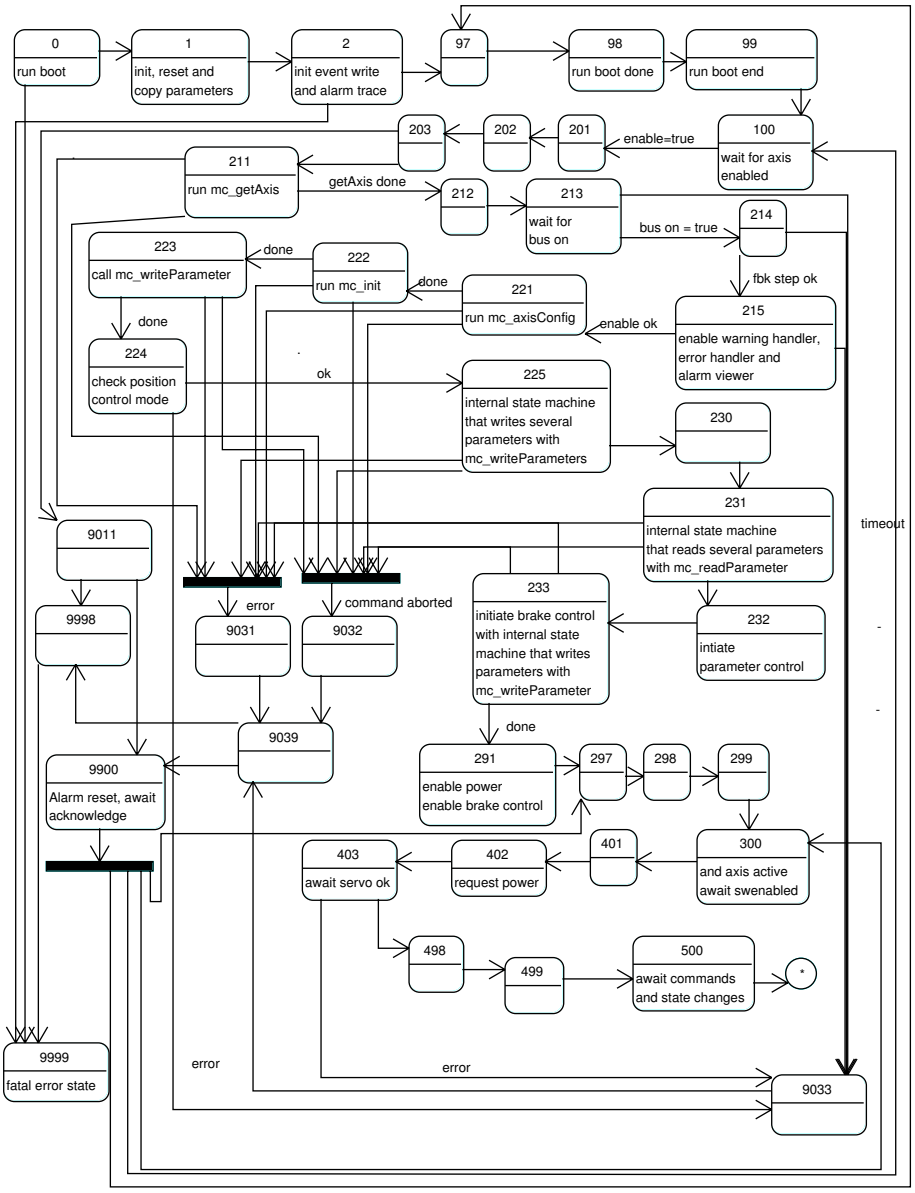


Figure A.1: A minor part of the axis handler state machine. The function block starts in state 0 and the axis is up and running (and awaiting external commands) in state 500.

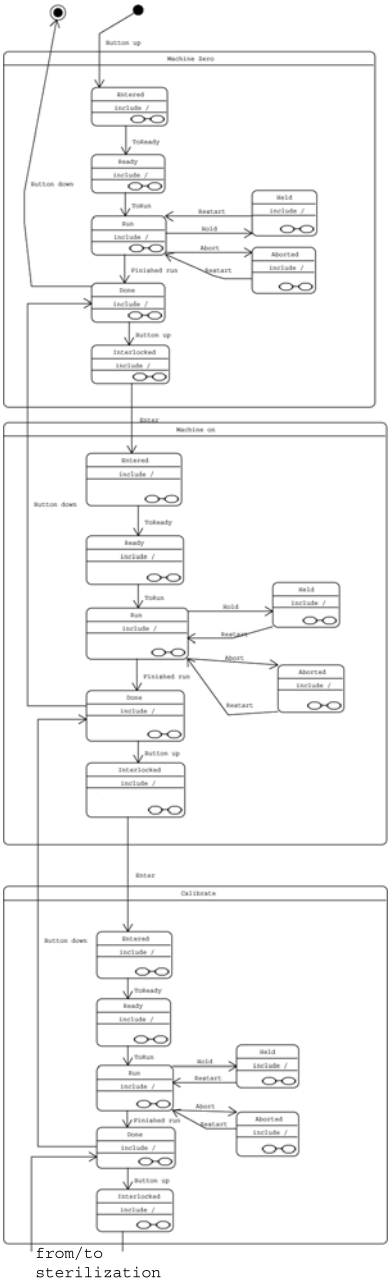


Figure A.2: Machine phases. Continuing in the next figure.

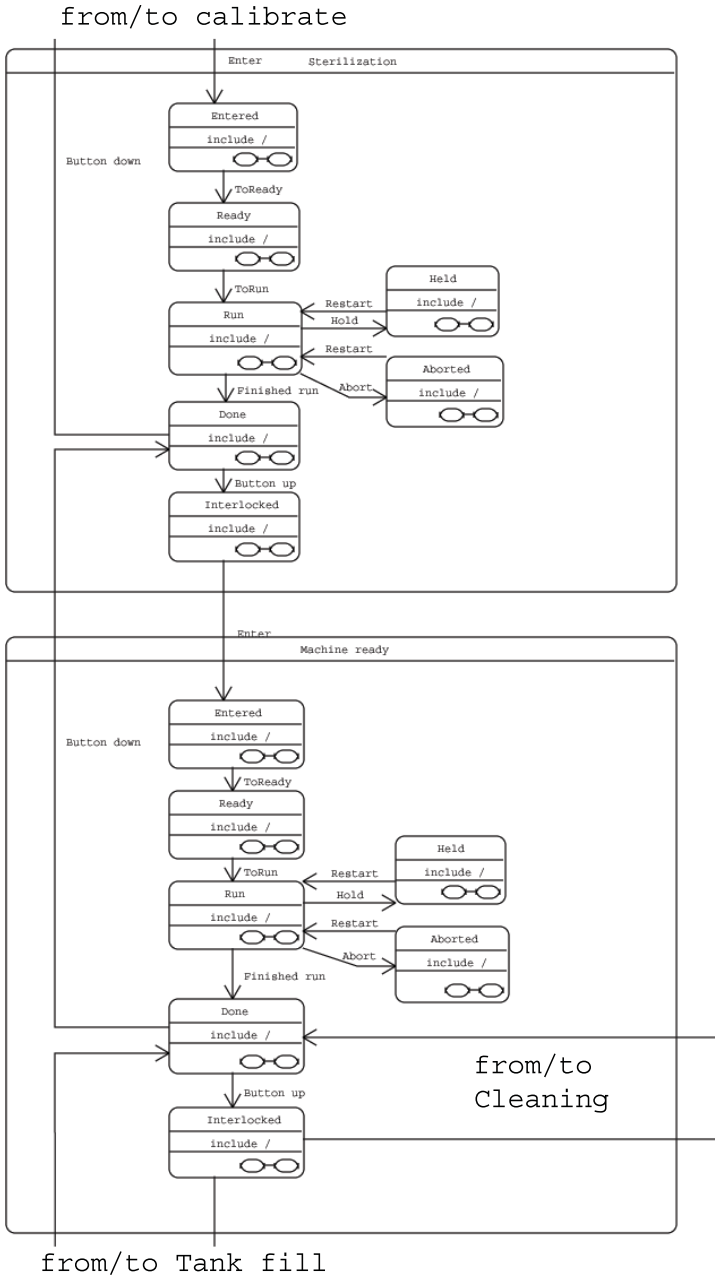


Figure A.3: Machine phases. Continuing in the next figure.

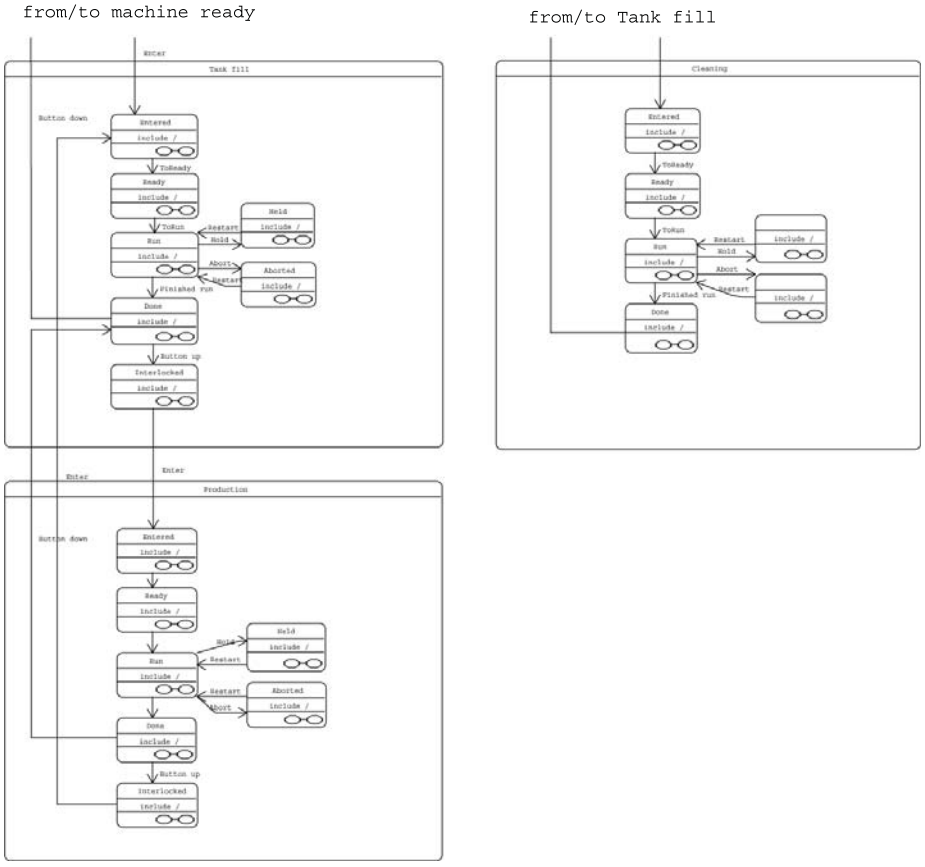


Figure A.4: The first of the machine phases.

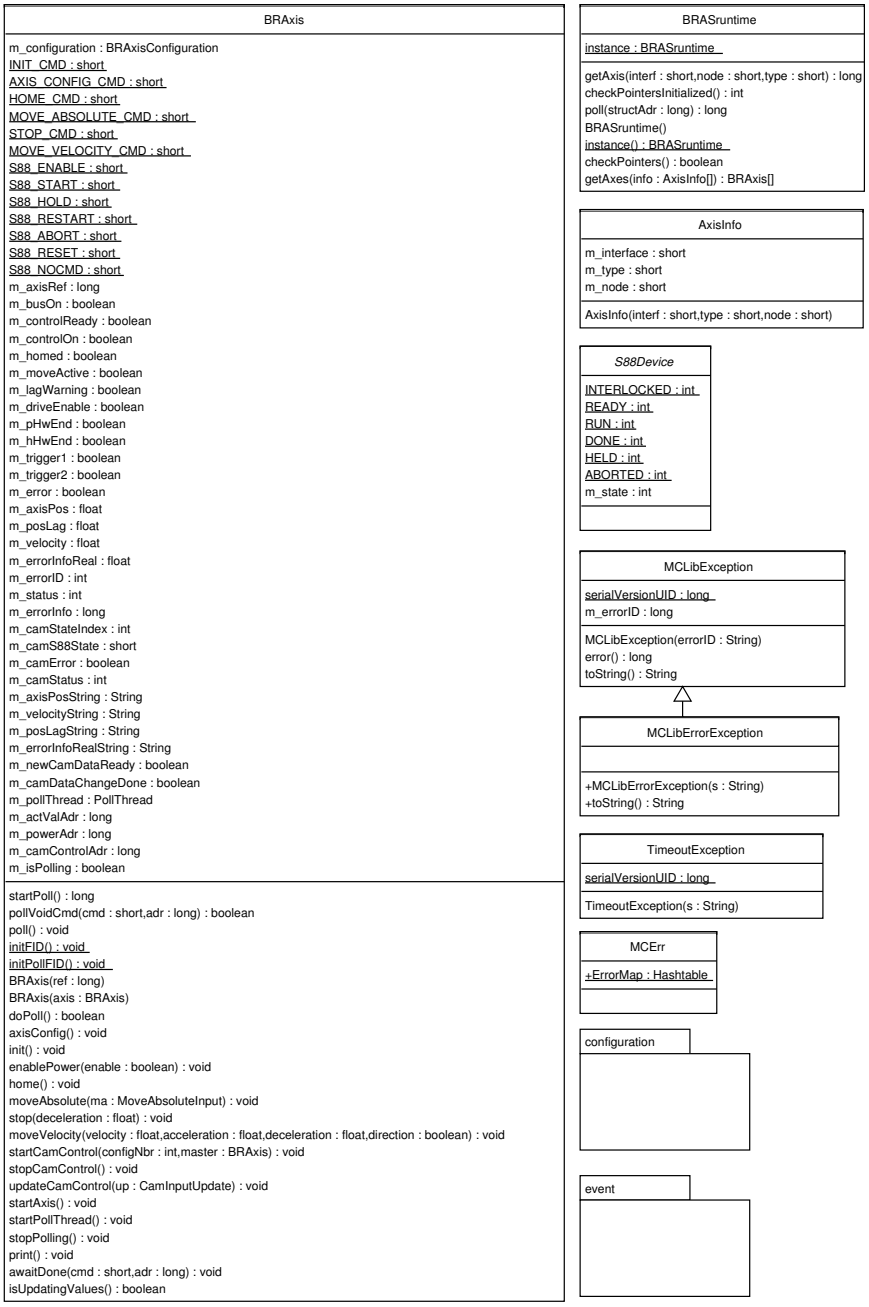


Figure A.5: The brasruntime package providing basic motion controlling functionality

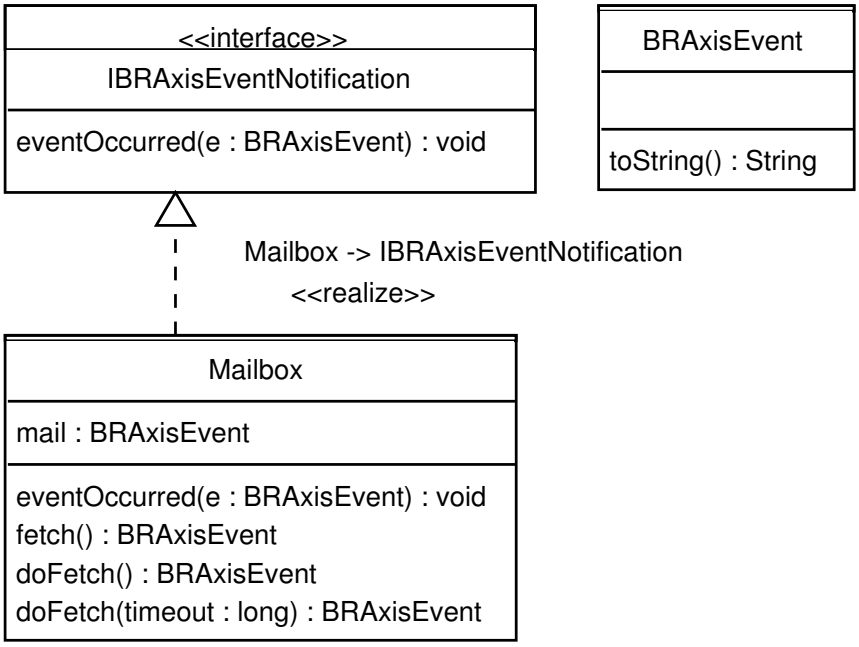


Figure A.6: The brasruntime.event package providing basic event handling

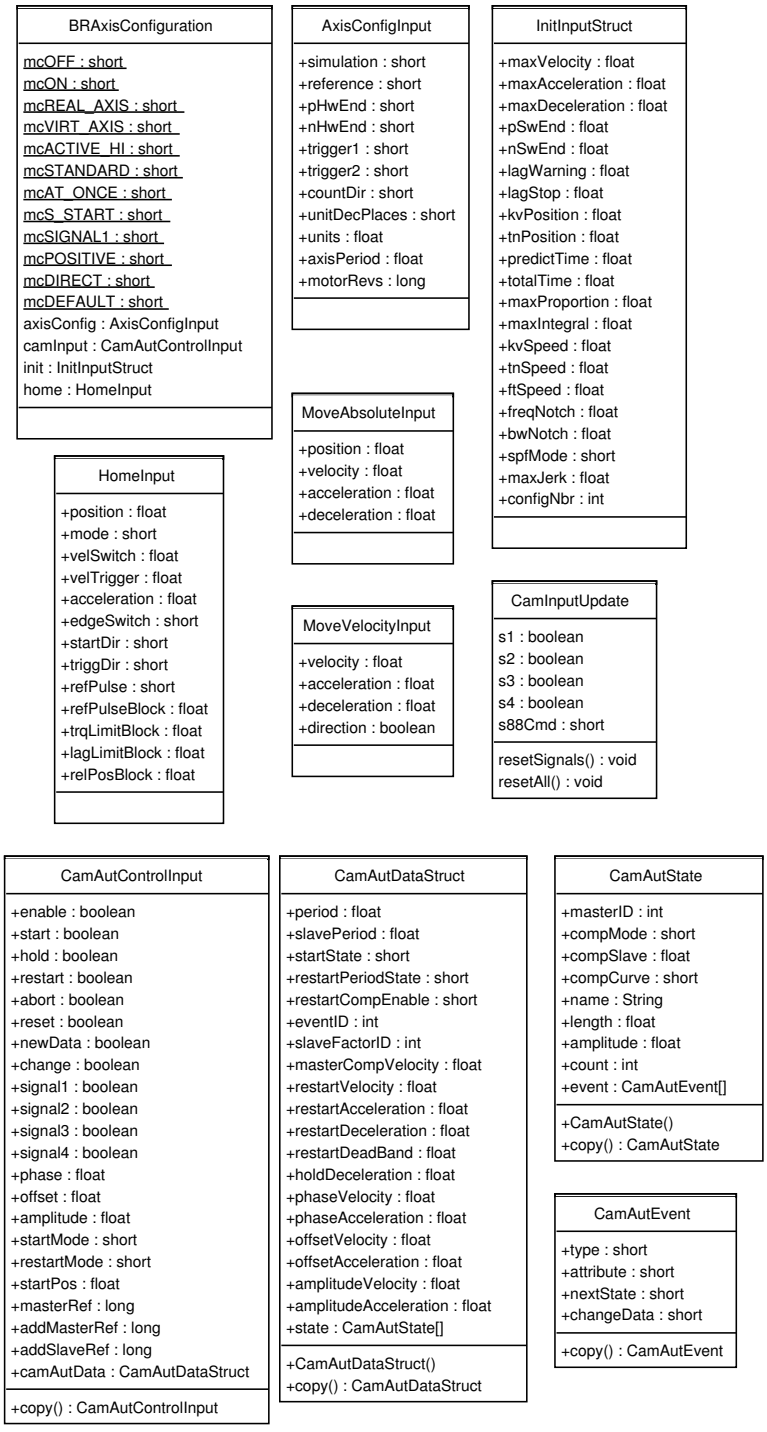


Figure A.7: The brasruntime.configuration package facilitating axis configuration

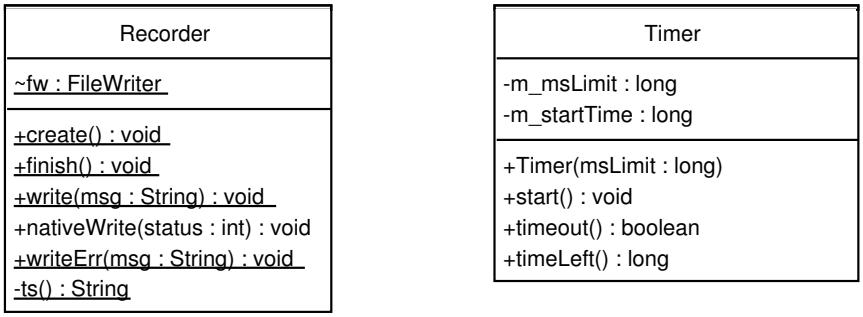


Figure A.8: The utilities package with a few useful classes.

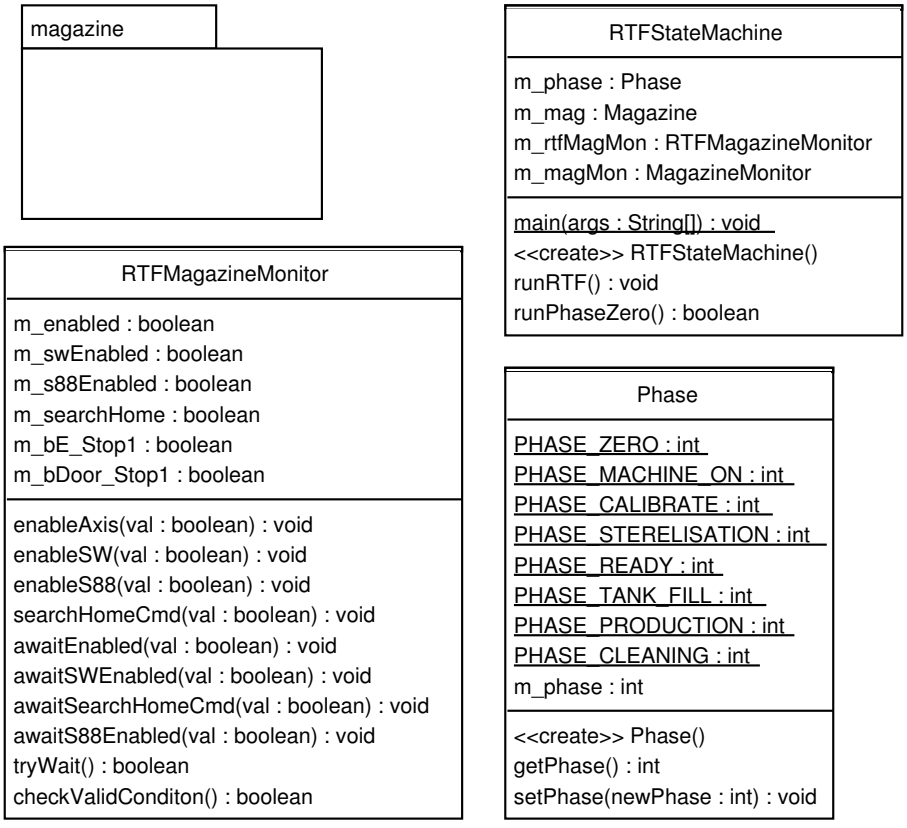


Figure A.9: The com.tetrapak.a5.packagingline.rtf package representing the rest of the machine communicating with the magazine.

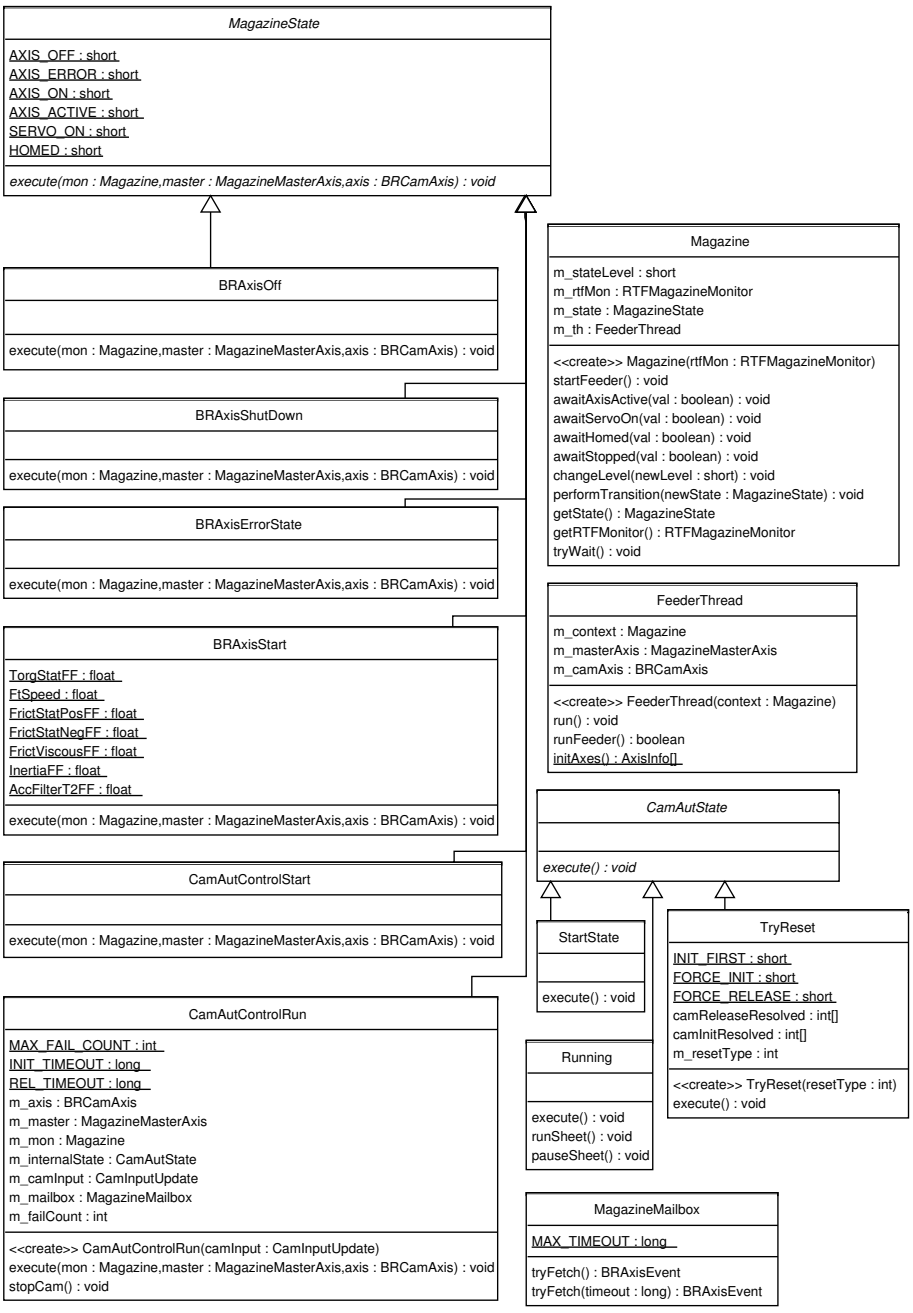


Figure A.10: The com.tetrapak.a5.packagingline.rtf.magazine package containing classes for magazine control.

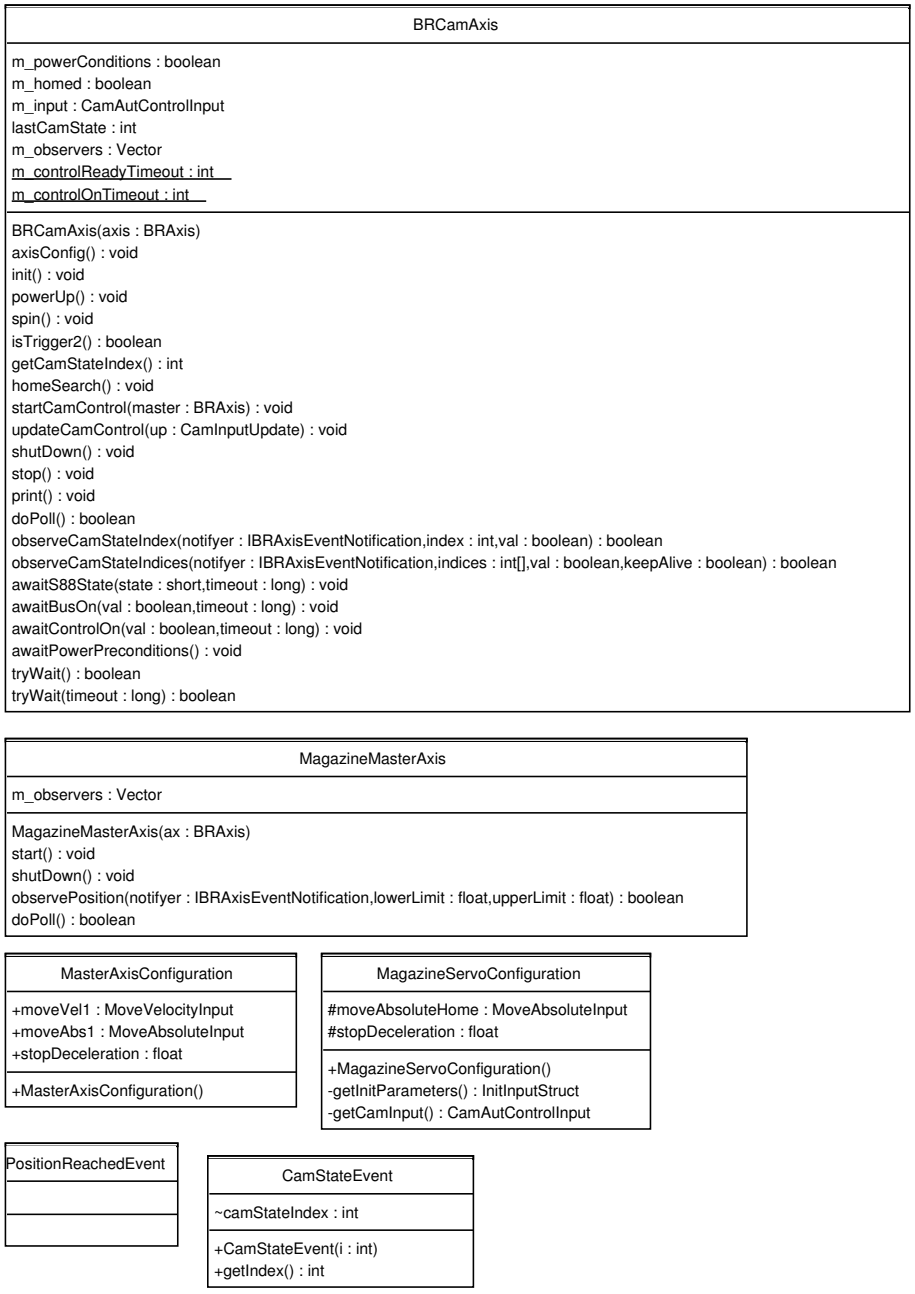


Figure A.11: The com.tetrapak.a5.packagingline.rtf.magazine.magazineServo package containing axis-classes for the magazine servo.

Appendix B

Test Results

This chapter presents the measurements from tests described in Chapter 7. Table B.1 shows the stop delay in ms for the ordinary Java program, allocating as little new objects as possible every test run. Table B.2 shows the stop delay in ms when 5 extra new-calls where added to the functions. The new-calls instantiated Time-objects which were dereferenced again as soon the test was over. The tests were performed immediately after one another, starting with the lowest speed and increasing. Table B.3 shows the corresponding values when testing on the PLC not using the Java program.

Table B.1: Stop delay in ms. The columns are speed in units/s and the rows are different test runs.

	200	300	400	500	600	700	800	mean
1	7.2	9.2	9.4	9.1	10.0	8.8	8.9	8.9
2	7.2	9.2	9.4	9.1	8.3	10.3	7.6	8.7
3	7.2	5.8	9.4	11.1	8.3	8.8	10.1	8.7
4	7.2	5.8	9.4	9.1	10.0	10.3	8.9	8.7
5	7.2	9.2	9.4	9.1	10.0	8.8	8.9	8.9
6	7.2	9.2	9.4	9.1	8.3	7.4	7.6	8.3
7	7.2	9.2	9.4	11.1	10.0	8.8	8.9	9.2
8	7.2	9.2	9.4	9.1	10.0	8.8	10.1	9.1
9	7.2	9.2	9.4	9.1	10.0	10.3	8.9	9.2
10	7.2	9.2	9.4	7.1	10.0	8.8	10.1	8.8
11	7.2	9.2	9.4	9.1	8.3	7.4	10.1	8.7
12	7.2	9.2	9.4	11.1	8.3	8.8	8.9	9.0
13	7.2	9.2	9.4	9.1	10.0	8.8	8.9	8.9
14	7.2	9.2	9.4	9.1	10.0	10.3	8.9	9.2
15	7.2	5.8	6.9	9.1	10.0	8.8	8.9	8.7
16	7.2	9.2	6.9	7.1	8.3	8.8	10.1	9.2
17	7.2	9.2	9.4	9.1	8.3	8.8	8.9	8.7
18	7.2	9.2	9.4	11.1	10.0	8.8	8.9	9.2
19	7.2	9.2	9.4	9.1	8.3	8.8	8.9	8.7
20	7.2	5.8	9.4	9.1	10.0	10.3	8.9	8.7
mean	7.2	8.52	9.15	9.3	9.3	9.0	9.0	

Table B.2: Measured stop delay in ms with allocation and dereferencing of 5 new Time-objects during each test. The columns are speed in units/s and the rows are different test runs.

	200	300	400	500	600	700	800
1	7.2	5.8	9.4	11.1	10.0	10.3	10.1
2	7.2	5.8	6.9	9.1	10.0	10.3	10.1
3	7.2	9.2	9.4	7.1	10.0	8.8	10.1
4	7.2	9.2	9.4	9.1	10.0	8.8	8.9
5	7.2	9.2	9.4	11.1	10.0	8.8	8.9
6	7.2	9.2	9.4	11.1	8.3	8.8	8.9
7	7.2	9.2	9.4	9.1	8.3	10.3	8.9
8	7.2	5.8	9.4	9.1	10.0	8.8	10.1
9	7.2	9.2	6.9	9.1	8.3	8.8	8.9
10	7.2	9.2	9.4	9.1	10.0	8.8	8.9
11	7.2	9.2	9.4	9.1	10.0	8.8	8.9
12	7.2	9.2	9.4	9.1	8.3	10.3	8.9
13	7.2	9.2	9.4	11.1	10.0	8.8	10.1
14	7.2	9.2	9.4	9.1	8.3	8.8	8.9
15	7.2	9.2	9.4	7.1	10.0	8.8	10.1
16	7.2	9.2	9.4	9.1	8.3	8.8	8.9
17	7.2	9.2	9.4	9.1	10.0	10.3	8.9
18	7.2	9.2	9.4	9.1	10.0	10.3	10.1
19	7.2	9.2	6.9	9.1	10.0	10.3	8.9
20	7.2	9.2	9.4	7.1	8.3	8.8	10.1

Table B.3: Corresponding measured stop delays for a cyclic Automation Runtime task in ms. The columns are speed in units/s and the rows are different test runs.

	300	500	800	mean
1	9.4	9.3	9.8	9.5
2	9.8	8.5	9.3	9.3
3	9.8	8.6	9.3	9.3
4	8.5	8.6	9.8	9.0
5	8.5	8.5	9.8	9.0
6	8.5	9.7	8.6	9.0
7	9.8	9.3	8.6	9.3
8	9.3	9.7	8.6	9.3
9	9.8	9.7	8.6	9.4
10	9.3	9.4	8.6	9.1
11	9.8	9.8	8.6	9.2
12	8.5	9.3	9.8	9.3
13	8.5	8.5	9.8	9.0
14	8.5	8.5	9.4	8.9
15	9.8	8.5	9.4	9.3
16	9.8	8.5	9.8	9.3
17	9.3	8.6	9.8	9.3
18	9.8	9.8	9.4	9.7
19	8.5	9.3	9.8	9.3
20	5.9	9.7	9.8	8.5
mean	9.1	9.2	9.3	

Appendix C

Implementation Methods and Issues

It was not obvious what parts of which software tools to use and how to use them. This chapter describes the specific methods and tools, issues, problem solutions and reasons. It will probably be of interest only for those who wants to try out a similar problem themselves with similar software and hardware tools.

C-compiler issues

The mclib is compiled with B&R's gnu C-compiler, but it was not compatible with Jamaica source code, due to optimizations made by B&R. Therefore, another one had to be used to compile the Jamaica C-code. Aicas recommended the compiler used in Tornado from Wind River, the former software development environment for VxWorks. A smaller issue was that the Tornado environment was no longer available, the present one is called Workbench.

Running Jamaica on target

The plan is to write a small C-program, which runs as a cyclic task from within Automation Studio. The small C-program's only task is to start the standalone pre-built Jamaica application. The Jamaica application obtains a pointer to a pusher function, so that it can push mclib commands to a cyclic task.

The -includeClasses option

To make the Jamaica JNI-implementation aware of all the classes used, the -includeClasses option must be included into the configuration. All classes that

is to be reached from the JNI, but do not contain any own native methods have to be mentioned.

File system

B&R target uses a FAT file system and provides the IO-library to manage files. The CPU can be configured to include "File Devices" - a special path on target with an assigned device name. The external modules were transferred with ftp to such a device and could then be loaded in AS.

Eclipse Remote Debugging

A few strange behaviours were observed during the Eclipse remote debugging. Surely some of them origin in incorrect behaviour of the user.

- Presenting variable values: Sometimes true boolean values were presented as false, and float- and double values were always zero. This was solved by presenting them as Strings, which were always shown correctly.
- When having many breakpoints (especially in the vicinity of native methods), the debugger got disconnected. The program appeared to be still running on the target though, stopping at breakpoints as usual and had to be interrupted by force on target.