Automation for Standardisation and Blending Systems

Christian Isaksson, Andreas Månsson

Department of Industrial Electrical Engineering
and Automation
Lund University

in co-operation with

Tetra Pak Dairy & Beverage Systems AB

Advisors

Lund University

Gustaf Olsson
Gunnar Lindstedt

Tetra Pak Dairy & Beverage Systems AB

Fredrik Gunnarsson
Magnus Önnheim

Lund
2006

Department of Industrial Electrical Engineering and Automation
Lund University
Box 118
SE-221 00 Lund
Sweden

http://www.iea.lth.se/

CODEN:LUTEDX/(TEIE-5221)/1-58/(2006)

Christian Isaksson and Andreas Månsson, 2006

Printed in Sweden by Tryckeriet i E-huset
Lund University
Lund, 2006

Abstract

The purpose of a standardisation and blending system is to produce milk products after a certain recipe with a uniform composition, e.g. with a uniform fat and/or protein content. To get high accuracy in the outgoing product, high accuracy is needed in all parts of the process; i.e. sensors, actuators and calculations. During recent years more complex compositions with more ingredients are wanted which puts further requirements on the control system.

The combination of high performance and low price of SoftPLCs make them interesting. There is however some questions about the reliability of industrial PCs and with that SoftPLCs. The standardisation and blending process requires high performance but also high reliability. The SoftPLCs undoubtedly have higher performance but it is the combination of sufficient performance and sufficient reliability that is the optimal solution. To achieve higher reliability with SoftPLC the industrial PC should not have any moving parts, e.g. hard drives or fans. An UPS should also be used to maintain service even during short power failures.

Both PLCs and SoftPLCs were investigated to see if they could be used for the standardisation and blending process. The investigation analysed each company's control system according to some specified criterions, e.g. performance, memory capacity and I/O-possibilites. Larger companies who's control systems already were used within Tetra Pak were the main candidates; i.e. Rockwell, Siemens and ABB (which was used for the current version of the standardisation machine). The smaller companies B&R and Beckhoff was also investigated (though only their SoftPLC alternatives). Here ABB was considered to have the best PLC while Siemens considered to have the best SoftPLC.

Since the current machine has a limitation of the number of ingredients that can be mixed in the program code, one task was to develop and implement a program structure without this limitation. This was done with a decentralised calculation principle where the calculations that involve a certain ingredient is moved to the program controlling that ingredient instead of being centrally located in the program. The program structure was also implemented together with a process simulation in an ABB system.

Also, more complex controller structures was investigated. This was because many ingredients may cause that the system becomes less robust. To increase robustness an alternative controller structure was proposed, based on local compensation in each ingredient.

The user interface of the current machine was analysed according to some usability theories. Some suggestions for improvements were given.

Preface

This master thesis was written as the final part of a Master of Science education at the Lund University. The authors come from a background in Electrical Engineering with special interest in automation, automatic control, signal processing, electrical measurements and programming. This mix of knowledge attracted Henrik Jönsson, Tetra Pak Dairy & Beverage AB, in fall of 2005. We thank Henrik for this opportunity.

The goal was to do a pre-study for further development of a blending system. Together with our advisors at Tetra Pak and LTH, a work plan was drawn up with some points that were to be analysed and investigated. During the progress of the work, some of these points developed and became more important while other were given less priority. However, in the end we feel that a good overview of many areas have been covered with more in-depth analysis in some special areas.

We would specially like to thank our advisors: Gustaf Olsson for his ever lasting enthusiasm, Gunnar Lindstedt for his helpfulness, Fredrik Gunnarsson for his openness and last but not least Magnus Önnheim for his invaluable help with everything from process knowledge to program understanding. Other persons that have had an imprint on this thesis are Alf Lönsjö and Tommy Carlsson at Tetra Pak. We would also like to thank our friends at *the bench* for the years at LTH.

Early summer 2006, Lund

Christian Isaksson and Andreas Månsson

Table of Contents

Preface iii

1 Introduction 1
- 1.1 Aim . . . 1
- 1.2 Approach . . . 2
- 1.3 Structure of the thesis . . . 2

2 Process description: Standardisation and blending 3
- 2.1 Standardisation . . . 3
  - 2.1.1 Manual standardisation . . . 3
  - 2.1.2 Inline standardisation and blending . . . 4
- 2.2 Future requirements from the process . . . 9

3 PLC – SoftPLC comparison 11
- 3.1 Operation reliability . . . 11
  - 3.1.1 Real-time operation . . . 11
  - 3.1.2 Moving parts . . . 14
  - 3.1.3 Power supply . . . 15
  - 3.1.4 Redundancy . . . 15
- 3.2 Performance . . . 15
- 3.3 Summary . . . 17

4 Investigation of control systems 19
- 4.1 Choosing criterions . . . 19
- 4.2 Priority analysis . . . 20
- 4.3 I/O specification . . . 21
- 4.4 Investigated systems . . . 21
  - 4.4.1 Rockwell Automation . . . 22
  - 4.4.2 Siemens . . . 23
  - 4.4.3 ABB . . . 24
  - 4.4.4 B&R . . . 25
  - 4.4.5 Beckhoff . . . 25
- 4.5 Choice of system . . . 25

5 Program structure and implementation 29
- 5.1 Decentralised calculation principle . . . 29
  - 5.1.1 Flow calculations . . . 30

5.2 Recipe management . . . 31
5.3 Overview of the program structure . . . 31
5.4 ABB information . . . 32
5.4.1 Control modules – optimised function blocks . . . 32
5.4.2 Connection to user interface . . . 33

6 More complex controller structures **35**
6.1 Control of the total flow rate . . . 35
6.2 Control strategies . . . 36
6.2.1 Focus on mixing composition . . . 36
6.2.2 Focus on robustness . . . 36

7 User interface **37**
7.1 Theory . . . 37
7.1.1 Usability . . . 37
7.1.2 Heuristic evaluation . . . 38
7.1.3 Cognitive walkthrough . . . 40
7.2 Analysis of current system . . . 42

8 Summary and conclusions **49**
8.1 Future work . . . 50

Bibliography **51**

Glossary **53**

A Dairy introduction: From raw milk to packaged product **55**
A.1 Composition of milk . . . 55
A.2 Reception . . . 56
A.3 Heat treatment . . . 56
A.4 Separation . . . 56
A.5 Standardisation and blending . . . 57
A.6 Homogenisation . . . 58
A.7 Packaging . . . 58

Chapter 1
Introduction

For most people milk is just a white liquid served chilled in a glass. Other examples of milk products are whipped cream, yoghurt, chocolate milk and cheese. One difference between the milk products is the percent of fat. Fat standardisation is a useful operation in a dairy, guaranteeing an uniform product quality and allowing surplus fat to be used for more fat needing products. In addition to this, protein standardisation and blending with different flavours, such as chocolate or strawberry, has become more important during recent years as the dairy product range has grown and the amount of ingredients has increased. Together with future requirements for additional ingredients, higher flexibility and performance of the control system is demanded.

If the reader does not have experience within dairy processing, you are advised to read through appendix A for a general dairy introduction. A number of technical words and abbreviations will be used throughout the thesis. These are explained in the glossary.

1.1 Aim

The main goal of this thesis is to give a foundation for the choice of a modern control system for development of complex systems. To be able to apply the investigation to a concrete application, the standardisation and blending task is used because of its complex nature and its requirements on the control system. Other goals are to show how a flexible program structure can be built, investigate the use of more complex controller structures and review the operator interface.

A new trend in modern control systems that is discussed in this thesis is PC-based control systems, also known as SoftPLC. This has become more and more interesting because of the combination of increased reliability, high performance and lower price of PCs.

The thesis tries to answer the following questions:

- SoftPLC – Is SoftPLC an option that could (and should) be used?
- Control system – What control system should be used?
- Flexible structure – How can a flexible program structure be built and how is it implemented in the PLC?
- Complex controller structures – How can more complex controller structures be used to get an optimised blending?
- User Interface – How is the current user interface design and what should be different?

1.2 Approach

To be able to choose one control system, many have to be investigated. A wide base analysis of control systems is done, for both PLCs and SoftPLCs. However, since SoftPLC only is a software, the hardware platform is only discussed in general with industrial PCs versus conventional PLCs. The different PLCs are however discussed in more detail.

A flexible structuring of the program is approached with the goal to get a more decentralised calculation. This structure is then implemented in a single control system.

Complex controller structures are investigated from the decentralised program structure and how it could be used to increase controller robustness.

There are many different theories and procedures for design of user interfaces. To get some knowledge about these, people within LTH are contacted for discussion and the theories are studied. An evaluation of the current user interface is then done with a critical eye and some suggestions on changes and improvements are given.

1.3 Structure of the thesis

Chapter 2 Description of the standardisation and blending process.

Chapter 3 Comparison between PLC and SoftPLC with regard to reliability and performance.

Chapter 4 Investigation of different control systems, both SoftPLC and PLC.

Chapter 5 A program structure allowing flexibility and easy multiplication of functions is developed and implemented in one system.

Chapter 6 Complex controller structures and how they can increase controller robustness are investigated.

Chapter 7 The user interface of the current application is analysed and some suggestions on improvements are given.

Chapter 8 Summary and final discussions.

Chapter 2

Process description: Standardisation and blending

This chapter describes the standardisation and blending process. In this thesis the word standardisation is primarily meant as the concept of mixing together skimmilk and cream to get an uniform fat content in the final product. The word standardisation can also be used for protein and SNF standardisation but if nothing else is said, standardisation is equal to fat standardisation. Common for all standardisations is that the content of some substance in the final product is interesting.

The word blending is meant as mixing with more ingredients like flavours or vitamins. This is usually done by adding additional flows after the remix of cream. Such an additional flow is commonly called an additive. These ingredients probably contain fat and/or protein and will influence the standardisation calculation. They will however not be the active component in the standardisation. Instead, they will be mixed into the flow after a certain recipe, most commonly written as a ratio of the outgoing flow or a fix flow. However, additives can also be used for standardisation purposes which are described in the chapter.

2.1 Standardisation

The process of standardising milk has its origin in separator sales. A separator divides the raw milk into two main components, cream and skimmilk. To get standardised milk with a lower fat content than the raw milk, a mix of cream and skimmilk is needed. It is then possible to get a large variety of milk fat contents without having to change anything in the separator. One way of doing this is by manual mixing but it is more effective when done inline. Both ways are presented here.

2.1.1 Manual standardisation

Manual mixing of cream and skimmilk is calculated by using fat and mass balances. With this, the amount of cream and skimmilk needed to get a given amount of standardised milk can be calculated. The mass of cream and skimmilk into the system have to be equal to the mass of standardised milk out of the system. Similarly, the mass of fat in has to be equal to the mass of fat out. This can be illustrated in equation (2.1) and (2.2) where C is the mass of cream (in kg), S the mass of skimmilk, M the mass of standardised milk, f_C the fat content in

the cream (between 0 and 1), f_S the fat content in the skimmilk and f_M the fat content in the standardised milk.

$$Cf_C + Sf_S = Mf_M \tag{2.1}$$

$$C + S = M \tag{2.2}$$

If f_C, f_S, f_M and M are known, the mass of needed skimmilk, S, is calculated as (2.3) and the mass of needed cream, C, is then calculated as $M - S$ or by using Equation (2.4). An example of the calculations is given in Example 2.1.

$$S = \frac{M(f_C - f_M)}{f_C - f_S} \tag{2.3}$$

$$C = \frac{M(f_M - f_S)}{f_C - f_S} \tag{2.4}$$

Example 2.1: Manual mixing using equations
$f_C = 0.40$, $f_S = 0.0005$, $f_M = 0.03$ and $M = 100$ kg. The amount of skimmilk, S, and cream, C, needed are calculated by use of (2.3) respectively (2.4):

$$S = \frac{100(0.40 - 0.03)}{0.40 - 0.0005}\ \text{kg} = \frac{37}{0.3995}\ \text{kg} = 92.616\ \text{kg}$$

$$C = \frac{100(0.03 - 0.0005)}{0.40 - 0.0005}\ \text{kg} = \frac{2.95}{0.3995}\ \text{kg} = 7.384\ \text{kg}$$

2.1.2 Inline standardisation and blending

Even though manual standardisation is an easy way of mixing together skimmilk and cream it is not applicable when larger volumes or higher accuracy is wanted. Since the fat content is not constant in raw milk, it has to be measured continuously to get a more precise fat content in the final product. The actual mixing will then have to be done in the tubes rather than in the tank.

A similar approach can be used for protein standardisation; all you need is some way of measuring the protein content. One way to do this is to measure the SNF content which can be done with a density meter. The protein is about 40% of the SNF and 80% of all protein is casein (which often is the interesting protein). The rest of the SNF consists of lactose and minerals (also called ashes). A more accurate, and expensive, way to measure protein content is to use some sort of IR sensor, most commonly based on NIR technology.

When standardising the fat content to a lower fat content than in the raw milk there will be some surplus cream since the inflow of raw milk is continuous and not all is remixed. The surplus cream can then be processed and stored for more fat needy products like butter or whipping cream. This is the big advantage with inline standardisation which quickly pays back the investment for a dairy.

Inline standardisation is almost always done directly after a separator, see Figure 2.1. The cream fat content is then standardised by a cascade controller that

combines measurements of flow and density. Why both flow and density meters are used is because of the variations in fat content in the inflow raw milk. If a density meter had not been used, these variations would not had been detected which would had affected the fat content in the outgoing product. A standard setup for an inline standardisation process can be seen in Figure 2.1. The normal setting is to have the inflow to the separator controlled to a fixed value. The amount of cream and skimmilk out from the separator can then be calculated with equations (2.7) and (2.8). These are derived from the mass and flow balances of the separation point, equations (2.5) and (2.6). ϕ_W is the flow (in kg/h) and f_W is the fat content of the whole milk. ϕ_C, ϕ_S are the flows of cream and skimmilk. f_C and f_S are, as before, the fat contents of cream respectively skimmilk.

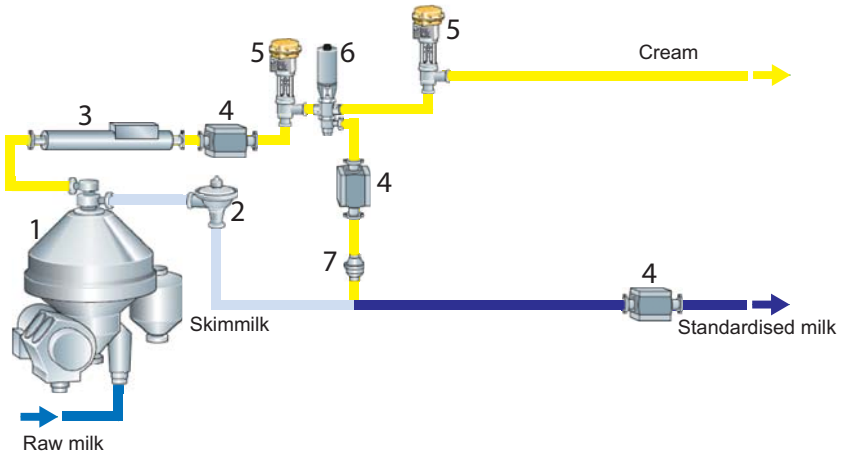

Figure 2.1: Flowscheme of the inline standardisation. 1 – separator, 2 – constant pressure valve, 3 – density meter, 4 – flow meter, 5 – control valve, 6 – shut-off valve, 7 – check valve.

$$\phi_W f_W = \phi_S f_S + \phi_C f_C \tag{2.5}$$

$$\phi_W = \phi_S + \phi_C \tag{2.6}$$

$$\phi_C = \phi_W \frac{f_W - f_S}{f_C - f_S} \tag{2.7}$$

$$\phi_S = \phi_W \frac{f_C - f_W}{f_C - f_S} \tag{2.8}$$

Control of fat content is actually a control of the remix flow, ϕ_R, of cream. To calculate the controller's set point another mass balance have to be used, Equation (2.9). From this equation the set point for ϕ_R is calculated as Equation (2.11) where ϕ_S is calculated with Equation (2.12) (ϕ_{RPV} is the currently measured process value of the remix flow). Since the set point is dependent of the process value there will be some time before it has converged, thus the PID-controller will not change the valve setting instantly. Another way to calculate the same set point can be seen in Equation (2.13). The only difference is that the flow balance (2.10) has been used in the mass balance (2.9) to derive the equation instead of using a calculated value for the skimmilk. The choice of equation to use is not important

since both will give the same answer. See Example 2.2 for a calculation with some standard values.

$$\phi_M f_M = \phi_S f_S + \phi_R f_C \tag{2.9}$$

$$\phi_M = \phi_S + \phi_R \tag{2.10}$$

$$\phi_R = \frac{\phi_M f_M - \phi_S f_S}{f_C} \tag{2.11}$$

$$\phi_S = \phi_M - \phi_{R_{PV}} \tag{2.12}$$

$$\phi_R = \phi_M \frac{f_M - f_S}{f_C - f_S} \tag{2.13}$$

Example 2.2: Fat standardisation by inline cream remix
$f_W = 0.04$, $f_C = 0.40$, $f_S = 0.0005$, $f_M = 0.03$ and $\phi_W = 1000$ kg/h. ϕ_C and ϕ_S are calculated with equations (2.7) and (2.8):

$$\phi_C = 1000\frac{0.04 - 0.0005}{0.40 - 0.0005}\ \text{kg/h} = 1000\frac{0.0395}{0.3995}\ \text{kg/h} = 98.87\ \text{kg/h}$$

$$\phi_S = 1000\frac{0.40 - 0.04}{0.04 - 0.0005}\ \text{kg/h} = 1000\frac{0.36}{0.0395}\ \text{kg/h} = 901.13\ \text{kg/h}$$

These calculations only show the function of the separator; the flows ϕ_C and ϕ_S are actually measured respective calculated and used in equation (2.11) to calculate the set point for the remix controller. If the inflow of raw milk is considered completely constant both in flow and composition (fat content) the flows of cream and skimmilk will also be constant. If the above given values are used, the first calculation of the set point with equation (2.11) will give:

$$\phi_M = \phi_S + 0$$

$$\phi_R = \frac{(\phi_S + 0) f_M - \phi_S f_S}{f_C}\ \text{kg/h} = \phi_S \frac{f_M - f_S}{f_C}\ \text{kg/h} =$$

$$= 901.13\frac{0.03 - 0.0005}{0.4}\ \text{kg/h} = 66.46\ \text{kg/h}$$

This value is the set point for the controller which now will increase the process value $\phi_{R_{PV}}$ from 0 to a higher value. If it in the next set point calculation is, for example, 30 kg/h the calculation will be:

$$\phi_R = \frac{(901.13 + 30) \cdot 0.03 - 901.13 \cdot 0.0005}{0.4}\ \text{kg/h} = 68.71\ \text{kg/h}$$

The process described above is fairly straightforward compared to the following. It has been mentioned earlier that protein standardisation sometimes is used. Then the process will try to standardise both fat and protein at the same time. A setup for such a process can be seen in Figure 2.2 where the earlier process has gotten an additional inflow, additive, that will take care of the protein standardisation. Since there seldom is a flow that only contains fat or protein, there will be some influences between the controllers. Instead of only one mass balance there will be two, one for fat and one for protein, as can be seen in Equations (2.14) and (2.15). Here p_M, p_S, p_C and p_A are the protein contents for each flow. A is the additive. The flow balance, Equation (2.16), will also include an additional inflow, ϕ_A.

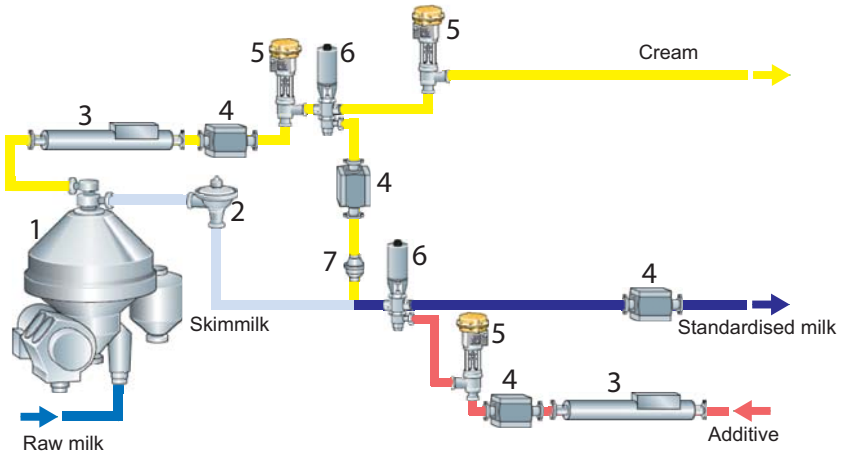

Figure 2.2: Flowscheme of the inline standardisation with one additive. 1 – separator, 2 – constant pressure valve, 3 – density meter, 4 – flow meter, 5 – control valve, 6 – shut-off valve, 7 – check valve.

$$\phi_M f_M = \phi_S f_S + \phi_R f_C + \phi_A f_A \tag{2.14}$$

$$\phi_M p_M = \phi_S p_S + \phi_R p_C + \phi_A p_A \tag{2.15}$$

$$\phi_M = \phi_S + \phi_R + \phi_A \tag{2.16}$$

These equations are then used to derive the set point equations for the cream remix flow, Equation (2.17), and the additive flow, Equation (2.18). (2.18) will be valid except when the protein standardisation is done by diluting the flow with water. Then the protein content in the additive, and with that the denominator in the equation, will be zero. This will give an undefined set point calculation and is of course not desired. Instead the equation can be derived using the flow balance, Equation (2.16). This will give Equation (2.19) for the protein set point (ϕ_S is as before calculated by knowing the other flows). A similar approach can be used if all cream is remixed (called full remix) and the fat content is standardised by adding appropriate amounts of water.

$$\phi_R = \frac{\phi_M f_M - \phi_S f_S - \phi_A f_A}{f_C} \tag{2.17}$$

$$\phi_A = \frac{\phi_M p_M - \phi_S p_S - \phi_R p_C}{p_A} \tag{2.18}$$

$$\phi_A = \frac{\phi_M (p_M - p_S) - \phi_R (p_C - p_S)}{p_A - p_S} \tag{2.19}$$

A large system with many inflows does not only contain fat standardisation and protein standardisation. There may as well be additional ingredients that are added for taste and/or consistence. These are normally added with proportional flow control, i.e. as a ratio of the outflow, or a fix flow. An equation for a proportional flow can be seen in Equation (2.20), where k is the ratio. Since these flows also contain fat and protein, their contents have to be taken into account by the fat/protein standardisation. An example of a large scale system with many additives can be seen in Figure 2.3.

There are also more complex control functions as fat/protein ratio control, where a ratio between the fat and protein are controlled to a certain value by adding an appropriate amount of either fat or protein. These functions are however not discussed further in this thesis.

$$\phi_B = k\phi_M \tag{2.20}$$

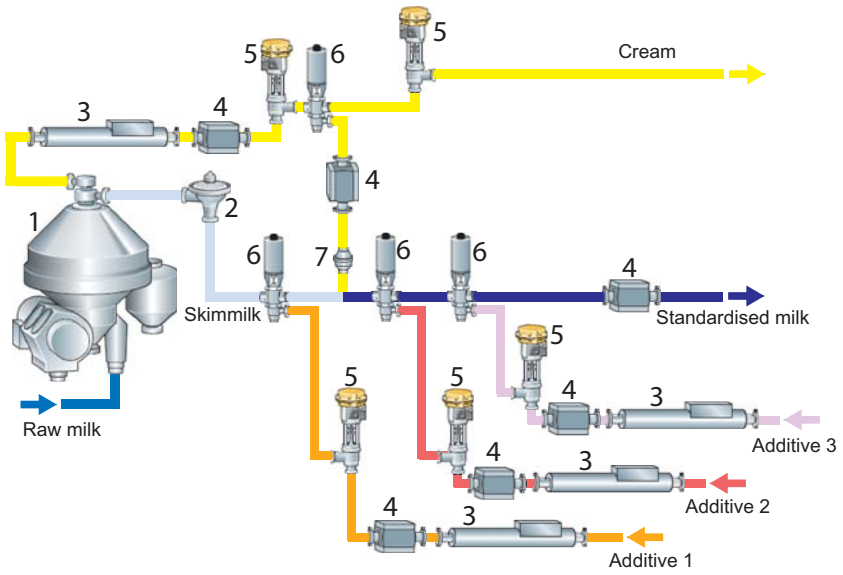

Figure 2.3: Flowscheme of a large scale system. 1 – separator, 2 – constant pressure valve, 3 – density meter, 4 – flow meter, 5 – control valve, 6 – shut-off valve, 7 – check valve.

A blending process does not always contain a separator. It could just as well take its products from a tank. Then all ingredients will function as additives to a base flow (normally skimmilk, just as when having a separator). This will however not affect the flow and mass balances. The set point equations for the standardisation flows are still derived by solving the mass balances.

2.2 Future requirements from the process

The most common setups for standardisation and blending do not need more than cream remix and maybe one additional inflow (additive). But more and more complex setups are wanted, thus more additives have to be handled. Since one additive have influence on all other, the standardisation of fat and protein content in the final product becomes more difficult with more calculations and more simultaneous control tasks. The difficulty is not only to manage the control task of a certain number of additives; it should also be easy to customize the program code to the customers demand. It should also be possible to construct additional functions, e.g. lactose standardisation, in a later stage of development. For some suggestions on solutions for these problems, the reader is referred to Chapter 5.

Chapter 3
PLC – SoftPLC comparison

A PLC (Programmable Logic Controller) is a control system for machines and processes. SoftPLC, also called PC-based automation, is the concept of a PLC-program run on a PC platform, normally an industrial PC with high demands on reliability. In the rest of this thesis it will always be mentioned as SoftPLC. During recent years the market has become more and more interested in SoftPLC. This interest depends on many factors including price, performance and increased reliability. This chapter is an attempt to explain some of the questions that may come to mind when deciding if an ordinary PLC or a SoftPLC should be used. Some basic facts on PLCs are discussed for example in [Olsson & Rosén, 2000].

3.1 Operation reliability

Operation reliability is the most critical issue for control systems. If not enough reliability can be accomplished, it has no significance that the performance is outstanding. To get a sense of the factors that are included in operation reliability and the difference between PLC and SoftPLC in this matter, it is discussed in the following sections.

3.1.1 Real-time operation

In any ordinary control system, many different execution processes (also called tasks or threads; threads are processes with less status information, so called light-weight processes) run at the same time. In control systems, most processes are periodic processes, i.e. they have specified timing intervals when they should be executed, e.g. every 10 ms or every 200 ms, which is called the scan time or cycle time. The difference between the ideal start time and the actual start time is called the latency of the process and the execution time of the process is called the service time. An illustration of the times can be seen in Figure 3.1. A real-time operating system (RTOS) is deterministic and guarantees a worst case latency time, contrary to ordinary operating systems where the average-case latency is most important.

There are also non-periodic processes, interrupts, which can be called from either internal or external events. An internal event, such as a clock interrupt, comes from within the system while external events are called by an input signal, e.g. a button push or an alarm signal.

A central role in real-time systems is priorities. The priority of the process tells how important that process are. A high priority process is normally the control

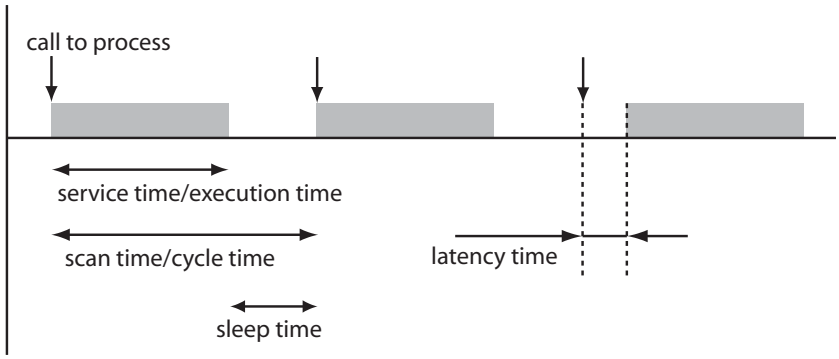

Figure 3.1: Illustration of times for processes/tasks. The third execution block is delayed and has a latency time.

calculation task. A low priority process is the updating of the graphical user interface (GUI). If a low priority process is executing and a call to a high priority process comes, the process with lower priority will be preempted and the system will start executing the one with higher priority. This change of executing process is called a context switch and involves storing the actual state of the process and its variables in the memory. When the interrupting process is finished, there will be another context switch where the states are restored and execution is continued from the same place as it was stopped.

Some parts of the processes are very important, e.g. the calculation of the control output signal. If the process is interrupted during this calculation there could be an error in the calculation if the parameters or input signals are changed. To prevent this, a critical part of the process can be protected against context switches. This mechanism is called mutual exclusion. Generally, this should be used when using common resources, using locks that prevent other processes from executing if they want to access the locked resource. However, this introduces another problem, deadlock. Deadlock can occur when two processes both want to access the same resources. An illustration of the deadlock phenomenon is shown in Figure 3.2. Process A locks resource 1 but is interrupted. Process B locks resource 2 and executes until it needs resource 1. Process A then resumes its execution until it needs resource 2. Now both processes is in a state of waiting of each others locked resources, thus a deadlock has occurred. To prevent deadlock the order of resource allocation should be the same in all processes, i.e. first resource 1, then resource 2.

Another part of a real-time operating system is priority inheritance. If the low-priority process A locks some common resource but is interrupted by the higher priority process B. B is then interrupted by process C that also wants the resource locked by A. The resource is however blocked by A and A is blocked by B. Then C can not be executed and B has to finish before A and C can execute. This is called priority inversion and is not a desired behaviour since a lower priority process (B) is in fact blocking a higher priority process (C). To prevent this, priority inheritance is used; A gets the priority of C when C tries to access the resource, making A to finish the execution with the resource before C can interrupt. See Figure 3.3 for an illustration of the execution without and with priority inheritance. [Årzén, 2003]

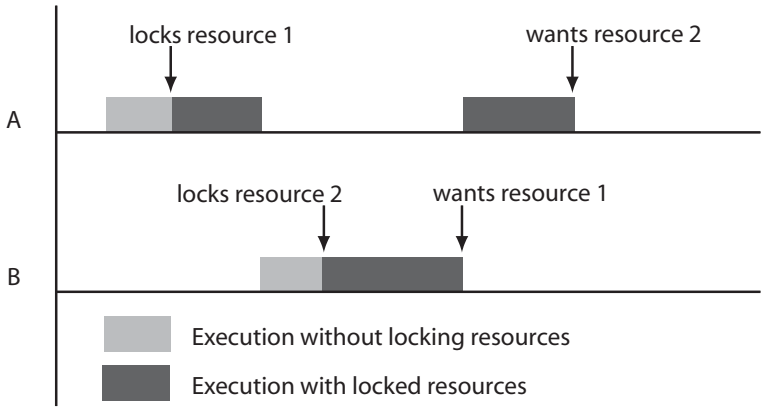

Figure 3.2: Illustration of the deadlock phenomenon. B has higher priority than A.

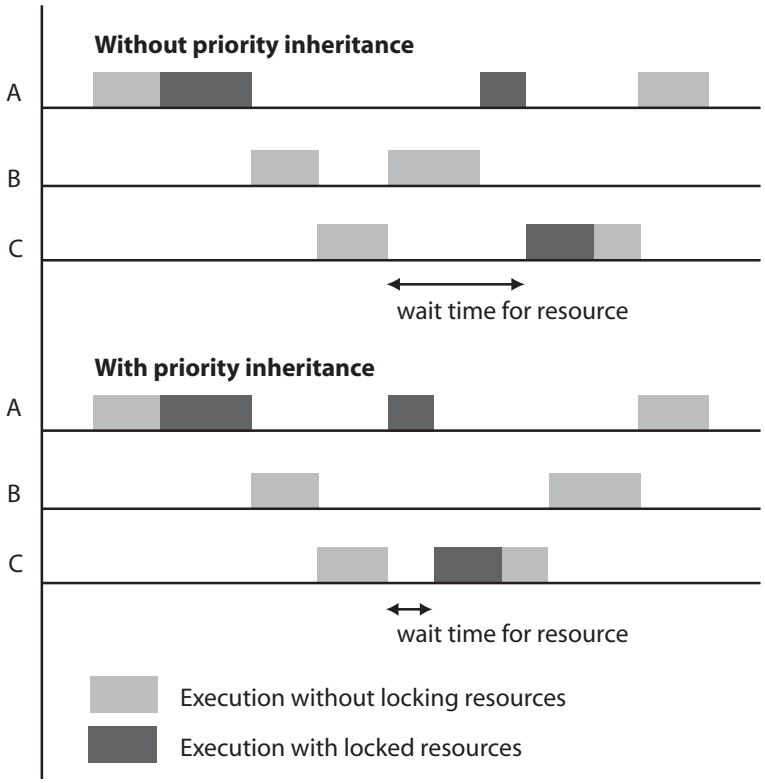

Figure 3.3: Illustration of the effects of priority inversion. Above is the execution without priority inheritance and below is the execution with priority inheritance. A has low priority, B has medium priority and C has high priority.

An ordinary PLC runs its processes in a real-time system. In a SoftPLC this is not always ensured. A SoftPLC-application run on an ordinary PC with Microsoft Windows operating system does not ensure real-time properties itself. Windows lacks some of the principal components to be called a real-time operating system; it has too few different priority settings, it has a non-deterministic scheduling and priority inheritance is not permitted. [Venturecom, Inc., 2003], [Fischer, 2005]

If real-time operation is wanted, the SoftPLC either has to be run on a real-time operating system (RTOS) or on a RTOS-kernel inside Windows. If a Windows/RTOS symbiosis is used it will allow other applications to be run along with the control application. This is an advantage if external programs such as HMI (Human Machine Interface), database system, firewall or web-server is wanted. It is also good if Windows standard components for communication can be used. In a special made RTOS this could perhaps be a problem. There are however manufacturers of control systems who choose to run their SoftPLC-application on solely Microsoft Windows. This demands higher performance by the computer to minimise latencies for processes. It is however not deterministic and is not recommended for high demanding processes, see Figure 3.4 for motivation.

An advantage for the Windows/RTOS combination (e.g. RTX from Ardence) is that if Windows crashes (blue screen) the RTOS will continue operation in a limited way. It will allow a shutdown routine to be run, shutting down the control system in an orderly fashion.

3.1.2 Moving parts

A major disadvantage of normal PC, and with that SoftPLC, compared to a PLC is that it has moving parts, i.e. hard drives and cooling fans, and therefore is not considered as robust as a PLC. Like all moving parts, these will eventually break, causing a production stop if used as a control system. Because of this, specially made PCs for industrial use have been developed by certain manufacturers, e.g. B&R, Beckhoff and Siemens. These will however not be evaluated individually in this thesis, only discussed generally with some information from the manufacturers.

The industrial PC has instead of hard drives some kind of flash memory, most commonly a CompactFlash card (also known as CF card). CF cards are usually used in digital cameras; these are however not enough for industrial use since they have a limited amount of writes to the memory. To increase the number of writes, industrial CF cards have to be used. These will however not allow an unlimited number of writes, only an increase. So if Windows or some other operating system is installed on the CF card it should not use a page file, as this will write data to the card continuously. To overcome this in Windows, the embedded version has to be used. If no page file is used it might require more hardware memory to operate. [James-Damato, 2006]

Cooling fans are in industrial PCs replaced by extensive use of heat sinks combined with low temperature parts, e.g. the processor is commonly made for laptop or embedded use and is not as heat radiating as desktop processors.

3.1.3 Power supply

A problem that resides in both PLCs and PCs, as well as in all other electronic products, is their lack of resistance to power failure. If the power will shut down, even if only for seconds, all operation will do so too. To prevent this, an uninterruptible power supply (UPS) should be used. An UPS is in general a battery that kicks in when it detects a power failure. This will allow a short runtime until the power gets back or, if the power is gone too long, an orderly shutdown of the control system.

If the control system has been shut down, it has to be restarted. If no UPS is used this will perhaps happen more frequently, depending on the local power grid. The restart time then becomes a critical property of the control system. If the flow through the pipes is still on when the system shuts down everything that is produced meanwhile the shutdown will be uncontrolled and will affect the quality of the product. A PLC has a very short restart time. A PC generally takes much longer time, depending on the amount of drivers and applications that should be loaded.

3.1.4 Redundancy

PLCs have, at least the larger and more expensive versions, different options of redundancy, e.g. double controllers, double I/O cables, double fieldbus connections. A redundant controller is in a stand-by mode and is activated when the first controller fails and takes over the control. This gives a more reliable system as the probability that both connections and controllers should fail is very low.

For SoftPLC similar redundancy can be used. Double computers are the equivalent to double controllers for ordinary PLCs. The stand-by feature must be supported by the software if it should have the same functionality as double controllers. Double processors or a dual-core processor is another option for SoftPLCs. This is however not so much a reliability issue as a performance issue. The main purpose for having a dual-core processor is to get higher performance with simultaneous execution of two processes at the same time.

Another redundancy that could be used, if not using CompactFlash cards as described above, is a RAID-system of hard drives where the same content is stored on two separate drives. Hard drives gives a higher storage capacity than CF cards but does not remove the issue of moving parts and is not considered as a good option in a control system that should be able to run continuously for several years. It is more interesting when intensive data logging is needed and could perhaps be used in conjunction with a CF-card.

3.2 Performance

Performance for control tasks are not so much an issue of speed as an issue of enough speed. A slower computer can be better suited than a faster computer; the vital part is instead how the performance is utilised. The real difference between PLCs and SoftPLCs in performance is that much more powerful CPUs are used for SoftPLCs. The important question, through this section, is if all this power can be utilised?

One option for SoftPLC is to have double processors or a dual-core processor. This allows two processes to be executed simultaneously. If many processes try

to execute at the same time, the dual-core will give an increase in performance and a decrease in latency times. An interesting idea would be to let one core execute the control process and the other core the other processes. Double processors increase performance but are not a necessity; it depends on the number of processes and how long execution times they have. If only one process is executed there will be no performance enhancement when going from one to two processors.

As discussed earlier, real-time operation is of great importance. A PLC is always real-time which creates a deterministic and reliable system. Processes could be event triggered (e.g. started by a push on a button) or periodic. Periodic processes are executed at regular intervals. As said earlier, the time the process takes to execute is called scan time. If a process is executed in a fast CPU, the scan time is shorter than in a slow CPU. This would allow higher frequency of executions.

An ordinary Windows system is not real-time, it is a multiprocessing kernel where the mean latency time is minimised while a real-time environment minimises the worst-case latency. The difference can be seen in Figure 3.4, where latencies for a Windows system with a RTOS-kernel is compared to an ordinary Windows system. To get some real-time properties an external real-time kernel has to be installed under Windows. This kernel will then run the PLC programs while other applications will run directly under Windows. The real-time kernel will have a dedicated share of the CPU time available for its needs, as an example Siemens SoftPLC can be adjusted to have maximum 90%. If it does not need all of the CPU time, Windows will be allowed to use it for other applications. The remaining 10% will however not come all at the same time. The CPU availability will instead be spread out over time, giving a jitter in operation for other applications. The same phenomenon can happen if some low-priority task is interrupted often by a high-priority task. To prevent this, a guaranteed sleep time has to be set for the interrupting task. It is however not realistic to run the real-time kernel at 90% and at the same time have a number of Windows applications executing.

Finally, an example of how the performance could be compared can be seen in Table 3.1. There the execution times of some PLCs are compared to a SoftPLC (Siemens WinAC RTX, since this is the only one with indicated execution times). If you compare the SoftPLC with the fastest of the PLC systems, there is a factor 10 between the times. The tested CPU, Intel Pentium 4 2.4 GHz, is however a more powerful CPU than most industrial PCs use. A qualified guess is that the SoftPLC in an industrial PC will be approximately five times faster than a PLC of the same class as Siemens S7-319. The reason that only Siemens systems are presented is that they are the only producer that has both types of systems (see more in next chapter). Other manufacturers do not present their execution times in the same way, making additional comparisons unfair. The important here is the performance difference between the PLC and the SoftPLC. The CPU frequency is unfortunately not known for the PLC systems.

	CPU	*Float operation* (μs)
PLC		
Siemens S7-319	n/a	0.04
Siemens S7-416	n/a	0.12
SoftPLC		
Siemens WinAC RTX	P4 2.4 GHz	0.004

Table 3.1: Execution times for PLC and SoftPLC systems. All are from Siemens to get a fair comparison.

3.3 Summary

As a small conclusion, some criterions that a SoftPLC should fulfill if it should be chosen are presented:

- A real-time kernel, either a RTOS or a real-time kernel inside Windows.
- No moving parts, i.e. no hard drives or fans.
- The embedded version of Windows has to be used to prolong the lifetime of CompactFlash-cards.
- An UPS has to be used, to prevent long restart times for short power failures.

A SoftPLC has both its advantages and disadvantages versus a PLC. The PLC is more industry tested and even if an industrial PC with all the above criterions are fulfilled it is not considered as reliable as a PLC. However, the SoftPLC gives more performance and flexibility. It is also better when heavy data logging is needed.

The important when looking at the choice is to not concentrate solely on the performance and price. The performance is not as much better as one can first think. If a PLC has a CPU of 50-100 MHz and a SoftPLC has 2.4 GHz, one can easily think that the SoftPLC is 24-48 times faster. This is however *not* the case, as can be seen in Table 3.1. The difference between the PLC and SoftPLC is instead about a factor 5-10 in execution time. Of course, the performance will increase with faster processors, but a industrial PC can not have a processor with too much power consumption since this will need fans or other cooling devices to not overheat.

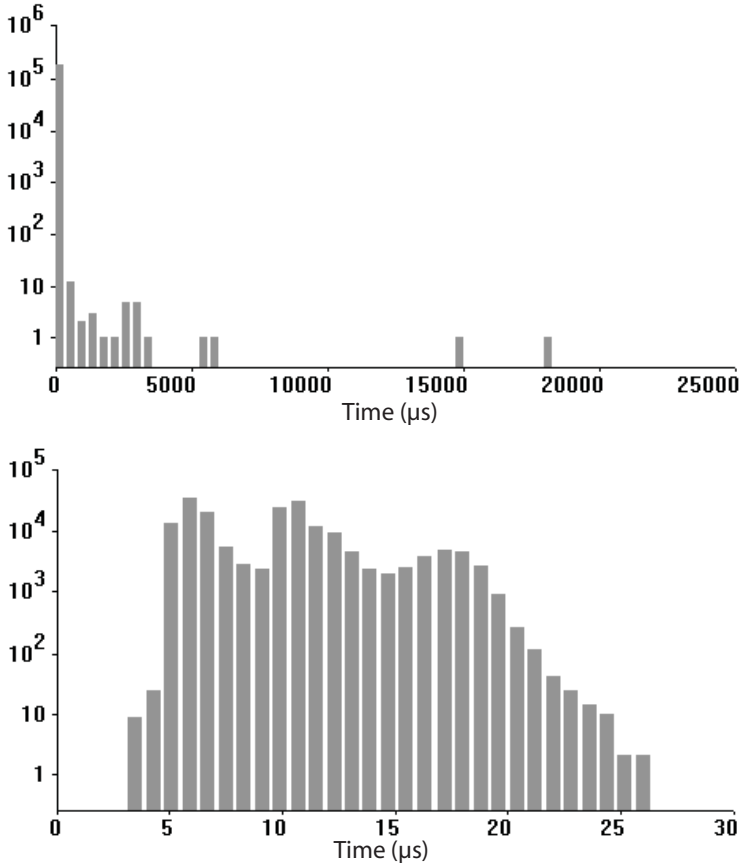

Figure 3.4: Timer-interrupt latencies for a Windows system (above) and a Windows system with a real-time kernel (below). In both cases the same workload of disk searches, network activity and more is present. [Venturecom, Inc., 2003]

Chapter 4
Investigation of control systems

One of the main tasks of the thesis is to investigate different control systems and find out what system that is most suitable for the needs. To be able to do this, some criterions are set that a system should be able to fulfill, at least in some way, to be a candidate. The number of candidates is then reduced by some comparison and discussion. Left are the systems that could be possible to use.

4.1 Choosing criterions

Performance
- Be able to have lower scan times than today.
- Enough program memory for both the system as it is today and future additions.
- Ability to calculate (fast) with floating point numbers.

Reliability
- No moving parts.
- Hardware should be tested for industrial use.
- Software should be tested for long term use.
- Different levels of redundancy, e.g. multiple CPU, sensors and cables.

Communication
- Communication with sensors and actuators. For I/O-specification see Section 4.3.
- Possibility to communicate with other brands of I/O.
- Have some of the most common communication standards: PROFIBUS DP/PA, DeviceNet, ControlNet, PROFINET, EtherNet/IP.

Development
- Development environment with a good overview and structure.
- Advanced features for automatic control (PID with gain-scheduling and auto-tuning, fuzzy logic, freezing).
- Module-based programming. (Modules are self-made complex function blocks that can contain other function blocks and code.)
- Code safety. Ability to protect and hide some parts of the code. Hide different routines for different users.
- Simulation.
- Ability to reuse the code from previous system. (only applicable for ABB).

Service
- Documentation and online help for operator.
- Fault diagnosis.
- Remote access (i.e. Internet, mobile).
- Software upgrades by remote access or local installation (complete upgrades and single module/library upgrades).

Operation and operator interface (HMI)
- User levels, i.e. different levels of access for different users.
- Logging of data (database).
- Trend diagrams.

Market
- Market share and goodwill. Not a too small company or a company with bad references.
- Future of the system. The control system should be a product that the manufacturer will continue to develop and maintain for many years.
- Experience and use within Tetra Pak.
- What do Tetra Pak competitors use?
- Price.

Existing installed systems
- Use of the system world-wide.
- How many installed systems?
- What environments/processes are the systems used in? (Machine control or process control?)

4.2 Priority analysis

Since most control systems fulfills the criterions mentioned above, some prioritising is done. Below are the most important features listed in a falling scale of priority:

1. An absolute requirement is of course the performance and memory requirements. Since this is closely connected to the development environment and the amount of code there is no way to exactly decide the requirements in absolute numbers.

2. Because of the demands of flexibility in the code there has to be some kind of scalability in the code. This is done with modules, self-made function blocks, and therefore this is also an absolute requirement.

3. Availability of I/O-modules for sensors and actuators. Alternative communication with fieldbuses (mainly PROFIBUS) could be used for some I/O.

4. Since the system is time critical, a real-time control system should be used. Ordinary PLCs are always real-time but for SoftPLCs this is not always the case. Because of that a SoftPLC should have a real-time kernel. Also for SoftPLC, the hardware have to be developed for industrial use. No investigation of different industrial PCs is done but a general discussion can

be read in Chapter 3, giving some background information when looking at PCs.

5. Price.
6. Experience and use within Tetra Pak.
7. Manufacturer market share and use of their control systems world wide.

4.3 I/O specification

I/Os needed:

- Digital inputs and outputs.
- Temperature inputs (analog inputs). Used for input from PT100. Preferably with 4-wire technology.
- Analog outputs. Used for sending signals to control valves and external recorder.
- Pulse counter module. Should be able to measure the time between pulses (period time) continuously. Used for both density meters and flow meters. The density meter gives a signal of about 700 Hz (1400 μs) and an accuracy of about 1 ns is wanted. Today, this is accomplished by calculating a mean period time over several periods measured with a 10 MHz internal clock. The flow meter does not need the same high accuracy but instead both the period time and the number of pulses are necessary. This might require two counters for a single flow meter.

The I/Os described above are the required module types. Normally, when attaching I/Os these are connected at the same backplane. An alternative way is to instead use distributed I/Os (also called point I/Os). These are connected to the controller with a fieldbus connection. Distributed I/Os often have smaller modules and is cheaper but requires a fieldbus module for the controller (a master) and a fieldbus slave that the I/Os are connected to.

As seen below in Section 4.4, not all systems will have good enough I/Os available. Since more and more systems require fieldbus communication, this could be used for connecting sensors as well. If all flow meters, density meters and PT100 is connected via a fieldbus (often PROFIBUS DP) all I/Os that are needed are digital inputs and outputs and analog outputs. The important thing is that the advanced pulse counter module is not needed anymore. However, fieldbus connections for all sensors do not come free. The sensors have to have a fieldbus device and a corresponding fieldbus master is required for the controller.

4.4 Investigated systems

With the criterions mentioned above in mind, an investigation of available systems is done. It is concentrated on large suppliers, but some smaller alternatives are also considered. Common for all companies is that Tetra Pak already has some experience and/or business relations with them.

The investigation consists of first a performance/memory investigation of the different CPUs. Then different options for I/Os are investigated, i.e. ordinary I/Os, distributed I/Os (point I/Os) and fieldbus alternatives. These are compiled

into standard packages where the prices also are compared. This information is however confidential and will not be mentioned here. Specifications for the I/Os can be seen above in section 4.3. The last part is a roughly investigation of the development environment and other tools to the system.

4.4.1 Rockwell Automation

[Rockwell Automation, 2006]

PLC – ControlLogix

There are three interesting types of Rockwell's PLC ControlLogix: 1756-L61, 1756-L62 and 1756-L63. The difference is mainly memory capacity where L61 has 2048 kB user memory while L62 has 4096 kB and L63 has 8192 kB (this is where the program routines are stored). All have the option of a removable CompactFlash card to store information. There are no known performance values for ControlLogix.

The ordinary I/Os does not provide a 4-wire module for temperature measuring but there is a 3-wire module called 1756-IR6I. For pulse counting there are some different options. The one chosen seems to be specially made for flow meters and is called 1756-CFM. It has a 16 or 20 MHz internal clock and 2 channels which allows connection of one flow meter or two density meters per module.

The distributed I/O also only has a 3-wire temperature module (1734-IR2). There also is a pulse counter (1734-IK) with one channel but the specification does not give any information about the resolution. To connect the distributed I/Os a ControlNet module is also needed.

To directly connect to the sensors a fieldbus module (PROFIBUS DP, since this is what the sensors have) is needed.

The development environment does not permit creation of own functions blocks (AOI). This will however be implemented in a soon to come version. Program languages supported are LD, FBD, ST and SFC. There are ready-made functions blocks with PID controllers and such.

SoftPLC – SoftLogix

Rockwell also has a SoftPLC solution, SoftLogix. This uses the same development environment as ControlLogix with addition of using libraries made in C/C++. SoftLogix is run under Windows XP with real-time priority, without a real-time kernel. See more in chapter 3 for further explanation.This is however not a hard real-time environment. Since one of the criterions for choosing a SoftPLC-solution is that it should be real-time, SoftLogix is disqualified from further analysis.

4.4.2 Siemens

[Siemens, 2006]

PLC – S7

There are two series that could be interesting: the 300 and the 400 series. In the 300-series there is a PLC, S7-319-3, that has high performance (float operation in 0.04 μs) but it has not very much memory (1.4 MB total user memory that have to be used for both program and data). To get more memory capacity the 400-series have to be used, where the interesting PLCs are the S7-416-3 and S7-417-4. These are Siemens' largest PLCs with 5.6 respective 20 MB user memory where half is used for program code (i.e. 2.8 and 10 MB program memory). These are however slower than the 319 (float operation in 0.12 μs for 416 and 0.09 μs for 417). Both the 300-series and the 400-series can have an optional flash card for storing other project information.

The ordinary I/Os (the 300 and 400-series have different I/Os) have a temperature module with 4-wire technology. The pulse counter module (FM350-1 for 300 and FM450-1 for 400) counts period times by using a 16 MHz internal clock and taking an average over all pulses that comes within a specified time window. The time window is set by the user and can be changed during execution. FM350-1 has 1 channel and FM450-1 has 2 channels.

The distributed I/Os (ET200S) have all the same functions as the ordinary I/Os but with smaller and more flexible modules.

All Siemens PLCs have integrated PROFIBUS DP interface. This can be used for connecting directly to sensors and for connecting to distributed I/Os.

The development environment for Siemens is called Step 7. This provides creation of own function blocks and supports the following program languages (in the complete version): LD, FBD, ST, IL, SFC and S7-SCL (Pascal), CFC, S7-Graph, S7-HiGraph (state diagrams).

SoftPLC – WinAC

WinAC RTX is Siemens real-time enabled SoftPLC. RTX is a real-time kernel by Ardence that is used inside Windows. There are some built-in limitations in the system but after referring with experts on Siemens system, these are not considered relevant. A big advantage with the WinAC is that it uses the same development environment as the S7-systems (Step 7). This gives the possibility to use a S7-system as an alternative to the SoftPLC. Since there also is knowledge in-house of Siemens systems this is also an advantage. For WinAC, the most common I/Os are the distributed I/Os (ET200S) and PROFIBUS DP to sensors when possible. If an industrial PC from Siemens is used with WinAC there is a built-in PROFIBUS DP interface, else a PCI-card have to be used.

4.4.3 ABB

[ABB, 2006a]

PLC – AC 800M

There are three interesting PLCs in the AC 800M-series: PM861A, PM864A and PM865. These differs both in performance and memory capacity. Today an ABB SattLine 200 CPU40/80 system is used. CPU40 has a processor with 28.8 MHz. PM861A has 48 MHz while PM864A and PM865 has 96 MHz. PM861A has 8 MB program memory while PM864A and PM865 have 24 MB. To get a feeling about the performance difference between the different processors the execution times for a PID-function (PIDAdvancedCC) can be compared. CPU40 (is equivalent with PM254) takes 2982 us, PM861A takes 1153 us and PM864A takes 577 us (approximately half of PM861A). [ABB, 2006b]

There are two I/O-systems available for AC 800M: S200 and S800. S200 is what is used today while S800 is a newer I/O-system. S200 can only be used with AC 800M over a PROFIBUS DP connection. S200 has all the features that is needed, including 4-wire temperature module (200-IR8R) and a pulse counter module (200-IP4) with 4 channels, that either can be set to do high accuracy period time measurements (taking the mean over several pulses) or simultaneous period time/number pulses measurements. This means that either 4 density meters or 4 flow meters can be connected to a single module. The S800 however have no 4-wire temperature module, only one with 3-wire. Furthermore, the pulse counter in S800 (DP820) only has 2 channels and only provides one of the measurements.

ABB have no special distributed I/O but all I/O-modules can be connected by using a fieldbus. The sensors can be connected with PROFIBUS DP by using an interface module (CI801).

The development environment for AC 800M is called Control Builder. It provides the IEC languages IL, ST, LD, FBD and SFC together with creating own function blocks. It also has a control module language which can be seen as optimised function blocks where the code is sorted at compilation according to the data flow. This is an influence from the older ABB Sattline system. More about this can be read in Section 5.4.1.

SoftPLC – SoftController

ABB also have a SoftPLC, SoftController. This is however not a product they are promoting and is mainly used for simulation. Besides, it does not have a real-time kernel. Because of that it is disqualified from further analysis.

4.4.4 B&R

[B&R Automation, 2006]

SoftPLC – Automation Runtime

B&R has a SoftPLC called Automation Runtime that is run with a real-time kernel under Windows. As all SoftPLCs the performance and memory capacity is dependent on the computer that is used.

To get a temperature model with 4-wire technology the X67 I/O-system has to be used which is an IP67 class system. To get a pulse counter module (CM211) the 2003 I/O-system has to be used. This means that two or more different I/O systems have to be used at the same time.

Automation Studio is the development environment for the SoftPLC. It provides the following languages: IL, ST, LD, SFC, C-code and AB (B&R's own Automation Basic). There is the possibility to create own function blocks but these have to be created using IL, ST, AB or C. One type of function block can then be accessed in another function block.

4.4.5 Beckhoff

[Beckhoff, 2006]

SoftPLC – TwinCAT

Beckhoff has a similar product as B&R, i.e. a SoftPLC with a real-time kernel. Beckhoff's I/O-system is however not very convincing; there is no 4-wire temperature module or any appropriate pulse counter module.

The software provides the following languages: IL, ST, LD, FBD, SFC and CFC (similar to FBD). It also provides function block creation.

4.5 Choice of system

Since there is a time limit until the thesis has to be finished, not all systems can be analysed in full detail. During the investigation it has also been obvious that some systems are less suitable than others to control the blending process.

Beginning with the SoftPLC systems, only the ones with a real-time kernel are qualified for further analysis (see Chapter 3 for explanation). Since both Rockwell's SoftLogix and ABB's SoftController lacks this feature, they are not discussed further. The remaining three systems (Siemens WinAC, B&R Automation Runtime and Beckhoff TwinCAT) are similar in many ways. However, Siemens is considered superior in several areas, i.e. size of the company, experience within Tetra Pak, available I/Os and besides has the possibility to use the same code on a PLC system as on the SoftPLC. Without saying if a SoftPLC system should be used or not this means that if a SoftPLC system is wanted it should be Siemens WinAC.

When comparing the PLC systems, an important feature is of course the performance and memory capacity. Since the manufacturers does not provide comparable performance values in form of execution times there is no simple way to

do this. Depending on the program code, there may also be differences both in execution times and memory demands. For instance, a function block with a PID controller may take longer time to execute in one system than in another. There may also be different memory demands depending on what function blocks are used and how the program is structured.

Since a fair comparison between the systems can not be made, a qualified guess about the systems performance is done. ABB's PLCs (AC 800M) can be compared to the SattLine system, that is used today, with the execution times in [ABB, 2006b]. This gives an estimate that the PM864A (AC800M) is about five times faster than the CPU40 (SattLine). This is considered as fast enough. The PM864A has 24 MB, three times as much as the CPU40 that is used today. The other systems, i.e. Siemens S7 and Rockwell ControlLogix, are also considered as fast enough. For Siemens, the S7-319-3 is the fastest PLC but it does not have very much memory (only 1.4 MB). Instead the S7-416-3 is considered a better choice with its 2.8 MB memory even though it has execution times that are three times longer for float operations. The S7-400-series is however not usually used for control of a single machine. It is more often used to control a full-scale industry which also can be seen by the physical size of the controller, it is rather large and may not even fit inside an ordinary machine panel. For ControlLogix, no performance values are available and because of that the performance difference between the different PLCs are not known. The memory capacity is however known and by this knowledge the 1756-L62 seems as an appropriate choice with its 4 MB.

Since the system is dependent on high accuracy sensors, corresponding I/O-modules are needed to make use of the sensors accuracy. This is especially important for the pulse counter module. To overcome the demands on the I/O-modules, fieldbus connections to the sensors could be used. This will make the price for the sensor higher, since the signal processing has to be moved from the I/O-module to the sensor. The accuracy demand is also moved from the I/O-module to the sensor electronics. In return no expensive pulse counter module is needed. It also allows systems that do not have an adequate I/O-module to still be under consideration. Even though fieldbus connections could be used with new systems, there is one advantage with the ABB system: it will allow reuse of the old I/O-modules and sensors. The only thing that then has to be replaced is the PLC. ABB's S200 is also one of few I/O-systems that completely fulfills the specification. The only other system that does this is Siemens. There is however some questions if ABB's S200 will stay in the market as long as needed. Siemens controllers also have the advantage with having a built-in PROFIBUS DP interface.

Even though many aspects when choosing a control system are hardware-related, a very important matter is the development environment and the tools that comes with it. This is also where the experience within Tetra Pak comes in. Siemens, Rockwell and ABB are all used today at Tetra Pak. Many systems do however not require as heavy calculations as the blending process. This is the reason why a real-time kernel has to be used for a SoftPLC system.

The nature of the blending process requires that it is easy to make changes and setting up a system according to the customers needs, i.e. specifying the number of additives and functions. In Chapter 5 a program structure will be discussed that tries to solve this. The program structure is however dependent on the abil-

ity to create own function blocks. Since this feature is not available in Rockwell's development environment RSLogix at the moment, Rockwell is not chosen for the further analysis and implementation. With this it is not meant that Rockwell should not be chosen at a later stage of development. Both Siemens and ABB have the possibility to create own function blocks using any of the IEC 61131-3 languages.

Another aspect is the price of the control system. When comparing prices a standard system setup (without any additives) containing three flow meters and one density meter is used. This requires different amounts of I/O-modules depending on the amount of inputs and channels on the different modules. If using distributed I/Os and/or fieldbus connections, there will be a decrease in price. When connecting sensors through fieldbus there is however a higher cost for the sensor. The cheapest alternative is to use a SoftPLC with distributed I/O and fieldbus but this will require an industrial PC and perhaps some PCI-cards for communication. More detailed price information has been produced for internal use within Tetra Pak and will not be published here.

As a summary it can be said that ABB, Siemens and Rockwell all are possible to use for the blending process. All have their strengths and weaknesses, e.g. I/O-possibilites, price, SoftPLC-option etc. The two main alternatives are ABB and Siemens.; ABB mainly because the development environment and that it is the system used today. Siemens mainly because of the superior SoftPLC-option and that it is used in other Tetra Pak machines. Rockwell is considered to have a SoftPLC that is equal to ABB's with regard to real-time performance. Rockwell does not have the possibility to create own function blocks at the moment. This will be possible in a near future but disqualifies Rockwell from being a candidate for our implementation.

The best would be to develop a prototype-program in all systems and choose the best out of that. Since this is not possible for a time-limited study like this thesis, a single system is chosen together with Tetra Pak. To decide between using ABB or Siemens a large matter is the use of an ABB system today. This makes the use of some of the current solutions possible also in the new system. Since the authors do not have very much PLC-programming experience, ABB's development environment also feels easier to use than Siemens'. All together this means that ABB is chosen for the implementation. It is important to state that the choice of control system in this thesis is not the same as Tetra Pak's choice of control system. This is a decision that is beyond the scope of the thesis. This investigation instead tries to give a technical basis for the choice.

Chapter 5

Program structure and implementation

To get a flexible system with easy configuring of additional flows, additives, and functions a flexible program structure is needed. The purpose for this chapter is to present an idea for a basic structure, containing the most important parts of the system; i.e. the recipe management and the control calculation.

Today, there is a limitation of three additives. If one more should be added, extensive recoding is necessary in all the previously mentioned sections of the program. The goal with the program structure in this chapter is to overcome this problem and allow easy setup of an unlimited number of additives.

5.1 Decentralised calculation principle

Since the main goal of the program structure is to allow easy addition of additives the calculation of set points for each controller can not be made centrally in the program. Instead it is moved inside the additive. The problem with this is that the addititve then has to have information from all other additives to be able to calculate the set point. To overcome this problem, the additive does not calculate with fat contents in the same way as mentioned earlier in Chapter 2; instead it will calculate with fat mass flows.

An illustration of the calculation for a fat standardisation can be seen in Figure 5.1, which shows the process line with skimmilk, cream remix, two additives, a flow meter and a target. The central unit here is the target which knows the total flow and the current fat content set point (as it is set in the recipe). It then calculates the wanted fat mass flow by multiplying the total flow and the wanted fat content. This is the set point value for the fat mass flow (called SP) and is the signal sent to the left in the figure. From the left a process value for the fat mass flow (called PV) is calculated in the skimmilk unit (by multiplying the skimmilk flow and fat content) and is sent to the right. For each controller (cream remix and additives) that is not set to do fat standardisation, in the figure the additive, the SP and PV values are recalculated by subtracting respective adding the fat mass flow that is supplied by the current controller, see Equations (5.1) and (5.2). The SP will then decrease on its way towards the left while the PV will in increase towards the right.

$$SP_{left} = SP - Flow * Fat\ content \tag{5.1}$$

$$PV_{right} = PV - Flow * Fat\ content \tag{5.2}$$

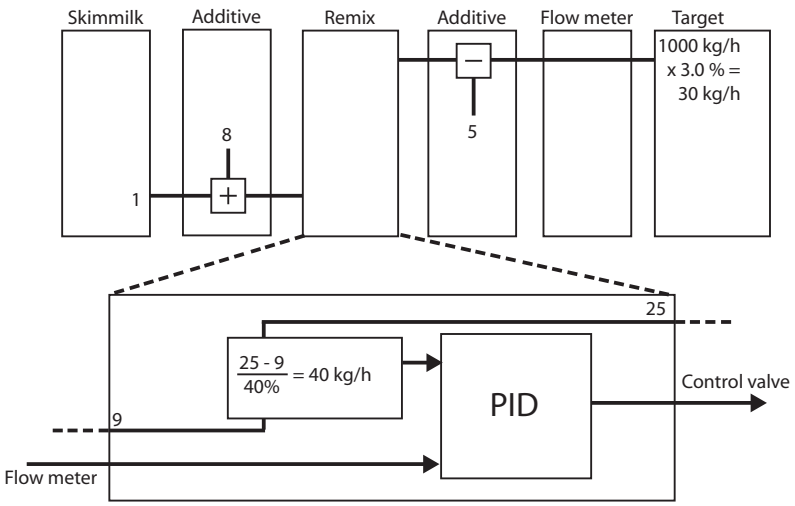

Figure 5.1: Illustration of the decentralised calculation principle. The top line is the SP value and the bottom line is the PV value. In this example the Remix is set to do fat standardisation and uses the SP and PV values to calculate a set point for the PID. The SP and PV values are sent as fat mass flows.

When it reaches a controller that is set to perform a fat standardisation, the cream remix in the figure, it will calculate a set point for its own flow controller using the SP and PV values and the fat content of the added flow. This is illustrated in Equation (5.3). A pleasant feature with this structure is that one controller never has to know how many other controllers there are. This simplifies the set point calculation to its simplest form; notice the resemblance between (5.3) and (2.11). In fact, the only thing the structure does is to divide the numerator in (2.17) so that every controller calculates a part of it. It is done in the same way for each standardisation type, i.e. fat, protein, SNF etc. All that is needed are the SP and PV values and some calculation code.

$$Flow = \frac{SP - PV}{Fat\ content} \tag{5.3}$$

5.1.1 Flow calculations

Every controller has its own flow meter but there is only one flow meter in the mixing line. This is normally placed after the last additive, directly measuring the total milk flow out. It could however also be placed before some of the additives requiring some calculation to get the total flow. The flow value measured by the flow meter is simply sent to the right and every controller will take the value from its left and add its own flow before forwarding the new value. When the value reaches the target box it will be the total flow.

The same principle is also used to calculate the skimmilk flow, the flow meters value is sent to the left and every controller will take the value from the right and subtract its own flow before forwarding the value. When the value reaches the skimmilk box it will be the skimmilk flow.

Both these values are used for the calculations described above and have to be calculated before the actual calculation for every scan. An illustration of the flow calculations can be seen in Figure 5.2.

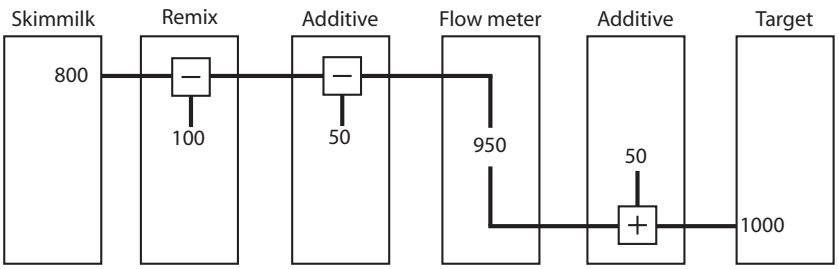

Figure 5.2: Illustration of the flow calculation. The upper left value is the skimmilk flow and the lower right is the total outflow. All values are in kg/h.

5.2 Recipe management

Since there should not be an upper limit for the amount of additives, the actual recipe storage is distributed to the controllers itself, i.e. the controllers have a number of recipes that store recipe number, function number and function set point. They will also have a number of ingredients, stored in the same way. In the recipe management the recipes are paired together with a certain set of ingredients, by storing the recipe number and the ingredient number together. If the ingredient number is changed for the running recipe, this will be forwarded to the controllers that will change their current settings.

5.3 Overview of the program structure

To get an overview of the program structure, the structure is outlined in Figure 5.3. The main parts are the control calculation and the recipe management. Behind these lies the process simulation. These three parts are executed in individual tasks, with different scan times. This is because the control calculation is more vital for the process than the recipe management that is not altered very often. The simulation will have the shortest scan time since it is supposed to simulate a continuous system. Between the control calculation and the recipe management there is some communication. The control modules are described briefly here:

- Skimmilk – gives a start value for the PV and takes care of termination of some signals.
- Remix – take care of everything with cream remix, e.g. set point calculation and control signal calculation. Also stores all recipe and ingredient settings for this controller. Needs connection to a flow meter and a valve.
- Additive – very similar to the Remix module, but gives different possibilities of functions. All additives are independent of each other and to add another additive a simple copy/paste is required together with connection to flow meters and valves.
- Flowmeter – used to calculate skimmilk flow and total flow. Independent of where it is placed along the flow line, the calculation will still be correct without any code changes.
- Target – this module keep track on what standardisation values that are wanted (e.g. fat content) and use this to calculate a wanted mass flow for each component (e.g. fat). This is forwarded to the controllers as a SP value. The Target module also takes care of the communication with the recipe management.

- RecipeManagement – keep track of what recipe that is running at the moment. Also stores recipe numbers and ingredient numbers, to know what ingredient group that should be used with a certain recipe. There are also some sub-modules that contain logic to take care of recipe changes and also some graphical elements that is used in this test-implementation.
- ProcessSimulation – simulation of the separation and mixing process.

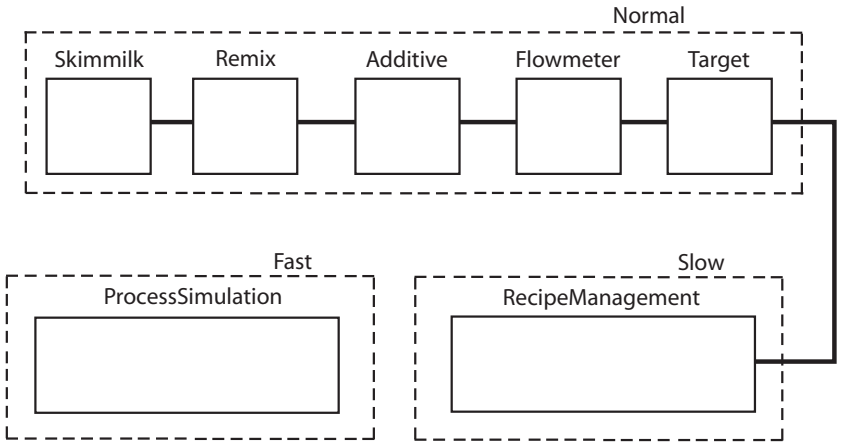

Figure 5.3: Overview of the program structure. The lines between the boxes represents information flow in both directions. The dashed line boxes are tasks with different scan times.

5.4 ABB information

Since ABB was chosen for the implementation, there have been some use of ABB specific features. Here the most important are mentioned, i.e. control modules and how to connect to an user interface.

5.4.1 Control modules – optimised function blocks

As has been mentioned earlier in the thesis, the possibility to use modules (self-made function blocks) is vital to be able to develop a scalable system. The functionality of a function block can then be expanded by placing it inside another function block with additional code around it. For more conventional object oriented programmers this can be compared to inheritance of classes.

As said in the previous chapter, ABB has been chosen for the final implementation. In ABB's Control Builder another option is available when a scalable system is wanted, i.e. control modules. Control modules can have the same code as function blocks but the code is sorted after data flow rather than program flow, which is the normal case. The sorting will then try to make an optimised code sorting according to the parameters and variables that are read and written in each code block. To get a correct sorting the code has to be divided into a number of code blocks, each taking care of the signals in one direction. An example of this is the calculation principle above where the SP values are forwarded to the left and the PV values are forwarded to the right. The code for updating and forwarding the variables will then be divided into two code blocks, e.g. named *left* and *right*. If

two control modules are connected to different tasks, these are not sorted together. Every task has its own sorting.

Control modules can also be graphical and have graphical connections to other control modules. They can also display graphical elements as buttons, texts and diagrams. If these elements are used, the system is very similar to ABB's older Sattline system. When the structure was implemented, these graphical elements were used to get a graphical overview of the system and to be able to show some results of the process simulation. This is however normally not used as a user interface, instead it is connected to the user interface using special variables which is described further in Section 5.4.2.

5.4.2 Connection to user interface

An AC800M can be connected to a user interface in some different ways, depending on what is wanted. There is the 800xA way, which is a fully integrated system that could be used as a supervisory system. It could also be connected with an OPC-server and in that way be used together with any OPC-compatible client. However, the way that has mainly been studied is how to connect the control system to a panel.

To allow a user interface to gain access to variables they have to be added to the Access Variables list. Any variable, except arrays and queues, can be used as Access Variables. These can then be used in the user interface that is built with Panel Builder. As mentioned earlier, the implemented user interface is not connected this way. This was considered to be outside the scope of the thesis since it is not a part of the actual program structure.

Chapter 6
More complex controller structures

Today, all automatic control in the system is done with PID controllers (commonly used as PI controllers). This means that every additive has its own PID controller. The set points for the controllers are calculated centrally every scan. Another way to do this is to distribute the calculation, as can be seen in Chapter 5. But, if one controller standardises fat content and another one standardises protein content, they will still unavoidably affect each other. This is perhaps not a very big issue if there only are 2-3 controllers but if there are 10 controllers this will become somewhat of a stability problem. There is however some different setups of the system where different control strategies are more or less interesting. This chapter will divide these setups into some groups and give an approach on how to choose suitable control strategies.

6.1 Control of the total flow rate

A central issue in the control is how the total flow rate is controlled. There are some different setups where the total flow rate is controlled at different points in the system:

- **Pasteuriser controls flow rate before the mixing point**
  Here is the flow rate into the separator controlled to a constant value. Since the fat content of the raw milk is not constant, this will give a varying skimmilk flow and thus a varying outflow.
- **Pasteuriser controls flow rate after the mixing point**
  Here the outflow is controlled to guarantee a fix flow through the pasteuriser. If the additives change their flow rates, the pressure in the mixing point will change which is used to control the inflow into the separation. This will in return change the skimmilk and cream flow.
- **Blending system controls the flow rate**
  This final option is normally only used when no separator is used and the flow rates instead come from tanks. Then all inflows can be controlled totally by the blending machine.

6.2 Control strategies

The process has three main properties that a control system can be optimised after. The first one is the composition of the ingredients, the second one is the controller robustness and the third one is the total flow rate.

If the mixing composition is of the utmost importance the additives should always react as fast as possible to adjust the mixing. This is perhaps more important when smaller volumes are used as the target since there is less space for compensation. To optimise the controller robustness all additives should not react on all changes. The last property is the total flow. In cheese production a constant flow out of the system is often used. This is however often controlled by an external controller (as described above), at least if a separator is used.

The strategies described below are focused on two of these three properties, i.e. the mixing composition and the controller robustness. The robustness strategy is perhaps more interesting if systems with many additives is used and will only function if no external flow controller of the outflow is used.

6.2.1 Focus on mixing composition

This is the strategy that has been described earlier in the thesis where it has always been of the outmost importance that the mixing proportions become right all the time. This means that all controllers (additives) react on all changes without any communication between each other. If one additive suddenly has a drop in the flow rate it will make all controllers to take action to prevent this. This is perhaps not a big problem with a few controllers (it could sometimes give better accuracy in the outgoing product) but in large systems it will make the system less robust.

6.2.2 Focus on robustness

An alternative is to use local compensation in every additive, i.e. if one controller has a decrease or increase of a flow rate it will take care of this itself. For example, if an additive for some reason has a drop of its flow rate there will be a difference between the set point and process value. The difference will then be added to an accumulated error (AE). When the AE increases, this will increase the calculated set point. This could be seen as an integrator that keeps track of exactly how much of the additive that has been mixed in. One way to implement it is to have a PID with its point set to 0 and the AE as process value. The output from this PID is then added to the set point for the main PID controller.

The important in this strategy is that if one additive has some kind of disturbance or variation it will not be forwarded to the other additives. The only controller that is allowed to react is the one where the error has appeared. This will give an error in the outgoing product for the moment but this will be compensated for over time. Since no other controller should react on the error the additive with the error still have to report that it mixes the correct volume which in fact means that the set point for the flow is forwarded instead of the measured value. The total flow/skimmilk flow calculation also has to be changed.

A downside with this strategy is that it allows variations in the total flow rate. If this method is used in a system where the outflow is controlled by the pasteuriser the skimmilk flow rate will increase because of the pressure drop. This will affect all controllers and destroys the idea of the method.

Chapter 7
User interface

An important part for the operator of the machine is of course the user interface. This is sometimes called HMI (Human-Machine Interface), MMI (Man-Machine Interface) or for computers GUI (Graphical User Interface). Henceforth, it is always referred to as user interface or HMI.

To design a good user interface some guidelines and rules of thumb should be followed. These will however not guarantee some sort of *perfect* interface, only a basis for evaluation and further development. This procedure is called iterative design and the goal for each iteration is to get a better interface than before. In this thesis the design process itself is not discussed, only the analysis method.

Some theory behind evaluation and analysis is presented in this chapter along with some guidelines and rules of thumb. These are then applied to the currently used interface. The presented methods are heuristic evaluation and cognitive walk-through. One commonly used analysis method, that is not discussed at all, is user tests. For later stages in the design process this should however be used in some way. Especially interesting could be to see how easy users of an older model of the system can adapt to the new model.

7.1 Theory

Since design of interfaces is not really a science there is no right and wrong. Many different (and sometimes conflicting) theories exist. Here, some of the most accepted ones are presented.

7.1.1 Usability

One term often used for good user interfaces is user friendly. This should however not be used as there is no widely accepted definition of it. A better term is instead usability that is defined by ISO 9241-11 as:

> *"The extent to which a product can be used by specific users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."*

This definition includes three key words: effectiveness, efficiency and satisfaction. Effectiveness is defined as to what extent a goal or task is fulfilled. Efficiency is the amount of effort that has to be used to accomplish the task; less effort gives higher efficiency. Satisfaction is the amount of positive feelings that the use of the

system generates.

Usability can also be used as a part of the wider definition of usefulness which is defined by Jakob Nielsen, [Nielsen, 1994], as the sum of the utility and the usability. Utility is defined as to what extent the functions needed to achieve the specified goal are implemented. Usability is how well the user can use these functions. Nielsen defines usability with the following points [Nielsen, 1994]:

- Easy to learn – New users can quickly make use of the system.
- Efficient to use – Experienced users can complete tasks fast.
- Easy to remember – If a user already has done something in the system; it is easy to do it again, without starting from scratch.
- Few errors – Errors are prevented from occurring but if an error occurs it is easy to recover from the error.
- Subjectively pleasing – The user is subjectively pleased when using the system. There is nothing disturbing.

7.1.2 Heuristic evaluation

One way of evaluating the usability of a system is heuristic evaluation. This involves investigating the system according to some broadly principals, so called heuristics. These can be seen as guidelines or rules of thumb. Some of the most common sets of heuristics are presented here.

Nielsen's rules of thumb

Jakob Nielsen has presented ten rules of thumb that can be used when evaluating an user interface [Nielsen, 1994]:

- **Simple and natural dialogue.** Simplify if possible, every additional feature or item gives the user one more thing to learn and one more thing to possibly misunderstand. The information on the screen should be enough for the user to accomplish the task but not more. Of course there should be a possibility to get more information if wanted, but not before. Less is more; too much information does not only risk confusing the novice user, it also slows down experienced users. Graphic design and colours should be limited; there should not be more than 5 to 7 different colours and colour blindness (7% of all males and <1% of all females [Capio Medocular AB, 2006]) should be taken into account.
- **Speak the user's language.** Terminology used in the system should be based on the targeted users' language and not the system itself. As far as possible common non-technical language should be used. Abbreviations should be avoided if they are not widely known and accepted.
- **Minimize user memory load.** Humans easier recognise things that are shown to them than things they have to recall from memory. For that reason the system should give options to choose from rather than expect a keyboard input or similar. One way is by having menus.
- **Consistency.** A command or action should always do the same thing. All dialogues, messages, menus etc. should be the same throughout the whole system. Consistency can involve colours, terminology, layout, display of data and input of data.
- **Feedback.** All actions should give some sort of informative feedback. If a parameter is changed this should be shown in some way, e.g. by changing

colour of the value. Buttons that are pushed must inform that something has happened; it can be anything from opening a window to shutting down the system. Common actions do not need very much feedback but more unusual and serious actions should give much feedback, allowing the user to understand the operation. Irreversible actions should also include a confirmation question before they are performed. When operations take some time to complete there should be some indication that the system is working; it could be an hour-glass for relative short operations and a percentage indicator for longer operations (A rule of thumb is that operations taking longer than 10 s should have a percentage indicator).

- **Clearly marked exits.** Always give the user a clearly marked exit from a dialogue, allowing the user to feel in control of the system and not the other way around. A common way is to have an abort button.
- **Effective use.** Experienced users should have the possibility to use shortcuts to accomplish tasks fast.
- **Good error messages.** An error message should be clear and precise. The user should not have to refer to manuals to understand the error message. If error codes of some sort are used, these should be accompanied by an explanation of the error. The error message should also give some help in solving the error. Some sort of *More information*-button can be used when the user is interested in more detailed information. The system should also give the possibility to reverse the last action.
- **Prevent errors.** The user should be given some choices instead of having to use raw input, i.e. keyboard input. Serious actions should be confirmed by a dialogue. This should however not be used too often as it will only make the user to automatically press *OK* or *Yes*.
- **Help and documentation.** User manuals are not used by many users. There should be some sort of online help in the system, allowing the user to search and understand certain functions and errors. A help button for dialogues also allows users to get more information about the function.

Schneiderman's golden rules

Ben Schneiderman has presented eight rules for interface design [Schneiderman, 2004]:

- **Strive for consistency.** Dialogues and screens should look the same throughout the system. Use of terminology, colours and fonts should be consistent.
- **Enable frequent user to use shortcuts.** Decrease the amount of interactions to accomplish a task for experienced users.
- **Offer informative feedback.** All actions should give some feedback. The distinctness and information of the feedback should reflect how serious and frequent the action is. Minor and frequent actions should only give modest feedback while major and infrequent actions should give more information.
- **Design dialogues to yield closure.** All tasks should be organised into groups, giving a beginning, middle and end for the dialogue. It is important that the user knows when a task is at its end, allowing the user to go on with the next task.
- **Offer simple error handling and prevent errors.** Design the system such that serious errors can not be done. If an error is made, offer concise and clear information about the error and be give some instructions on how to proceed.
- **Permit easy reversal of actions.** An action should always be possible to

undo, independent if an error has occurred or not.
- **Support internal locus of control.** The user should always feel in control of the system. Unexpected events and lack of information makes the user anxious and feeling out of control.
- **Reduce short-term memory load.** A human has a limited short-term memory; a rule of thumb is that a maximum of 7±2 discrete objects should be used at the same time.

Norman's features

Donald Norman has some further points that should be taken into account. He has also defined the concept of user-centered design. Some of his points, the ones that are not covered by Nielsen and Schneiderman, are presented here as wanted features of system [Norman, 1998]:
- **Visibility.** The most important parts of the interface should be the most visible. The user should be able to determine the functions available and the state of the system. An experienced user can handle more information on the screen than a novice user, allowing faster access to interesting information and actions.
- **Affordance.** Affordance is what an object offers the user to do with it, i.e. what signals or clues the object gives to the functionality.
- **Mapping.** The appearance of the object should have some connection to the real world. It could be an appearance that has resemblance to a physical object or a cultural standard, e.g. red implies a warning. It could also be as simple as a button should look like a button if users can click on it.

7.1.3 Cognitive walkthrough

Since much of what is normally included in a cognitive walkthrough already has been mentioned in the heuristic methods above, only some of the parts of the cognitive walkthrough are mentioned here. [Hellström, 2005]

Perception

The human brain always tries to make out structures and patterns. By thinking of some principals of grouping [Mullet & Sano, 2004], illustrated in Figure 7.1, this can be applied when designing a user interface:
- **Proximity.** Objects that are close to each other are more likely to be grouped together than objects that are not as close.
- **Similarity.** Objects that are similar are more likely to be grouped together than objects that are not similar.
- **Continuity.** The brain rather sees a continuous object than a fragmented. In the figure you are more likely to see two lines crossing each other than four lines meeting in the middle.
- **Closure.** Even though an object is not completed, the brain tries to close the object. In the figure a triangle is perceived even though the lines are not completed.
- **Area.** When two objects overlap, the smaller object is interpreted as being on top of the larger object.
- **Symmetry.** The mind prefers objects with symmetry, making the example in the figure two overlapping squares rather than three polygons.

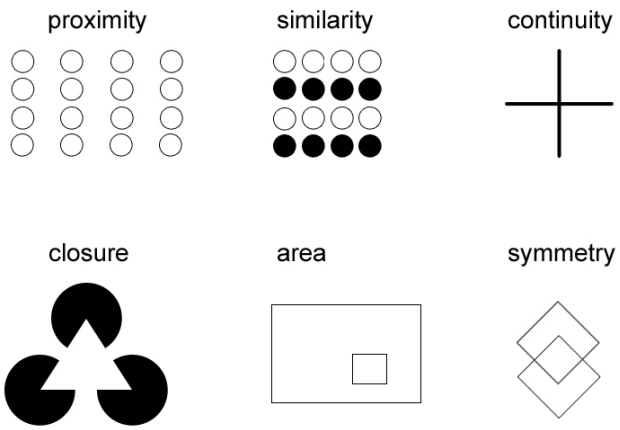

Figure 7.1: Perception examples.

Other ways of grouping together objects are by colour, frames and lines. Frames and lines should however be used with care, since they can take some of the attention away from the important parts of the interface.

Text

When reading a text on the screen it is important to have some lines for the eye to follow. This could be accomplished with spacings between the text lines. The text should also be aligned to the left side and be read from top to bottom and from left to right. The text should not be to thick, i.e. not to long sections or sentences. It should not either be too small, making it hard to read. When data is presented on the screen it is important that the user can control the display in some way, e.g. by sorting.

Attention

There are numerous ways to get the users attention. Some are mentioned here:
- Emphasise of text (bold/italic).
- Markings of text (underline).
- Size of text (not more than four different sizes).
- Fonts for text (not more than three).
- Colours (not more than four).
- Sounds.
- Blinking or flashing (only for very serious and infrequent events).

Colours

When using colours it is very important to not use too many colours at the same time. As mentioned above, not more than four colours should be used in the same window. It is also important that the colour corresponds to the user's real-life

colour perception; i.e. red for errors, yellow for warnings, green for OK and so on. There should however not be an over-reliance on colours; symbols should be able to work in gray-scale as well.

7.2 Analysis of current system

The user interface for a blending system includes many functions and menus. The most frequently used are the graphical system overview, the recipe management and the alarm handling. The graphical system overview gives the operator a realistic overview of the physical system, with components like valves and sensors illustrated with possible error signals. The recipe management makes changes of products and allows setup of the production. The alarm handling consists of an event list with warnings and alarms with different levels of risk. This analysis will specialise on these three functions. With the heuristic evaluation methods and the cognitive walkthrough in mind, a usability analysis is done.

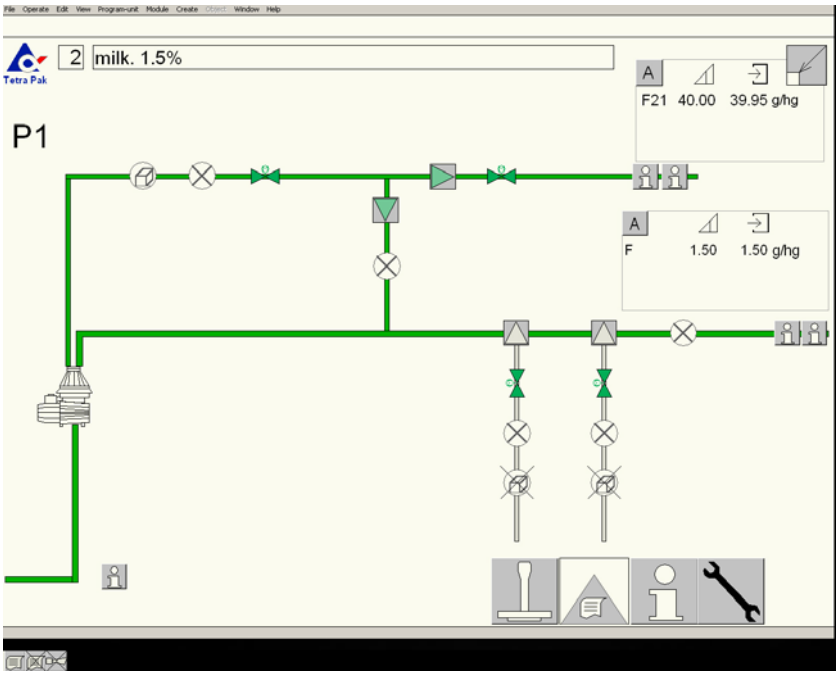

Figure 7.2: The user interface overview.

The system overview shows the different valves, flow meters, density meters and tubes. It also shows the separator, giving the operator an idea of where the blending system is placed in the larger dairy system. As can be seen in Figure 7.2, the large buttons at the bottom makes some impression of being important. This is also the case since the configuration menu, alarm list and information can be accessed through them. If the *i*-button is clicked, more information about the different objects will be shown to the user, see Figure 7.3. This is a good way of hiding information that is not needed very often. An even simpler mode can be accessed through the button in top right corner. This mode only shows

the production output, see Figure 7.4. These different modes give some sort of information levels where different users can access the information needed.

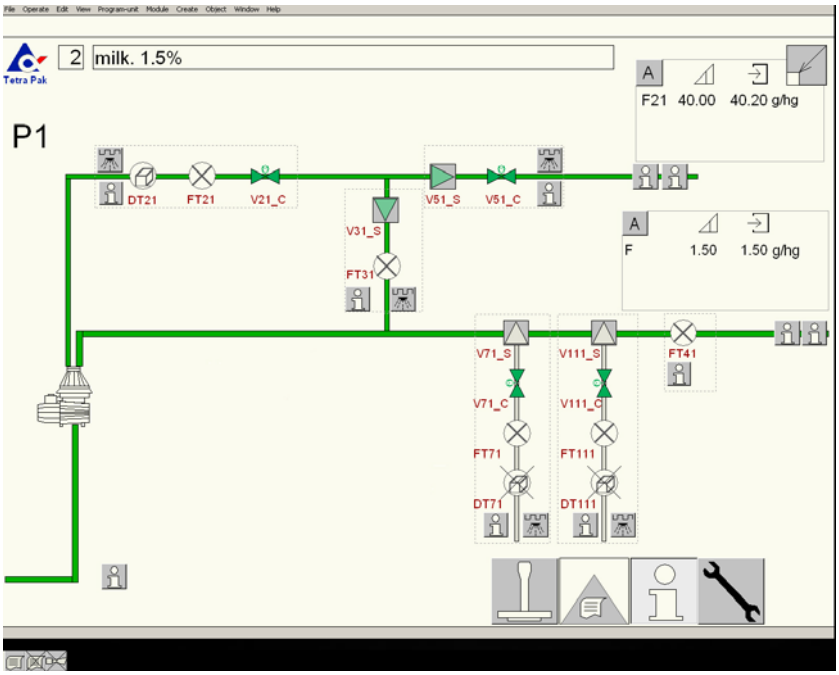

Figure 7.3: The user interface overview with more information showing.

An important feature for a user interface is consistency, giving the user a hint of what different buttons and objects do in different views. Generally, different windows are built the same way. One example of this is the recipe and ingredients management, shown in Figures 7.5 and 7.6, where the layout is similar in both windows. There is also a similar layout when editing the recipe and the ingredients, see Figures 7.7 and 7.6.

How to come from the recipe management to the ingredient management or list is not clear. The clickable area for this is shown in Figure 7.9, with marked out areas for what different clicks will do. Since the border between these areas is placed in the middle of the word *Ingredients*, it is not very intuitive to know what will show up. It would be better to have some kind of marking or even a button to better show the different actions. This is an example of bad mapping. Under the word *Ingredients* the ingredient number and name is displayed. These are also clickable and it is actually here one is supposed to click. If these were made into buttons it would be better.

Another example of bad mapping is apparent in the overview window. Almost all functions have an icon or button, but the short cut to come to the recipe list and management is not as clear. In the top of the overview window, Figure 7.2, it is the number and recipe name with borders around it. It would be better to either make two buttons out of these or to have two separate buttons next to the borders.

Both these bad mappings have actually been fixed in a later version of the user interface.

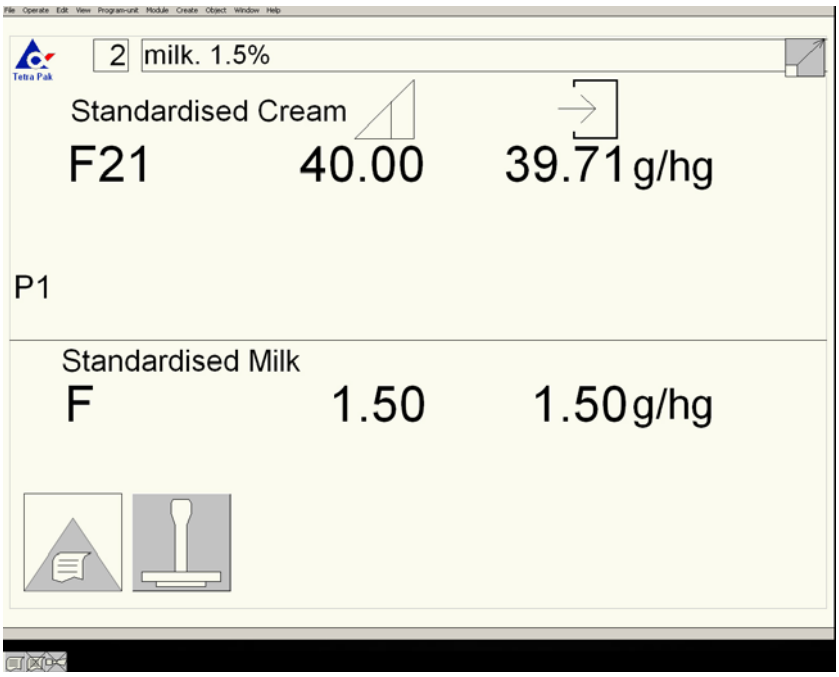

Figure 7.4: Simple product value presentation.

The objects are, in general, grouped good and used consistently. However, the icons (the appearance) of some objects are not very self-explanatory. For a new user, it can be confusing when searching for a certain function. Some examples of confusing use of icons can be seen in Figures 7.10 and 7.11. The On/Off-button shows Off when it is off and On when it is on. To switch the button from off to on, you have to click on a button that shows Off, which feels wrong. It would be better to have a button without the On/Off text and instead have this status displayed next to the button. The other example is the Save/Cancel-buttons, which have the same icons. It would be better to have different icons or the actual text (Save/Cancel) in the button. All objects should also have some kind of hint or description appearing when holding the pointer over the object. This could however be a problem when using a touch screen without a pointer.

A touch screen gives some limitations on how small objects can be. In the current system, this problem is especially detected when closing a window with the close button (the X in the top right corner). Sometimes it can be a little tricky to click the button. An alternative to the Save/Cancel-button functionality that is used today (the buttons appears when a change is made) is to have them visible all the time. If one of the buttons is pushed in the current system, the window will close. This is not very intuitive since they do not appear until a change has been made. If they would always be visible, perhaps as the OK/Cancel-buttons that is used in Microsoft Windows, this would help. The Cancel-button would then also provide a solution to the close-button problem mentioned earlier.

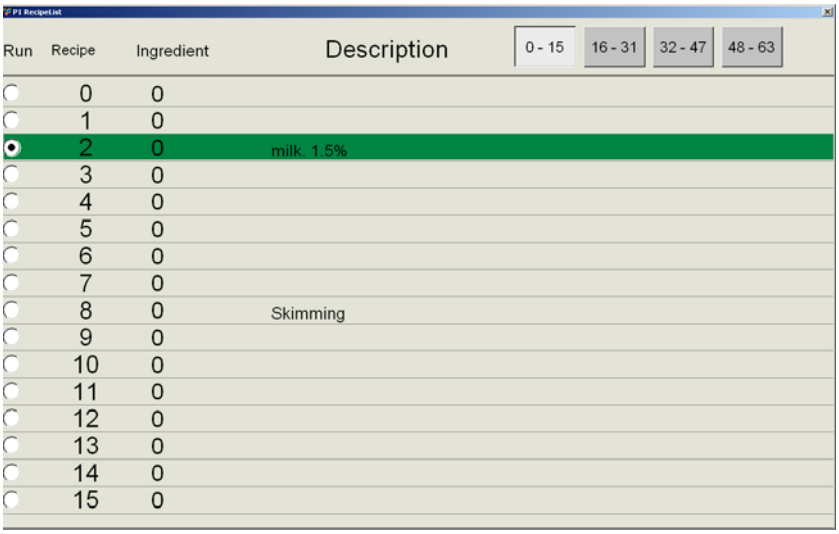

Figure 7.5: The recipe list.

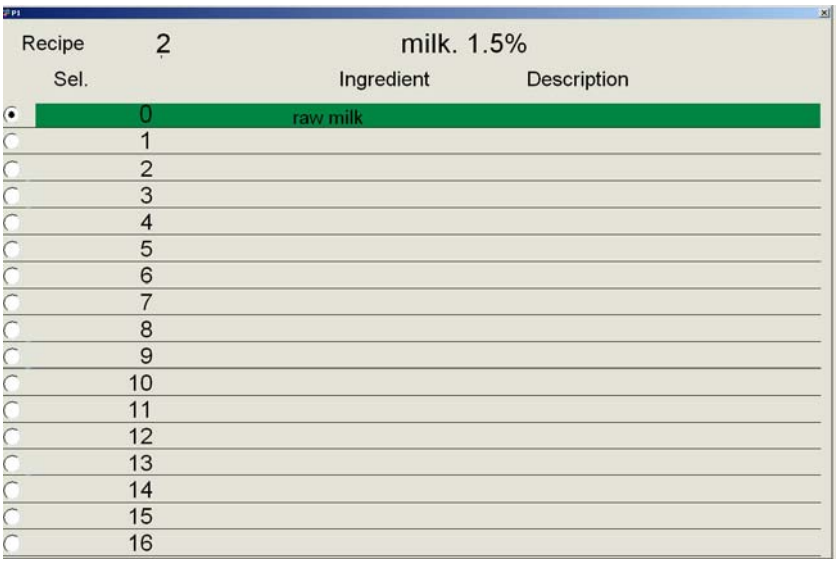

Figure 7.6: The ingredient list.

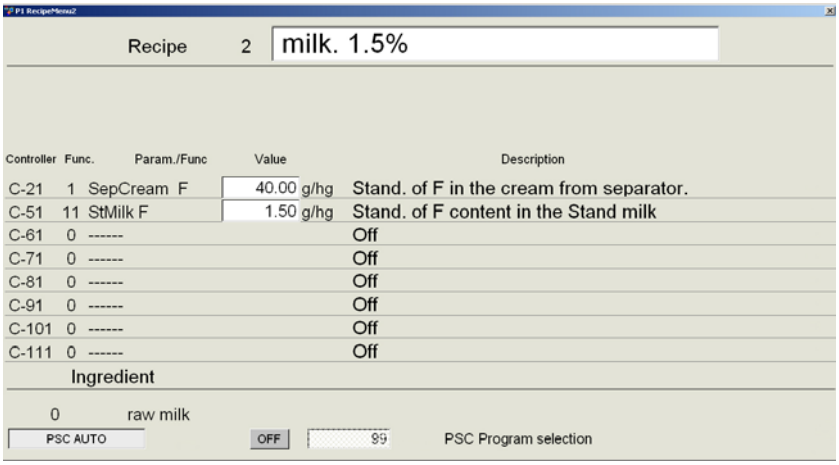

Figure 7.7: Edit window for a recipe.

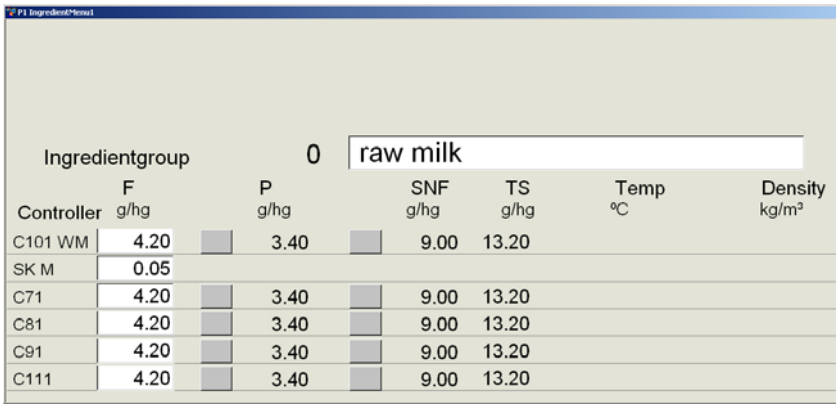

Figure 7.8: Edit window for ingredients.

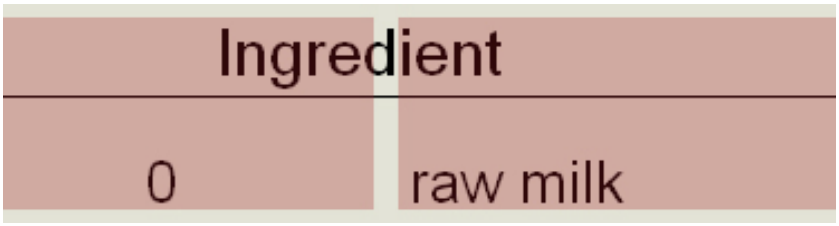

Figure 7.9: The clickable area for the ingredient management where the left marked area leads to the choice of ingredient group while the right area leads to the ingredient editing.

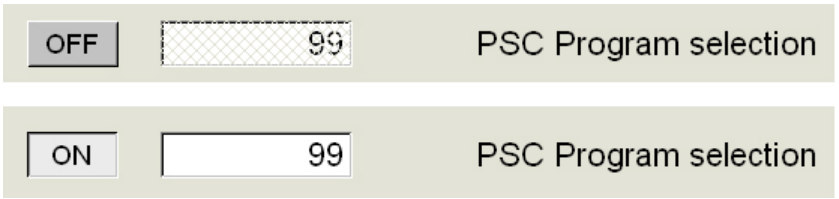

Figure 7.10: On/Off-button.

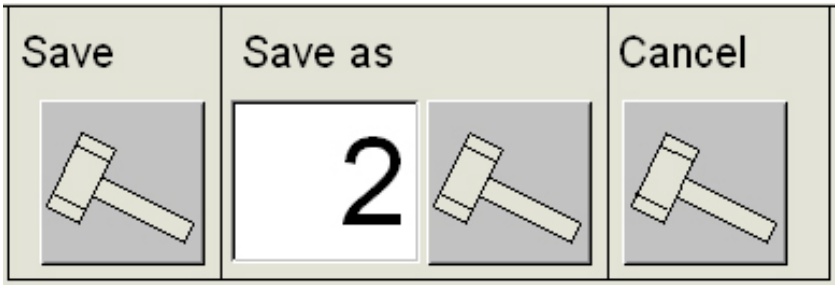

Figure 7.11: Save/Cancel-buttons.

A user interface should always provide some kind of feedback for every action, giving the user a sense of being in control. Feedback can be as simple as opening a window or switching a status indicator. More long-time actions should always provide an operation progress indication, e.g. showing an hour-glass or some percent bar. In this system there are not a lot of long-time actions but one is the alarm acknowledgement feature. When this button is pushed it will be in the down position for some seconds, while acknowledging all alarms. It would be better to change the pointer to an hour-glass or showing something like "Acknowledged 1/3 alarms.", counting up.

To get efficient usage out of an interface, some regard to a human's memory capacity has to be taken. If possible, a human should never have to write in raw data, e.g. a function number. This should always be controlled by some kind of choice. In the current system this is done very easy but some things could be better. When changing a value, a keyboard has to be used. For touch screen a virtual keyboard can be used if no keyboard is present. Another way of solving this is to use some kind of up/down arrows to increase or decrease the value.

One part of the interface that is often used is the alarm handling. Alarms can come from all parts of the system and theories behind alarm management are numerous. An important part is of course the presentation of alarms for the user. An example of how the alarm list can look like is shown in Figure 7.12. As can be seen there is some kind of text describing the alarm. There is however not any way to get more information if wanted. When clicking an alarm, there could be some extra information and perhaps some suggestion on how to proceed. The alarm list itself is not very pretty, since the columns in the alarms are not aligned properly. An option could also be to have some kind of Undo-button, undoing the last made change.

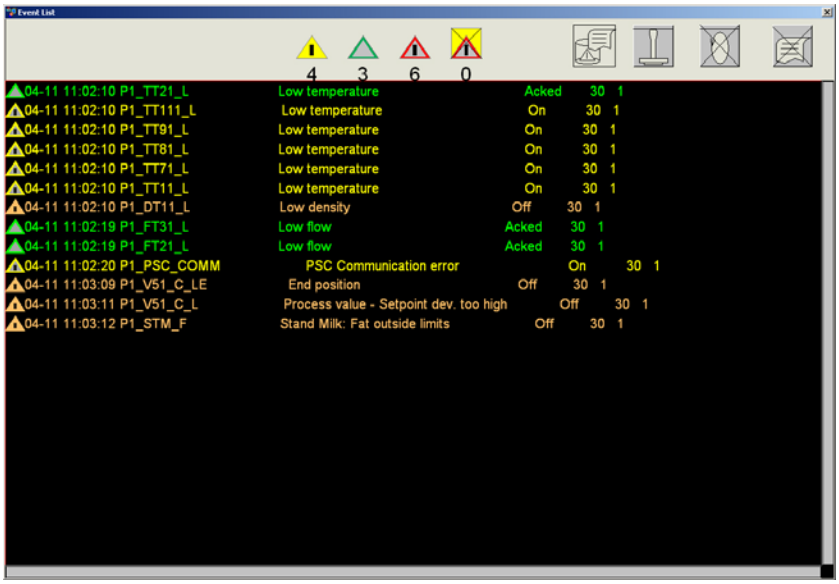

Figure 7.12: The alarm list.

Chapter 8
Summary and conclusions

During the latest years the number of additives wanted in a blending process has increased. This gives two main problems. The first is that different additives are dependent of each other which make the control task more difficult to handle. The second problem is that when developing a new system it should also be easy to add another additive which requires a flexible program structure.

In Chapter 5 a flexible program structure has been developed which takes care of the control calculation and the recipe management. This program structure simplifies the calculation by using fat mass flows instead of fat contents when calculating set points. The storage of recipes and ingredients are distributed to the additives, allowing a dynamic recipe handling.

Since an additive is dependent of all other additives there will be some stability problems when set point changes are made, at least for systems with many additives. An approach on how to use a different control strategy that has focus on control robustness is presented in Chapter 6.

The program structure and control calculation mentioned above might be implemented in any PLC-system. But to be able to use a PLC-system it also has to fulfill specifications for performance, memory capacity and I/O-connectivity. Because of the high performance demands, the concept of SoftPLC has been investigated and some basic requirements that a SoftPLC-system (including the industrial PC) should fulfill have been worked out in Chapter 3. Regarding if SoftPLC could and should be used, the answer is yes respective maybe. SoftPLC gives more performance and memory capacity for less money. It is however not extremely much faster, only 5-10 times, and there are still some questions about the reliability of the industrial PCs. A great advantage is if the SoftPLC manufacturer also provides PLC hardware, making it possible to easy go from a SoftPLC solution to a PLC solution if the reliability will prove to be less than satisfactory.

In Chapter 4, an investigation is done of PLC- and SoftPLC-systems. This is mainly focused on large-scale manufacturers like Rockwell, Siemens and ABB but some alternative systems are also discussed. The investigation is based on a requirement specification that has its origin in the system used today and its performance and accuracy. The two main alternatives are ABB and Siemens. ABB because of the development environment and that it is the system used today. Siemens because of the real-time kernel in the SoftPLC and the use among other Tetra Pak machines. Since an implementation should be done in one system, ABB is chosen mainly because of the easy-to-use development environment and the simulation possibilities. It should be stated that our choice is not Tetra

Pak's choice and our investigation is only a technical basis for the choice of system.

As a final part of the thesis, the user interface was analysed using some standard review methods. Some things were found that could be done better. Some of these have however already been fixed in a later version.

8.1 Future work

An investigation of how more complex automatic control, e.g. multivariable controllers, could be used to get a more optimised mixing.

If SoftPLC should be a realistic option, there have to be extensive testing of the industrial PCs. An investigation of how Windows XP Embedded should be setup for best performance together with the SoftPLC should also be done.

Bibliography

[ABB, 2006a] ABB (2006a).
Webpage: http://www.abb.com/.

[ABB, 2006b] ABB (2006b). Control Software and Control Builder M - Product Guide. ABB.

[Årzén, 2003] Årzén, K.-E. (2003). Real-Time Control Systems. Department of Automatic Control.

[Beckhoff, 2006] Beckhoff (2006).
Webpage: http://www.beckhoff.com/.

[B&R Automation, 2006] B&R Automation (2006).
Webpage: http://www.br-automation.com/.

[Bylund, 1995] Bylund, G. (1995). Dairy Handbook. Tetra Pak Processing Systems AB.

[Capio Medocular AB, 2006] Capio Medocular AB (2006). Färgblindhet.
Webpage: http://www.medocular.se/ogonfakta/fargblindhet/.

[Fischer, 2005] Fischer, P. (2005). Extending Windows XP into real time.
Webpage: http://www.embedded-computing.com/articles/id/?230.

[Hellström, 2005] Hellström, A. (2005). Safe-Doc – A Usability Evaluation with Suggestions for Improvements. Department of Design Sciences, Lund Institute of Technology.

[James-Damato, 2006] James-Damato, R. (2006). Thinking of Using Microsoft Windows NT or XP Embedded?
Webpage: http://www.automation.com/sitepages/pid1083.php.

[Mullet & Sano, 2004] Mullet, K. & Sano, D. (2004). Design Visual Interfaces: Communication Oriented Techniques. Prentice Hall.

[Nielsen, 1994] Nielsen, J. (1994). Usability Engineering. Morgan Kaufmann Publishers.

[Norman, 1998] Norman, D. A. (1998). The Design of Everyday Things. MIT Press.

[Olsson & Rosén, 2000] Olsson, G. & Rosén, C. (2000). Industrial Automation. Department of Industrial Electrical Engineering and Automation.

[Rockwell Automation, 2006] Rockwell Automation (2006). Webpage: http://www.rockwellautomation.com/.

[Schneiderman, 2004] Schneiderman, B. (2004). Design of the User Interface. Pearson Higher Education.

[Siemens, 2006] Siemens (2006). Webpage: http://www.siemens.com/.

[Venturecom, Inc., 2003] Venturecom, Inc. (2003). Hard Real-Time with Venturecome RTX on Microsoft Windows XP and Windows XP Embedded. Webpage: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxpesp1/html/tchxpesp1TechArt.asp.

Glossary

Additive An additional inflow, normally all inflows except the skimmilk and cream flow.

GUI Graphical User Interface.

HMI Human-Machine Interface.

I/O Inputs/Outputs. I/O-modules consist of some kind of inputs or outputs that are connected to instruments like sensors and actuators.

MMI Man-Machine Interface.

NIR Near Infra Red. An imaging method used for measuring protein contents in milk.

PLC Programmable Logic Controller.

PT100 A standardised temperature sensor.

RTOS Real-Time Operating System.

SNF Solids Non Fat; the solids in milk except the fat. SNF consists of proteins, lactose and ash (minerals). SNF is about 9% of cow milk.

SoftPLC Software PLC. Also called PC based automation or PCbA.

Appendix A

Dairy introduction: From raw milk to packaged product

This appendix is primarily written for those who do not have experience of a modern dairy production line. A brief overview of the dairy is given. An illustration of the product flow in the dairy can be viewed in Figure A.1. The largest information source for this appendix is [Bylund, 1995].

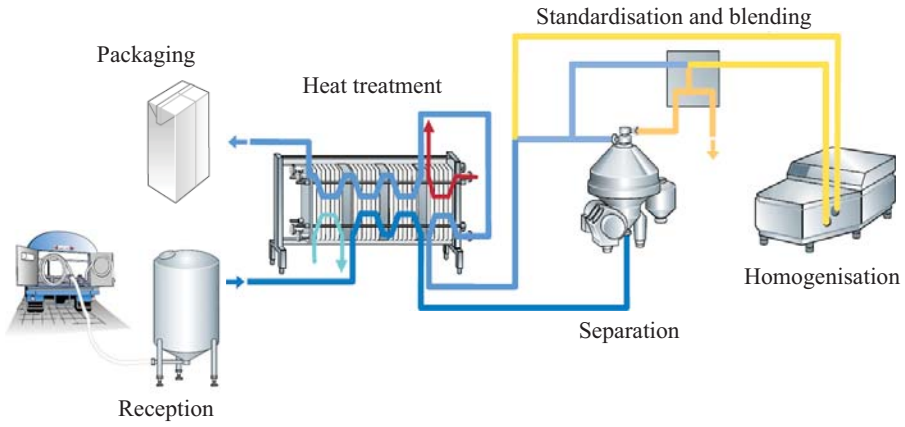

Figure A.1: Overview of the different parts of the dairy.

A.1 Composition of milk

Milk directly from the cow, called raw milk, consists of water, fat, proteins, lactose, ash (minerals) and some solid particles like hair and cow cells. The solid particles are disposed of when going through the separator. The milk contents can then be divided into three main components:

- Water. (87% of milk)
- Fat. (4% of milk)
- Solids Non Fat (SNF) (9% of milk) - Proteins, lactose and minerals.

Water and fat are fairly self-explaining. SNF is however more complex. Proteins are mainly casein (80%) and whey protein (19%). Lactose, milk sugar, is

about 5% of the milk and minerals is about 1% (also called ash since it will be the remaining part if milk is incinerated).

A.2 Reception

After milking, the milk is cooled to a temperature of about 4℃ to prevent growth of micro-organisms. Under transport to the dairy it is increased slightly above 4°C. Incoming milk is because of that re-cooled before storage in silos.

A.3 Heat treatment

Most micro-organisms found in milk have growth temperatures between approximately 10℃ and 60℃. There are also some heat-loving bacteria that endure temperatures above 70℃. When cooled the growth is prevented, but the organisms are not terminated. For this a high temperature treatment is needed, normally a pasteurisation. Pasteurisation (HTST - High Temperature Short Time) consists of a rapid heating of the milk to about 72-75℃ for a time of 15-20 s. This combination of temperature and time destroys the tubercle bacillus (T.B.) which is the most resistant pathogenic organism normally occurring in milk. One way of checking if milk has been properly pasteurised is to measure the activity of the phosphatase enzyme. If no activity can be detected the milk is considered to be adequately heated.

There are also other heat treatments available, e.g. LTLT (Low Temperature Long Time) pasteurisation which uses a lower temperature during a longer time (this is however not used in continuous processes because of the long heating times) and UHT (Ultra High Temperature) which uses a very high temperature of about 140℃ for only a few seconds.

The heating and cooling process is actually done in several steps; see Figure A.1, consisting of heat exchanger segments. A heat exchanger conist of a number of plates with tight spaces in between. Every second of these spaces is fed with a hot medium and the other is fed with the substance you want to heat. The heat will then be transferred to the cooler substance with conduction through the plate.

The first step is the heating of the incoming milk to about 55-65℃. This is done by some of the finished product. When the milk has gone through the separator, standardisation, blending and homogenisation (see following sections for more detailed information) it is heated to 72-75℃. It then goes through a delay pipe which will hold the temperature for the desired time, normally 15-20 s. After this heating the milk is rapidly cooled again to about 4℃, first by cooling with the incoming milk and then by ice-water. The regenerative heating and cooling that is used saves as much as 90% of the heat energy.

A.4 Separation

Raw milk consists of two main components; cream and skimmilk. One important difference between these is their density. If whole milk is poured into a glass and this stands for a while, the cream will travel to the top of the glass while the skimmilk will reside on the bottom. This process depends on that cream is lighter than skimmilk and is called sedimentation. Sedimentation uses gravity as

its force of separation. One way of making artificial gravity is by rotation which is used in modern separators. The centrifugal force pulls the heavier skimmilk to the peripheral of the separator and the cream is forced into the middle. Figure A.2 shows the flows inside the separator. Since the milk is not purified before separation some particles from the cow still exist in the milk, e.g. hair. Since these have the higher density than both skimmilk and cream, they will be forced into the peripheral of the separator. There they will remain while the separator is rotating. To get rid of this litter the separator either has to be opened and cleaned or an in-production cleaning system is used. This system allows the bottom part of the separator to be lowered, creating a small opening along the side where the particles exit. After a fraction of a second it is then forced up to its original position again, leaving the litter particles outside the separator. This is called a discharge.

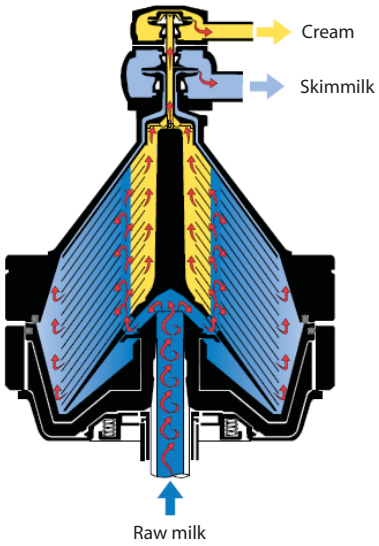

Figure A.2: Flows inside the separator. The inflow is raw milk and out comes cream and skimmilk.

A.5 Standardisation and blending

After the separation, the milk is divided into two main components; cream and skimmilk. Cream has normally a fat content of about 40% and skimmilk of about 0.05%. If milk with 3% is wanted, some of the cream has to be remixed with the skimmilk in order to get a final product with standardised fat content. This could be done manually without inline mixing by calculating how much cream and skimmilk that should be mixed together. More common, at least in large scale dairies, is to mix the components while flowing. To do this accurate, many properties has to be measured and taken into account; i.e. flows in different tubes, temperatures and density of the liquid. Also infrared sensors can be used to measure fat and protein content. Using feedback control, the flow is regulated to obtain a uniform fat content throughout the production.

For more complex products, additional ingredients are needed. This could be extra cream, flavours or proteins. These are added to the product flow according

to predetermined recipes. The fat contents of these additional flows have to be compensated for, to obtain a consistent fat content in the final product.

Besides from fat content, milk can also be standardised according to protein. This is done with the same mechanisms as for fat content but requires precise measurements of either density (really SNF standardisation) or by NIR-technology.

For more detailed technical process description, see chapter 2. There are also some predictions on future process requirements.

A.6 Homogenisation

To prevent large clusters of fat globules in the milk and to get a more appetizing colour a homogeniser is used. It disrupts large fat globules by forcing them through a narrow gap under high pressure (100-250 bar, normal pressure in the tubes is about 3 bar). One disadvantage with homogenisation is that homogenised milk can not easily be separated. This could be a problem if milk is going through the separator a second time, which is sometimes the case.

A.7 Packaging

The package of the final product has some important functions, where the most important is to preserve product quality. This allows transportation of the product from the dairy to shelves around the country. A normal package for milk consists of paperboard with a thin layer plastic on both sides. The paperboard gives stiffness, resistance to mechanical shocks and light while the plastic (polyethylene) makes the package leakproof. It also protects against condensation. For products with long shelf-lives a thin layer of aluminium foil is used to further the protection against light and oxygen.