CHALMERS
UNIVERSITY OF TECHNOLOGY

Decision Support Systems for Operators within the Process Industry

Development of a Concept Using Online Modelica Models

Master's thesis in Systems, Control and Mechatronics

TEODOR OLSSON

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2014
EX053/2014

REPORT NO. EX053/2014

Decision Support Systems for Operators within the Process Industry

Development of a Concept Using Online Modelica Models

TEODOR OLSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Division of Automatic control, Automation and Mechatronics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2014

Decision Support Systems for Operators within the Process Industry
Development of a Concept Using Online Modelica Models
TEODOR OLSSON

Report no. EX053/2014
Department of Signals and Systems
Division of Automatic control, Automation and Mechatronics
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 (0)31-772 1000

Department of Signals and Systems
Gothenburg, Sweden 2014

Abstract

As demands on efficiency and cost reduction within the process industry are increasing, new ways to optimize production are desired. The result of the optimization work is often reduction of energy buffers in conjunction with new operating procedures. Process systems are to a large extent controlled automatically, but parts of the system are still controlled by human operators. Increasing complexity and higher demands mean that it gets more difficult for the operators to make the correct control decision. Therefore, an interest has arisen for developing different solutions to aid the operators in their tasks.

The aim of this thesis is to develop a suitable concept for the design of a model-based decision support systems (DSS) to provide predictions of the future state of the plant to the operators. By allowing the operators to interact with the simulations, the effects of control changes can be studied and optimized before applying the control actions to the real system. The main factors taken into consideration during this work have been requirements for the models, handling of plant measurements and how the system should be implemented.

The proposed DSS concept makes the predictions by initiating a plant model developed in Modelica to a starting point representing the current state of the plant. Simulations of the model are then performed to provide predictions of the future behavior. The starting points for the predictor simulations are continuously generated from measurement data collected from the plant.

To evaluate the proposed concept, parts of the DSS have been implemented in a case study for a fictitious thermomechanical paper mill. Three different internal models were developed and the performance was evaluated. Two different techniques were evaluated for the estimation of the starting point, one simple filtering method and one more advanced method, where state estimation was performed using an extended Kalman filter. The results from the implementation of the DSS in the case study show that the construction of a DSS using the proposed system is promising, but much work still remains before a DSS could be designed for a real system.

Keywords: Decision Support Systems, Online Modelica Models, Nonlinear State Estimation, Process Simulation

Acknowledgements

I want to thank all the employees at Solvina AB for providing a pleasant workplace for me to do my thesis. Especially, I want to express my gratitude to my supervisor Carl Ressel for his help and guidance.

At Chalmers, I want to thank my examiner Torsten Wik for providing helpful advice and feedback.

Finally, I wish to thank my family for their support throughout my thesis work.

Contents

List of Figures

List of Tables

Abbreviations

In this section the abbreviations used in this thesis are presented.

DAE	Differential algebraic equation
DDE	Dynamic Data Exchange
DSS	Decision support system(s)
EKF	Extended Kalman filter
FMU	Functional Mock-up Unit
KF	Kalman filter
MHE	Moving horizon estimation
OPC	Object Linking and Embedding (OLE) for Process Control
TMP	Thermomechanical pulp
UKF	Unscented Kalman filter

REF	Pulp refiner
PT	Pulp tank
BL	Bleachery
BL.T1	Bleachery tank 1
BL.T2	Bleachery tank 2
PM	Paper machine
WT1	Return water tank 1
WT2	Return water tank 2
HWP1	Hot water production line 1
HWT1	Hot water tank 1
HWP2	Hot water production line 2
HWT2	Hot water tank 2
HWT_PID	Hot water tank PI-controller

1 | Introduction

This chapter gives an introduction to the topic and the motive of this thesis. The main objective is presented as well as the delimitations. Last in the chapter, the disposition of the report is presented.

1.1 Background

Process industries are generally large and energy intensive systems. Due to rising fuel and energy prices, large efforts are put into increasing efficiency to reduce costs. Historically, process industries have featured large energy buffers and overcapacity, which is now removed as part of the energy optimization. Reduced reserve capacity, in combination with new operating procedures, increases the complexity of the systems and introduce stricter requirements on the control systems.

The control of modern process industries is largely automated, although the systems still require human operators. Since the introduction of computer control the operator tasks have shifted away from simple tasks to more advanced ones such as supervision, optimization and to intervene when problems occur. To be able to do their job, the operators scan large amounts of information presented through different operator interfaces. The operators rely heavily on their training and previous work experience in the decision making process.

The more complex systems with *humans in the loop* inevitably mean that it gets increasingly difficult for the operators to make correct control actions. One possibility to aid operators with their tasks is to introduce different types of decision support systems (DSS).

Solvina, an engineering company based in Gothenburg, has extensive knowledge of working with design and optimization within the power and process industry. During the last couple of years, Solvina has seen an increased interest for systems aiding operators among its customers. Therefore, Solvina wants to investigate possibilities for constructing purpose built, model-based decision support systems for operators.

1.2 Objective and Specifications

The objective of this thesis is to propose a suitable concept for a model-based decision support system (DSS) for use within the process industry. The concept should meet the following specifications:

- The proposed DSS concept should work *online*, in parallel to the real process, giving the operators a fast overview of the current state of the plant as well as predictions of future behavior.
- The predictions should help the operators to optimize the process control, ensuring an efficient and reduced energy consumption during production.
- The proposed DSS concept should be evaluated using a case study of an integrated thermomechanical paper mill.
- The proposed DSS concept should be able to serve as a starting point for future development of DSS at Solvina.

1.3 Scope and Delimitations

Development of a complete DSS is a very large project. To ensure the feasibility of the project within the available timeframe for this thesis, the work has been focused on how to make the predictions using a controlled and accurate method.

The main delimitations include:

- **Model as replacement for real plant data**
  Due to difficulties in obtaining data from a real paper mill and to simplify the development of the DSS, a model is used as a replacement in the case study.
- **Operator interaction and user interface**
  An important part of a DSS is the interface used by the operators. Development of a suitable user interface can be considered as separate subject and is not covered here.

1.4 Disposition of the Report

This report begins with Chapter 2 presenting the theoretical background. Chapter 3 covers the plant process that is used in the case study. In Chapter 4 the proposed DSS concept is presented. Chapter 5 explains how some parts of the DSS concept are evaluated using the case study and the results are discussed in Chapter 6. In the final chapter conclusions are presented as well as ideas for future development.

2 | Theory

This chapter provides the theoretical background used.

2.1 Modeling and Simulation

The aim of creating a mathematical model is to describe the interesting behavior of parts of a system using mathematical equations. Model construction of physical systems can roughly be divided into two approaches depending on the principle that is used. With *physical modeling* the system to be modeled is broken down into smaller parts for which the behavior can be described with known physical equations such as laws of motion or thermodynamic laws. The other approach, *identification*, uses statistical methods to create models from measurement data by fitting parameters to a predefined model structure [1].

A model is never a true representation of a system. Each model has a so called *domain of validity* for which the model describes the system sufficiently close. The size of the domain of validity is related to the accuracy required by the intended use of the model. Once a model has been developed it is very important to validate the model to ensure it meets the specified accuracy. Validation of physical models can be done by performing a controlled experiment on the real system, and then compare the results to the same experiment simulated on the model. Models derived using identification techniques can use the same technique for validation. Normally this is done by verifying the model against another set of measurement data separate from the one used to create the model [1].

2.1.1 Modelica

Modelica is a free modeling language for modeling of complex physical systems, developed by the non-profit Modelica Association [2]. Modelica is an object-oriented language which can be used to describe both continuous and discrete dynamics of a system. The language syntax resembles that of other object-oriented programming languages such as Java or C++ [3].

One of the main objectives of Modelica is to provide a language for effective development of models which can be easily reused and shared as libraries for a wide range

of engineering domains. A Modelica model consists of a number of class instances, *components*, structured in an hierarchical structure. Examples of components include tank models or mechanical components [3].

As a specific modeling language Modelica differs in some key aspects from conventional programming languages. One important example is that Modelica offers the option to express relations between variables using equations without the need to specify the causality [3].

The objective of developing in Modelica is usually to provide a model which can be used for simulation purposes. In order to do this, the model has to be presented in a form such that it can be linked together with a numerical integration algorithm, a *solver*, and compiled into executable code. The Modelica specification states a number of rules for how the model structure should be *translated* into a so called hybrid differential algebraic equation (hybrid DAE). A hybrid DAE consists of differential equations describing the continuous time behavior, algebraic equations describing relations between the variables and discrete equations for describing discrete time behavior [3]. An overview of the translation process can be seen in Figure 2.1.

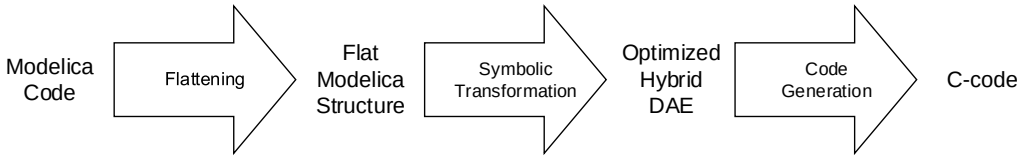

Figure 2.1: Overview of the Modelica translation process. The figure shows the steps of translating a Modelica model into C-code.

The translation process starts with the transformation, *flattening*, of the hierarchical model into a *flat* set of Modelica statements. In this step all equations from the different sub-components are expanded and connections (constraints) between the components are considered. It is also during this part of the translation process that suitable states, x, for which derivatives are formed, are selected [3].

Due to the design of the flattening process, the resulting hybrid DAE contains a large number of sparse equations. In order to simulate the model using numerical methods in a reliable way, the equations have to be restructured to a more suitable form. For this, Modelica specify the use of a symbolic translator to rearrange the equations and to reduce the index of the DAE. Once the symbolic translation has been performed the Modelica code can be compiled into executable code and linked to a solver [3].

When a simulation of a Modelica model is initiated, an *initial problem* first has to be solved to find consistent values for all variables in the hybrid DAE. Modelica offers the ability to specify initial conditions using any of the variables in the model.

There are two ways to provide initial conditions: either via start values for individual variables or as special initiation equations that only holds at the initial time. An example of an initial condition can be that the model should start at steady-state ($\dot{x} = 0$). In this case, the initial values of x will be calculated automatically using equations from the model [4], [5, pp. 121-131].

Once the model has been initiated, the hybrid DAE is solved using the linked solver. Since the models can feature both continuous and discrete dynamics, special techniques are used by the solver. Normally, when no discrete event is active, all discrete variables are kept constant and the model is treated as if it just contained continuous variables. If a discrete event is triggered the integration is halted and the event is processed. After the event, the integration is restarted with new conditions [3].

2.1.2 Modelica Standard Library

Parallel to the development of the Modelica language, the Modelica Association oversees the development of the *Modelica Standard Library* which features approximately 1300 pre-made model components and 900 functions for modelling within multiple domains. The content of the Modelica Standard Library is often constructed for general purpose use which makes it a useful resource when creating Modelica models.

2.1.3 Dymola

Dymola (Dynamic Modeling Laboratory) is a commercial modeling and simulation software from Dassault Systèmes based on the Modelica language. Dymola offers a graphical interface for interaction with Modelica models. The program features two different modes, *Modeling* and *Simulation*. The modeling mode helps the user to design Modelica models using a block or text based interface. The simulation part of the program includes the tools (e.g. symbolic translator and numerical integration algorithms) needed to translate the models into C-code used for simulation [5, 6]. A screenshot of the user interface in Dymola can be seen in Figure 2.2.

2.1.4 Functional Mock-up Interface

The Functional Mock-up Interface (FMI) is a standard for exchange of models between different simulation environments. FMI is linked to the Modelica project, but can be used for other modelling solutions as well. The first version of FMI was released in 2010 by Daimler AG in an attempt to simplify the exchange of simulation models between suppliers.

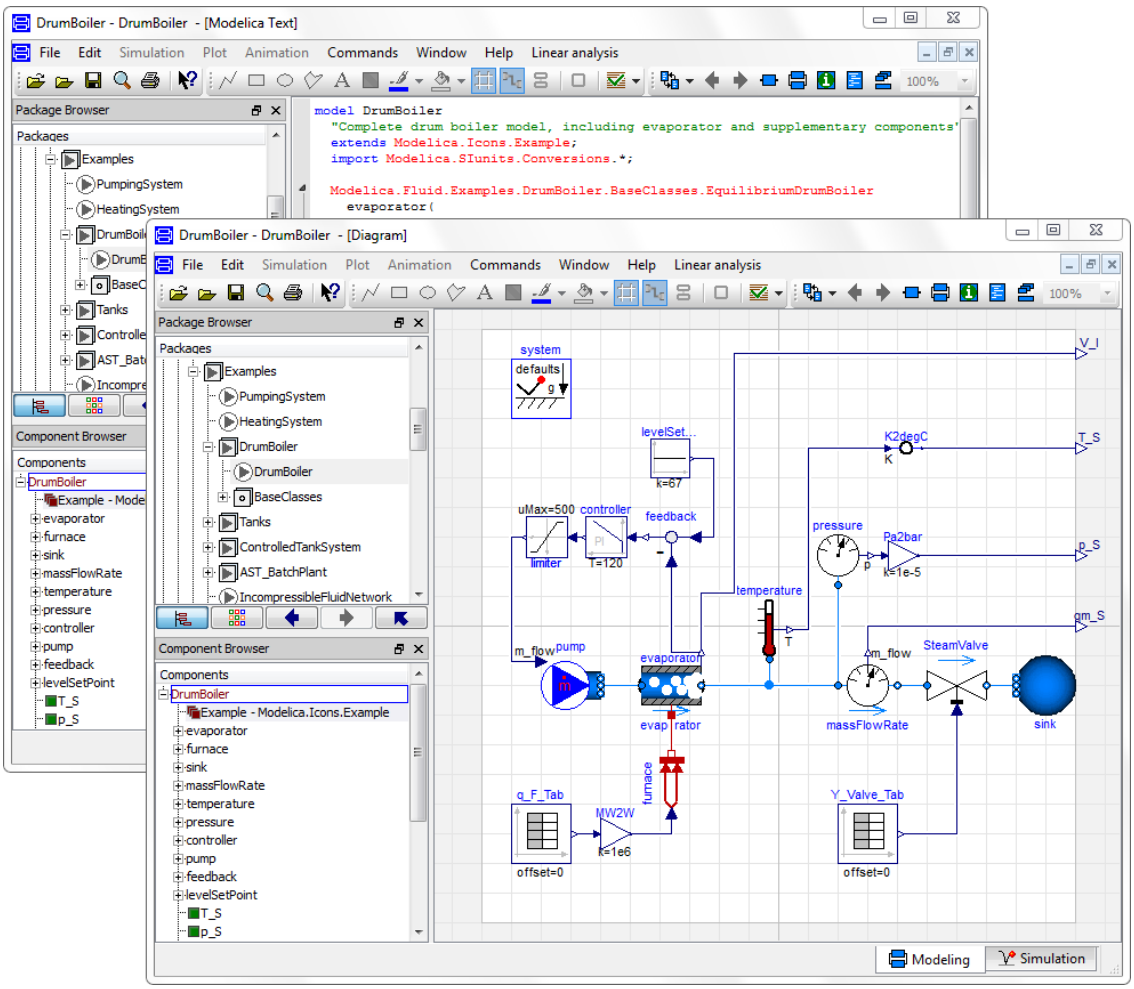

Figure 2.2: Overview the user interface in Dymola for a model of a drum boiler. The front image shows the block-based interface and the rear image shows the text-based interface.

The interface specifies how a model is exported as a combination of XML files and compiled C-code. The XML files provide information about the model (inputs, outputs, states etc.) while the C-code provides the model dynamics. FMI consists of two different editions, FMI for Model Exchange (FMI-ME) and FMI for Co-Simulation (FMI-CS). The difference between the two is that ME only provides the model while CS also includes a solver (see Figure 2.3).

A model exported with FMI is called a Functional Mock-up Unit (FMU). A FMU is a zipped file (with file extension .fmu) containing all of the model files. At the time of this report, the latest version of FMI was version 2.0 (released 2014-07-25) [7].

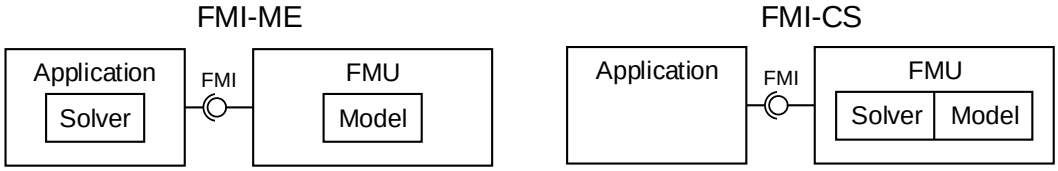

Figure 2.3: FMI interface for Model Exchange (left) and Co-Simulation (right).

2.2 State Estimation

In many control problems information about the states of a system is often required. All system states can rarely be measured directly without noise or disturbances. Therefore, it is often necessary to use some kind of state estimation technique to reconstruct the values of the states using observations (measurements) of the inputs and outputs combined with a model of the system.

2.2.1 Observability

For state estimation, the concept of *observability* is important. Observability describes if it is theoretically possible to calculate the system states from observations of the outputs. A system is said to be observable if for any combination of states and inputs it is possible to determine the state of the system using information of the output [8, 9].

A linear system on standard state space form

$$\dot{x} = Ax + Bu \tag{2.1}$$
$$y = Cx + Du \tag{2.2}$$

is said to be observable if and only if the observability matrix

$$\mathcal{O} = \begin{pmatrix} C & CA & CA^2 & \dots & CA^{n-1} \end{pmatrix}^T \tag{2.3}$$

has full rank, $rank(\mathcal{O}) = n$ where n is the number of states.

If a system is not observable, it might still be detectable. Detectability is a weaker property than observability. A system is said to be detectable if and only if all of its unobservable modes are asymptotically stable [8, 9].

2.2.2 Kalman Filter

The Kalman filter, or Linear Quadratic Estimator (LQE), is a popular algorithm for estimating the state of a system from noisy and uncertain data. The Kalman filter is a *recursive* estimator which can be formulated in two steps: *prediction* and *correction* [10].

Consider the following discrete time system on state space form

$$x_{k+1} = Ax_k + Bu_k + w_k \tag{2.4}$$
$$y_k = Cx_k + v_k \tag{2.5}$$

where the subscript k denotes the time step, w_k is the process noise and v_k the observation noise. Both noises are assumed to be additive zero mean Gaussian noise with covariances Q and R respectively ($w_k \sim \mathcal{N}(0, Q)$, $v_k \sim \mathcal{N}(0, R)$).

2.2.2.1 Initialization

For the first iteration ($t = 0$), the Kalman filter requires an initial guess of the state ($\hat{x}$) and a corresponding error covariance matrix (P):

$$\hat{x}_{0|0} = x_0 \tag{2.6}$$
$$P_{0|0} = P_0 \tag{2.7}$$

2.2.2.2 Prediction step

The prediction step uses the last state estimate ($\hat{x}_{k-1|k-1}$), the input signal (u_k) and the system model to predict values for the current state estimate ($\hat{x}_{k|k-1}$). The change of the error covariance matrix is also calculated according to the model and the given process noise:

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Bu_k \tag{2.8}$$
$$P_{k|k-1} = AP_{k-1|k-1}A^T + Q \tag{2.9}$$

2.2.2.3 Correction step

During the correction step, the filter uses the error covariances and system model to compute the *Kalman gain*:

$$K_k = P_{k|k-1}C^T(CP_{k|k-1}C^T + R)^{-1} \tag{2.10}$$

The Kalman gain is then used together with the observations ($\tilde{y}_k$) to correct the state estimate and error covariance matrix:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(\tilde{y}_k - C\hat{x}_{k|k-1}) \tag{2.11}$$
$$P_{k|k} = (I - K_kC)P_{k|k-1} \tag{2.12}$$

2.2.3 Extended Kalman Filter (EKF)

The basic Kalman filter requires the system model to be linear. Since many systems are described using nonlinear equations, several approaches to modify the Kalman filter for use on nonlinear systems have been developed. Commonly used in the industry is the extended Kalman filter (EKF). The EKF is based on the same equations as the basic Kalman filter (as it is just an extension). The nonlinearity of the system is handled by linearizing the system around the last estimate for every iteration [10].

Consider the nonlinear system

$$x_{k+1} = f(x_k, u_k) + w_k \tag{2.13}$$
$$y_k = h(x_k) + v_k \tag{2.14}$$

where w_k and v_k denotes the same type of noise as in Equation (2.4) and (2.5).

2.2.3.1 Initialization

The initialization is done in the same way as for the basic Kalman filter:

$$\hat{x}_{0|0} = x_0 \tag{2.15}$$
$$P_{0|0} = P_0 \tag{2.16}$$

2.2.3.2 Prediction step

The difference in the prediction step compared to the basic Kalman filter is that the EKF uses the nonlinear system model for calculating the state estimate and a linearized model when calculating the error covariance matrix:

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_{k-1}) \tag{2.17}$$
$$P_{k|k-1} = F_{k-1} P_{k-1|k-1} F_{k-1}^T + Q \tag{2.18}$$

where

$$F_{k-1} = \left. \frac{\partial f}{\partial x} \right|_{\hat{x}_{k-1|k-1}, u_{k-1}} \tag{2.19}$$

2.2.3.3 Correction step

In the correction step, the linearized model is used in the calculation of the Kalman gain. Apart from that, the correction step uses the same changes as for the prediction step. The nonlinear model is used to update the state estimates and the linearized model is used when calculating the Kalman gain and updating the error covariance matrix:

$$K_k = P_{k|k-1} {H_k}^T (H_k P_{k|k-1} {H_k}^T + R)^{-1} \tag{2.20}$$

where

$$H_k = \left. \frac{\partial h}{\partial x} \right|_{\hat{x}_{k|k-1}} \tag{2.21}$$

The Kalman gain is then used together with the observations ($\tilde{y}_k$) to correct the state and covariance estimates:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(\tilde{y}_k - h(x_k)) \tag{2.22}$$
$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{2.23}$$

A Schematic view of the recursive EKF algorithm is presented in Figure 2.4.

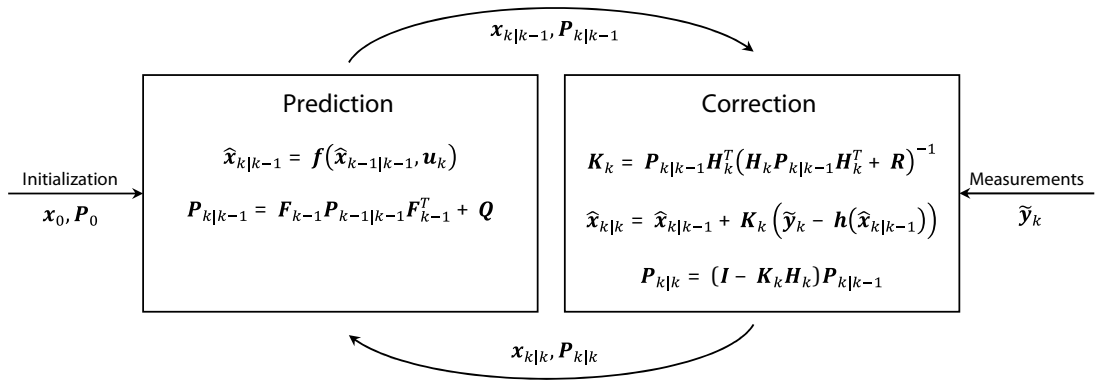

Figure 2.4: Schematic view of the EKF algorithm.

3 | Case Study: Plant Description

This chapter provides a description of the paper mill model used as a replacement for a real plant in the case study. The first part of the chapter gives a general introduction to how the plant process typically works in reality. The second part of the chapter covers how the plant is modeled in Modelica using Dymola.

3.1 Plant Process Overview

The plant in the case study is an integrated paper mill. The specific method used for making pulp is called thermomechanical pulping (TMP). Paper made from this kind of pulp is often used for newspapers and other printing purposes.

The process starts with debarking of wood logs. When the logs have been debarked they are turned into chips in a chipper. The wooden chips are then preheated and softened using steam, and then fed into a series of refiners. The refiners use large amounts of electricity to grind the chips between two large rotating steel discs. Grinding is done under pressure and at high temperatures (hence the name thermomechanical) forcing the cellulose fibers in the chips to separate. In addition to the pulp, due to the large energy consumption, large amounts of steam is generated by the refiners. After the refining, depending on the use, the pulp could be sent to bleaching where chemicals is used to increase the whiteness of the pulp. After this, the pulp is ready for the paper machine.

Paper machines are complex systems. The main part of a paper machine can be divided into four distinct sections: the forming section, the press section, the drying section and the calender section. The pulp enters the machine in the forming section where the pulp is formed into a wet paper web. The paper web then passes the press section where as much water as possible is removed by running the paper through roll presses. The paper then enters the drying section where water is evaporated using steam heated cylinders. The last major step in the paper machine is the calender section where the paper passes between cylinders which apply pressure to the paper. This makes the surface of the paper smooth and increases its glossiness [11].

In addition to the mentioned systems, a paper mill features several supplementary systems such as a steam network, hot water production systems and different energy recovery systems. A schematic view of the main plant process is presented in Figure 3.1.

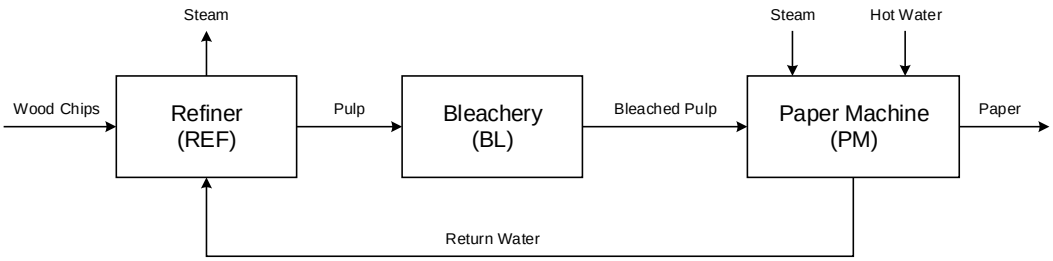

Figure 3.1: Schematic view of the main pulp to paper line.

3.2 Plant Model

The plant model is modeled in Modelica using Dymola. The plant model is to a large extent based on a previous project at Solvina [12], but with several changes to make the model better suit the requirements for this work.

The model describes the plant-wide behavior of one pulp to paper line (a real plant typically have several lines) with some of the related supplementary systems. Due to the wide model focus, only the most interesting and important aspects of the dynamics are modeled. Most of the components in the model originates from the Modelica Standard Library or an in-house Modelica library called SteamPower from Solvina. An overview of the implemented model in Dymola is presented in Figure 3.2. Some of the different model components are described in greater detail below.

3.2.1 Pulp Medium

To model the thermodynamic state of the pulp, a simple medium model is used. The medium is based on a template from the Modelica Standard Library. The thermodynamical properties of the pulp is determined from pressure, temperature and the mass fractions between water and fibers. All properties of the pulp is selected to be the same as for the medium *IF97 water* from the Modelica Standard Library except the enthalpy (h_p), which is calculated using

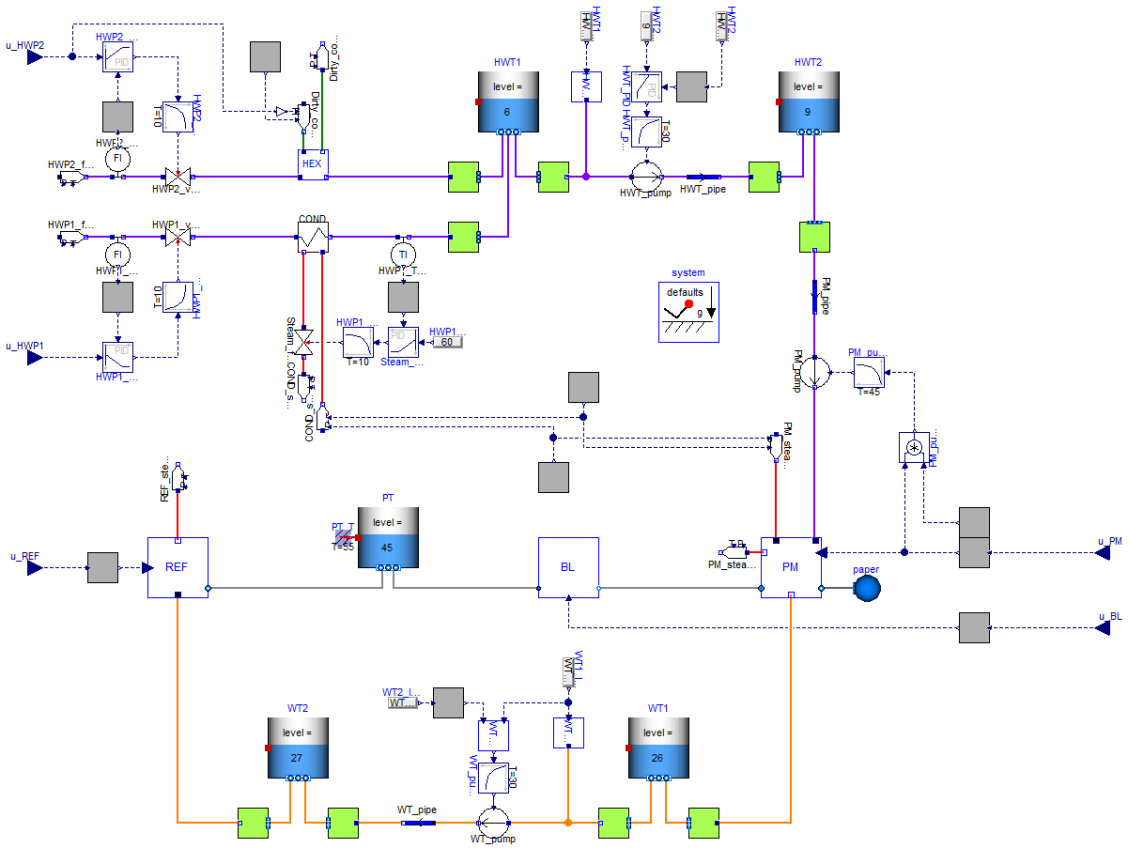

Figure 3.2: Overview of the plant model. The gray lines represent the pulp line, the orange lines the return water system, the purple lines the hot water production, the dark green lines dirty condensate and the red lines connections to the steam network. Grey boxes add disturbances and noise. Green boxes are adapters for connecting components from different Modelica libraries.

$$h_p = w_w h_w + w_f c_{p,f} T \tag{3.1}$$

where w_w is the mass fraction of water, h_w the enthalpy of the water, w_f the mass fraction of fiber, $c_{p,f}$ the specific heat capacity of the fibers and T the temperature of the pulp.

3.2.2 Refiner

The refiner (REF) is modeled using basic energy and mass balances. The input streams are wood chips and dilution water while the output streams are pulp and steam. The wood chips have a moisture content of 50 % and a temperature of 100 °C whereas the outgoing pulp have a consistency of 4 %. It is assumed that all dry content in the chips is turned into pulp. When the refiner is running at full

capacity it produces 225 kg/s of pulp. The pulp leaving the refiner is stored in a pulp tank (PT) at a constant temperature of 55 °C.

3.2.3 Bleachery

The bleachery (BL) model considers only its input-output behavior. The bleachery is comprised of two tanks (BL.T1 and BL.T2) and two pumps. Pulp is fed into BL.T1 where it stays for 1 h. Then the pulp is pumped into BL.T2 where it can be used by the paper machine. The pulp in the bleachery is stored at a constant temperature of 55 °C.

3.2.4 Paper Machine

The paper machine (PM) is, as the refiner, modeled using mass and energy balances. The model consists of two parts; a wet part (forming and pressing), and a dry part (drying and calendering). The wet part has pulp and hot water as input streams. Output streams are pulp (with a consistency of 50 %) and return water. The input streams for the dry part are the pulp from the wet part and steam. Output streams are condensate and finished paper, which contains 6 % water. When running at full capacity, the PM produces 7 kg/s of paper and consumes 175 kg/s of pulp.

3.2.5 Return Water System

The return water system recovers waste water from the paper machine and reuses it in the refiner process. The system includes two storage tanks (WT1 and WT2). Level control in the tanks is a simple on-off system which tries to keep the tank levels at certain values. To model the heat loss in the tank to the environment a heat transfer coefficient of 15 $W/(m^2K)$ is used.

3.2.6 Hot Water System

The PM requires hot water for its production. The hot water is produced from two different sources. The first, and main, production line (HWP1) uses steam to heat cold (5 °C) fresh water to 60 °C. The second production line (HWP2) uses leftover dirty condensate (92 °C) from another, not modeled, part of the plant to heat cold fresh water. Since the condensate originates from another part of the plant, the amount of hot water from HWP2 cannot be directly controlled by the operators. Hot water from HWP2 is merely additional hot water from an excess energy source.

The produced hot water is stored in a tank (HWT1). From HWT1 hot water is feed to a smaller tank (HWT2) closer to the PM. A control loop, using a PI-controller (HWT_PID), exists to keep the level in HWT2 constant at 9 meters. Heat loss from HWT1 and HWT2 are modeled in the same way as for the return water tanks.

3.2.7 Steam Network

The steam network is not included in the plant model. For all subsystems in the plant connected to the steam network, simple boundary components are used with a supply steam at 3.5 bar and 160 °C.

3.3 Operating Cycle

To be able to compare results from different simulations, a standard operating cycle (SOC) for the plant have to be established. Since no information about the operating cycle of a real plant is available, the SOC is chosen such that the resources of the plant are kept close to constant over a 12 h period, i.e. all of the tank levels should be roughly the same after 12 h of operation.

The central part of the plant is the PM. The objective is to have it running as much as possible and let the other parts of the plant run as much as required to satisfy the requirements of the PM. According to literature, the effective production time for a paper machine is $70 - 85\,\%$ of total time [13]. Usually, breakdowns or other stops in production are the cause of the paper machine not being in production. The frequency and length of these production stops are unknown so the PM is chosen to operate for 4 h and 15 min followed by a 45 min break in production (which gives 85 % of production time). The operating cycle for the other parts of the plant are chosen to balance the operating cycle of the PM. The resulting SOC for the plant can be seen in Figure 3.3.

3.4 Disturbance and Noise Modeling

To make the plant model behave more like a real system, disturbance and noise are added to the model. For this purpose, a specific component has been designed in Modelica. The component offers the ability to add white Gaussian noise with a given mean and standard deviation at a given sampling time. The component also offers the ability to add a sinusoid to mimic slow varying disturbances. The noise and disturbance component can be seen as grey blocks in Figure 3.2.

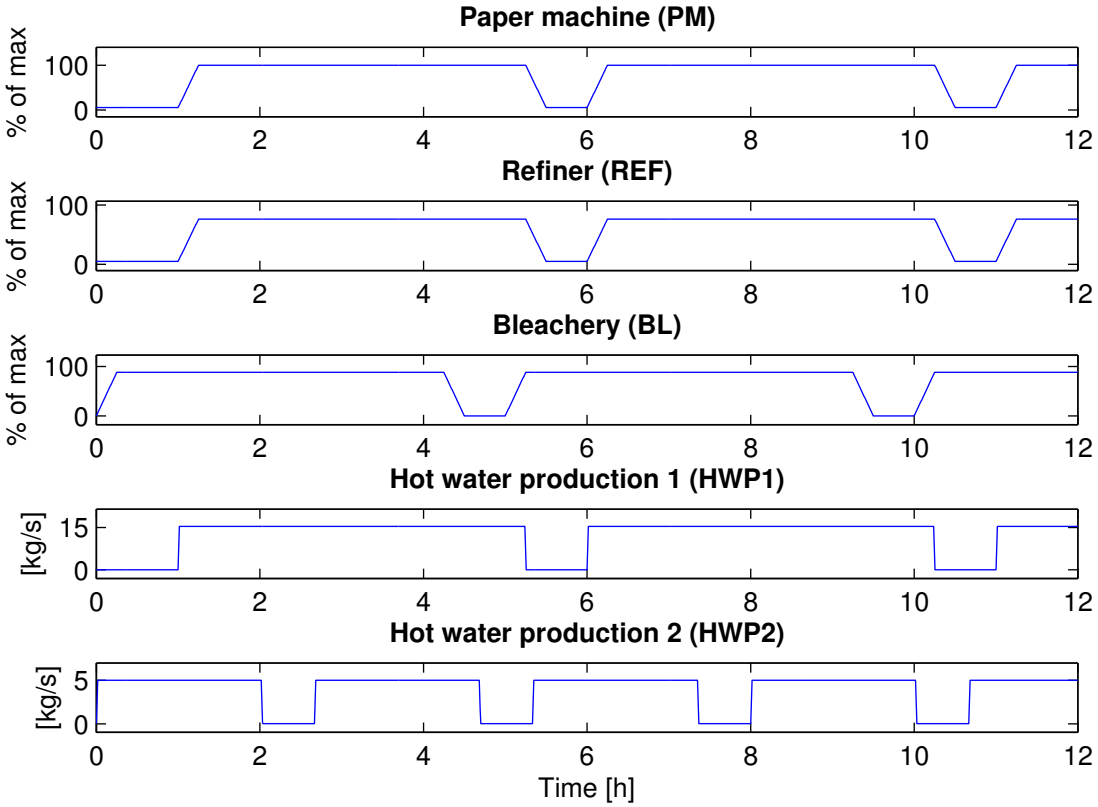

Figure 3.3: The Standard operating cycle (SOC) chosen for the plant in the case study. Note how REF, BL and HWP1 balance the needs for the PM. HWP2 operates independently from the others and is only used when dirty condensate is available.

Noise and disturbances are added according to Table 3.1.

- **Sensors and outputs**
  Measurement noise is added to the sensors in the system, both to the ones used internally by the controllers of the plant and to the ones used for outputs from the plant. The different noises are adapted depending on the type of sensor.

  Tank level sensors are assumed to be accurate with a low level of white Gaussian noise. Instead, the sensor readings are mainly disturbed by sloshing in the tanks [14]. The effect of sloshing is modeled by adding a sinusoidal signal.

  Temperature sensors are assumed to have a moderate level of white Gaussian noise. The readings from temperature sensors are also affected by sensor drift and changes in temperature due to stratification in the tanks [14]. This slow varying disturbances are also modeled by adding a sinusoidal signal.

  Flow sensors are assumed to have a moderate level of white Gaussian noise.

Table 3.1: Levels for the noise and disturbances added to the plant model. For the white Gaussian noise, μ is the mean value, $\%RSD$ the relative standard deviation compared to the variables nominal levels and t_s the sampling time. For the sinusoidal disturbance, A is the amplitude of the sinusoid relative to the variables nominal level and T the period time.

	White Gaussian			**Sinusoid**	
	μ	$\%RSD$	t_s	A	T
Sensors					
Tank level	0	0.02 – 0.05 %	5 s	0.2 – 1 %	10 s
Temperature	0	0.5 – 0.8 %	5 s	0.5 – 0.8 %	4 – 12 h
Flow	0	0.5 – 1 %	5 s	0.1 – 1 %	6 – 12 h
Control signals	0	0.5 – 1 %	60 s		
Steam network					
Pressure	0	1.5 %	240 s		
Temperature	0	0.2 %	120 s		

- **Control signals**
  A moderate amount of white Gaussian noise is added to the control signals to introduce small disturbances in the plant.
- **Steam system**
  A moderate amount of white Gaussian noise is added for the pressure and temperature of the boundary components that represents the steam network.

The effect of the added noise and disturbances are presented for some variables in Figure 3.4.

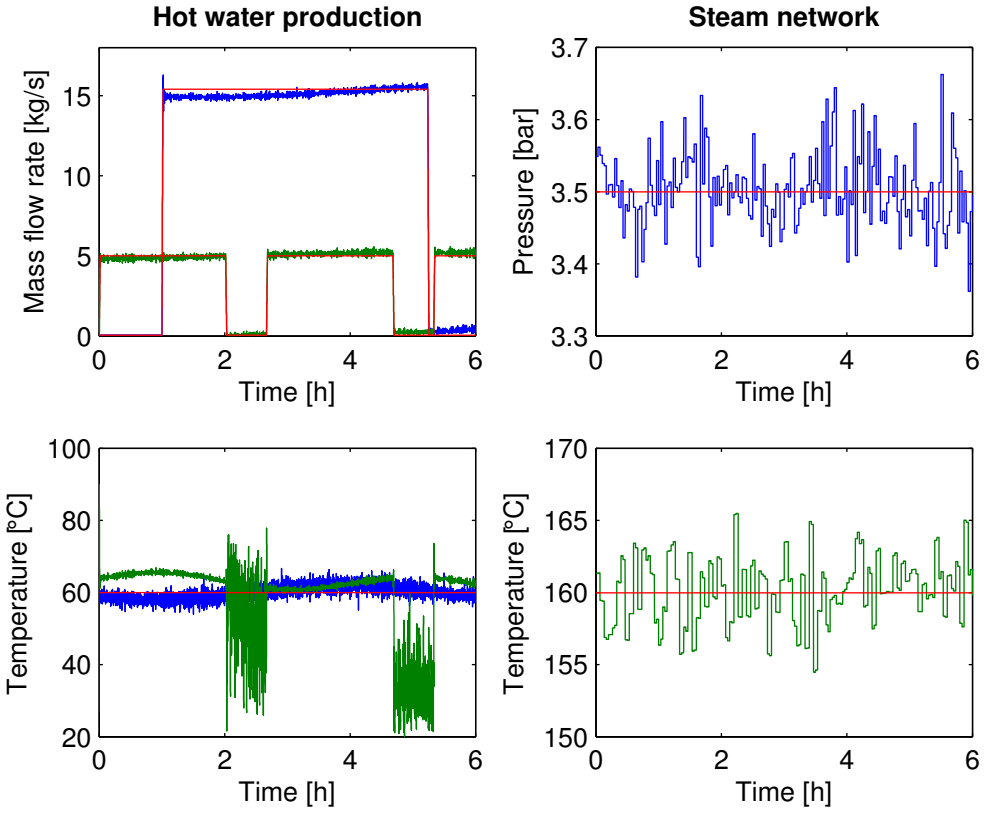

Figure 3.4: Examples of noisy signals in the plant. The left column shows how the hot water production is affected by noise. The blue lines show HWP1, the green lines HWP2 and the red lines indicate the desired value. The large variation in the temperature of HWP2 during short periods is due to a very small flow through the heat exchanger, causing numerical errors in the model which result in large fluctuations in the output temperature. Since the flow is so small, this has no impact on the temperature in HWT1. The right column shows how the pressure and temperature of the steam system vary. Here, the red lines also indicate the desired value.

4 | Proposed DSS Concept

This chapter presents the proposed DSS concept that was developed in this thesis. The chapter covers both the method developed for providing predictions and some general guidelines for different design choices. First, a short overview of the DSS concept is given, where the main structure of the DSS is presented and the function of the different parts are briefly explained. This is then followed by sections, where each part is explained in more detail.

4.1 DSS Overview

Decision Support Systems (DSS) are a broad area covering a large variety of tools related to assisting humans in decision making situations. Each DSS often has its own implementation and the design is often strongly related to how the DSS is to be used. The DSS considered here is supposed to be used online, parallel to a real plant process to provide operators with predictions of the future behavior of the plant.

The aim was to design the DSS concept to fulfill to the following:

- **Applicable for a variety of process systems**
  The DSS structure must allow development of DSS for a variety of process systems, not just the plant considered in the case study presented in Chapter 3.
- **Modular design**
  The structure should, as much as possible, be modular. Each module could then be adapted and upgraded separately from the others. The modular solution will allow the modules to be reused for other systems.
- **Stand-alone implementation**
  The DSS concept should be designed so it can be delivered as a stand-alone application without the need of licenses for expensive third party software.
- **High performance**
  The DSS concept should be able to deliver fast predictions in order to minimize the waiting time for the operators. As an example, the DSS should be able to deliver a 4 h prediction within 1 min when running on a standard personal computer.

An overview of the proposed structure for the DSS concept is presented in Figure 4.1 and is explained in detail below.

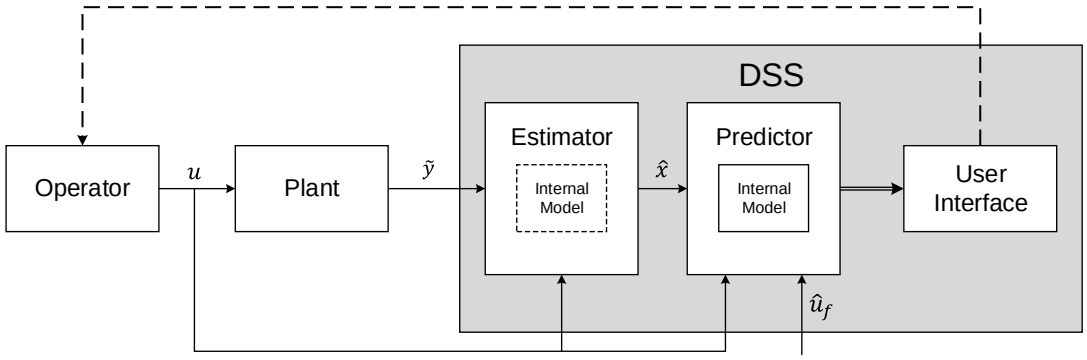

Figure 4.1: Overview of the structure of the DSS concept. u is the operator controlled inputs (control signals), $\tilde{y}$ the measured outputs from plant, $\hat{x}$ the estimated state used as the initial starting point for the predictor, and $\hat{u}_f$ the estimated future operator control actions (future change in production etc.).

The central part of the DSS is the predictor. The predictor provides information about the expected future behavior of the plant. To do this the predictor initiates an internal model, representing the plant, to a starting point ($\hat{x}$) which corresponds to the current state of the plant. The predictor then simulates the model for the amount of time that is requested. Besides values for the starting point, the predictor also requires values for the future control signals ($\hat{u}_f$) to be used in the simulation. More detailed information concerning the predictor is presented in Section 4.4.

The starting point for the predictor is to be calculated using measurement data ($\tilde{y}$) from the plant. Since measurements are often noisy and can be conflicting it can be hard to calculate a good starting point. To handle this problem, an estimator module must be designed. Depending on the available plant measurements and the design of the predictor model, the estimator might require an internal model of the plant. The estimator module is explained further in Section 4.3.

The internal models which are used to describe the plant are important for the DSS. The end-performance of the DSS is directly linked to how well the models are able to reflect the dynamics of the plant. When describing large process systems it is important to use a powerful modeling solution which allows for modeling of complex systems. Further information about the development of the models is given in Section 4.2.

4.1.1 DSS Usage

The proposed structure of the DSS concept offers a number of different potential usage applications. Two important ones are considered in this thesis.

The first (normal operation) is to use the DSS to monitor the current trend of the plant by automatically running the predictor at fixed intervals using the latest values from the estimator as the starting point. This will allow the operators to get a quick update of where the plant is currently heading.

The second, and maybe the most important use of the DSS, is to allow the operators to run the predictor multiple times using the same starting point. By letting the operators control the future control signals ($\hat{u}_f$) it is possible to run simulations where the effects of changes in control can be studied and optimized before applying them to the real system.

4.2 Internal Models

The DSS concept was chosen to use Modelica models developed in Dymola. Modelica offers a powerful modeling solution and was selected for a number of reasons:

- **The Modelica model structure**
  Modelica models are developed using components representing different physical parts of the plant (pumps, pipes, tanks etc.) connected in hierarchical structures. As Modelica models several physical properties simultaneously (for example both the pressure and the temperature changes) modeling will be fast and efficient for many aspects of the plant. The structure also allows models to be easily changed and expanded over time if the plant is redesigned.
- **Large number of available libraries**
  Thanks to the extensive use of Modelica within the industry, many existing libraries are available to support modeling of thermo-fluid systems [15]. This minimizes the need for developing large parts of the models from scratch.
- **Previous knowledge at Solvina**
  Another important reason for choosing Modelica was that Solvina has previous knowledge of modeling in Modelica/Dymola. Solvina also has developed a number of in-house Modelica libraries for modeling of process and power systems.

Modelica models have traditionally been used for offline simulation and design purposes. To use Modelica models online introduces a different set of requirements regarding numerical robustness and computational speed. Knowledge gathered when developing Modelica models for use within nonlinear model predictive control (NMPC) has proven to be useful since similar principles apply for the development of models for the DSS. Several such projects exist in literature [16, 17].

4.2.1 Model Scope

The models of the DSS must describe the dynamics of large process systems at a plant-wide basis and at the same time be fast to simulate. In order to keep the model efficient and the complexity to a minimum, the model designer has to decide which simplifications that should be made while still keeping the interesting dynamics. This requires a good insight into the process that is to be modeled. A simple rule of thumb is that, if a specific phenomenon is an important factor in the real system, it should also have an important role in the model [18].

Useful techniques to reduce model complexity include:

- **Simplify complex components into mass and energy balances**
  When constructing models for large energy systems an effective technique is to reduce complex components into simple mass and energy balance equations. Even though this technique has large advantages, it must be used with some care. For example, if reducing a large component like a whole paper machine to a single set of balance equations it might give an efficient model with a correct input-output behavior, but using such an approach will also spoil the ability to study any of the internal dynamics, which could be an important aspect for the operators using the DSS. Therefore, an important task of the model designer is to decide which components could be simplified and for which components it is important to keep the internal dynamic structure.

- **Utilize operational conditions**
  The complexity of a model can often be reduced by considering the conditions for which the plant is operated [18]. A potential area for such simplifications are control loops. For example, if it is known that a certain temperature in the plant is strictly controlled through a control loop, it might be useful to model the temperature as constant. When set to constant, no state will represent the temperature in the translated model.

- **Precompute properties**
  If the model contains complex relations, for example advanced thermodynamic properties, a way to simplify the model is to precompute the relations and form lookup tables. Using this technique, caution has to be taken not to introduce non-smooth derivatives which can cause numerical instabilities [18].

4.2.2 Selection of States

An important factor to consider during the development of the models is the selection of states. Normally, the variables chosen as states in Modelica models are selected automatically during the translation process. Though the automatic selection usually works well there could be some advantages selecting the states manually when developing models for online use. Manually selecting states that closely relate to variables measurable in the plant will reduce the complexity of the estimation problem (this is explained in Section 4.3). Selecting states manually also allows the model designer to select a less nonlinear representation [6, p. 347] and to avoid implicit equations [18].

Modelica offers the ability to manually affect the selection of states using the attribute `stateSelect`.

4.2.3 Formulation of Initial Problem

The DSS concept requires the models to be initiated using specific values for the states. The model designer has to make sure that the initial problem is formulated such that the given values for the states are not changed when the solving the initial problem [4], [5, pp. 121-131].

Initial values for variables are set in Modelica using the attribute `start`. By also adding the attribute `fixed=true` this means that this is a required value and not just a value to use as the initial guess [5].

4.2.4 Selection of Solver (Integration Algorithm)

When Dymola translates a Modelica model for simulation it is linked with a numerical integration algorithm, a solver. Dymola includes a number of solvers for the model designer to choose from. The choice of solver will affect the speed and the robustness of the DSS and the best choice depends on a combination of model properties. For most cases it may well be that the model work well with one solver but is unsolvable using another.

Key aspects to consider when selecting the solver are:

- **Fixed vs variable step algorithms**
  With fixed step solvers, the integration step size remain the same during the whole simulation. Variable step solvers adapts the step size to fit the specified error tolerances. Variable step solvers are usually preferred since they are usually more efficient. Fixed step algorithms are usually only used if the simulation has to be performed in real time or with a discrete time designed controller. For models used in the DSS, variable step solvers are to be used [5].
- **Single vs multiple step algorithms**
  The solver can either use a single or multiple step approach. Single step algorithms are designed to start fresh for every integration step and do not use information from any previous integration point. On the other hand, multiple step solvers use information from previous points. Since the solvers have to be restarted after every discrete event, and the cost of restarting single step algorithms are lower, multiple step solvers are only recommended for models with few events while single step solvers are to be used otherwise [5].
- **Algorithms for stiff models**
  If the model includes a combination of fast and slow dynamics it may cause the algorithms to become numerically unstable if the step size is not very small. If a model has this behavior it is said to be *stiff*. For these kind of systems, special solvers should be selected [5].

4.3 Estimator

A process plant features a large amount of sensors measuring different variables of the process. The main task of the estimator is to constantly evaluate the sensor measurements to find a good starting point for the predictor simulations.

To do this the estimator have the following tasks:

- **Sensor fusion**
  Use combinations of sensor data to form "better" approximations of the states than if single sensors were used.
- **Mapping measurements to model states**
  The model is a simplification of the real world. The estimator has to know how the measurements from the real plant relates to the states in the predictor model and map them accordingly.

The complexity of designing the estimator depends on how the available plant measurements relate to the states of the predictor model.

In the most simple case all information required for estimating the states of the predictor model can be directly measured or trivially calculated from plant measurements. Thus, at least one sensor measurement is present for every state of the predictor model. In this case the task of designing the estimator becomes easy, all that is required is to handle the noise in the sensors by proper filtering of the signals.

Unfortunately, the more common and more complicated case is that some of the states in the predictor model cannot easily be directly related to the measurements from the plant. One scenario is that a measurement might be missing due to the particular variable being difficult to measure, or that no sensor is present. Another scenario might be that the predictor model is such a simplification of the real world that a variable corresponding to the particular state does not exist in the same form in the plant and therefore cannot be measured directly.

The more complicated case requires some kind of model-based estimation technique to be used within the estimator. Process systems often include nonlinear dynamics so the estimator should be designed with this in mind. The field of nonlinear state estimation is broad and many different choices exist. A requirement is that the chosen method has to be compatible with Modelica models. Modelica models are translated into code for simulation with no easy access to the equations. Therefore, the estimator has to be designed so that the model can be interfaced by sequentially calling simulations of the model. Popular solutions include nonlinear Kalman filters, such as the extended Kalman filter (EKF) and the unscented Kalman filter (UKF), or Moving Horizon Estimation (MHE) for example. The best choice also depends on the model dynamics and the available computing power. While MHE is a very powerful solution, it requires a nonlinear optimization problem to be solved and therefore might not be suitable as the first solution to try for the implementation of the estimator. A better "first-try" solution is to use EKF or UKF. The EKF is just an extension of the basic Kalman filter where the system is linearized at every time step. This linearization can be done efficiently for Modelica/Dymola models (see Section 4.5), but it also means that the EKF might have poor performance for systems with highly nonlinear dynamics.

4.3.1 Constraints on State Values

When estimating the states one has to make sure that no invalid state values are generated. Since the models represents physical systems, the states cannot assume all possible values, for example a temperature cannot be below absolute zero. Trying to initiate the prediction model with invalid state values will fail and cause the DSS to crash. To prevent this, constraints related to the physical restrictions have to be implemented.

In their standard form the Kalman filters do not handle constraints. This is often one of the most common arguments to use a MHE based approach, but several versions of constrained nonlinear Kalman filters (CEKF and CUKF) have been presented [19, 20].

4.3.2 Estimation of More Than States

The above section only covers state estimation since it is an absolute necessity for the implementation of the DSS. The option also exists to estimate other factors, for example model parameters. If the plant features some type of time-varying parameter which need to be included in the model, introducing parameter estimation might be a way to improve the accuracy of the DSS. Extending the estimation to also cover parameters using Modelica based solutions have been shown [21, 22].

4.4 Predictor

The predictor is the part of the DSS that supplies the predicted future behavior of the plant. The predictor is basically a simulation of an internal plant model for a predetermined amount of time. The most important aspect of the predictor is to accurately describe the behavior of the plant for as long time as possible.

Generally three factors determine the accuracy of the predictions:

- **Error due to model inaccuracy**
  The first factor to affect the accuracy of the prediction is how well the predictor model represents the dynamics of the plant. Simply put, the better the model, the longer predictions can be made.
- **Error in starting point from the estimator**
  The second factor is errors in the starting point supplied by the estimator. This error will affect the prediction right from the start of the simulation.
- **Error in estimate of future control inputs**
  The third factor is the accuracy of the estimated future control inputs. Future control inputs are difficult to estimate since they are controlled by human operators and affected by unscheduled changes in production and plant breakdowns. Despite this, estimates of future control inputs are a necessity if predictions should be valid for longer than just until the next change in control.

In a largely automated process system, when the control signals directly controlled by the operators are few, it might be possible to get a relatively good estimate of future control signals from production schedules and schedules for planned maintenance work etc. For a system where many control signals are controlled by the operators it might prove impossible to provide good estimates for the future control signals and then a successful implementation of a DSS will be very challenging.

4.4.1 Validity of Predictions

During production, the operators might need time to study the predictions in order to make good control decisions. Therefore, it can be useful to monitor for how long a prediction is valid and may be used to support decision making. A situation might occur when an operator makes a change in control based on an old, invalid prediction and this can have large negative consequences for the control of the plant. To avoid such kind of situations, it may be a good idea to include an automatic system, a *validator*, in the predictor for monitoring the validity of the predictions.

The method used by the validator can be very simple. Once a prediction has been made, the validator compares the values of the prediction to the latest state estimate and control signals. If the estimate or control signals start to deviate to much from the values in the prediction, the operator is warned that the prediction might be invalid.

Another use of the validator can be to support automatic fault detection. If the control actions remain in line with what was given for the prediction, but the state estimates start to deviate, a malfunction has occurred in the plant (leaking pipe, stop in pump etc.) and the operator is warned.

4.5 Implementation

The DSS is going to be used during active production, which means requirements on stability and availability. This makes the implementation of the system important. As already mentioned in Section 4.1, the DSS should be designed so that it can be delivered as a stand-alone program needing no additional software to be installed. For example, Dymola should not be required by a completed DSS.

The DSS can be implemented in a number of different ways. Since it was chosen to use Modelica models developed in Dymola the best implementation practice is connected to the possible ways of interfacing with these models.

Dymola offers a number of possibilities for exporting a model, which can then be simulated outside Dymola:

- **As Dymosim**
  Dymosim is a stand-alone executable program (.exe) generated by Dymola. Interfacing with Dymosim is normally done using a simple file interface. An input file with the start values for the model variables is generated. Dymosim reads the file, performs the simulation and returns the result in an output file. The files can either be ASCII formatted text files (.txt) or version 4 Matlab (.mat) files. Using Dymosim with the file interface is the method used within Dymolas internal simulation mode. An overview of Dymosim's file interface is presented in Figure 4.2.

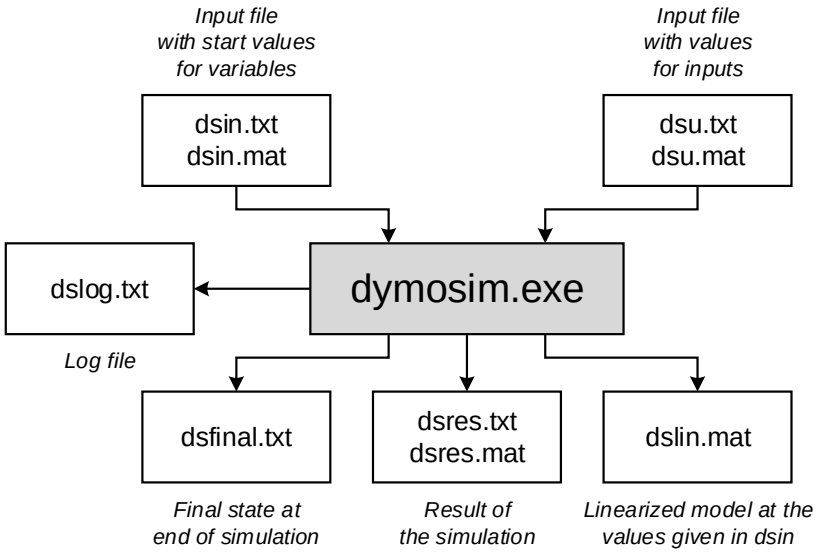

Figure 4.2: Overview of Dymosim's file interface.

In addition to the file interface, Dymosim can be compiled to include a DDE or OPC server. DDE and OPC are standardized forms of communication between applications. By running Dymosim with a DDE/OPC server one can communicate and interact with the simulation. Values for variables can be read and written while the simulation is controlled (start/stop/pause).

A third option for Dymosim is to compile it as a Dynamic Link Library (.dll). This means the model can easily be integrated into custom software.

- **As Simulink S-functions**
  Dymola also offers the ability to export models for use within Simulink. Dymola is most often required to be installed on the same computer as Simulink in order to simulate the models. Despite this, options exist to first compile the model in Dymola to S-functions which can then be used in Simulink without connection to Dymola [6, p. 236].

- **As FMUs using FMI**
  Another approach for export of Dymola models, that have arisen during the last years, is the use of the Functional Mockup Interface (FMI). Two versions of FMI exist, 1.0 and 2.0. FMI 2.0 is the best choice since it includes more functions and is more robust [23]. By exporting a model for Co-Simulation (FMI-CS) the model can be used for simulation in any environment supporting the FMI standard. Support for FMI is currently being implemented in a variety of tools. A number of libraries have also been developed for interfacing with FMUs directly for different programming languages. Libraries currently exists for C [24], Python [25] (part of the larger package Jmodelica.org [26]) and Java [27]. A benefit of basing the DSS concept around models exported with FMI-CS is that models can be developed using any tool supporting FMI-CS export, not only Dymola.

- **As C code**
  As a final possibility Dymola offers the option to export the models directly as source code (C-code). The model can then be linked with binary libraries containing the solver code. This option is only recommended for advanced users.

Depending on the design of the estimator, linearizations of the models can be required (for example when using an EKF). While linearization can be made numerically, two of the export techniques above offer analytic linearizations of the models, Dymosim and FMI 2.0. When using FMI 2.0, model linearizations are not offered directly but can be derived using the `fmi2GetDirectionalDerivative` function [7, p. 27]. Generally, FMI has become a popular approach when implementing nonlinear estimators with Modelica models [23, 22, 28].

5 | Case Study: Simulation Results

In this chapter different aspects of the DSS concept is evaluated using the case study presented in Chapter 3.

5.1 Internal Model Development

Three versions of internal models for the paper mill were developed to describe different levels of model accuracy. The developed models were designed such that they can be used both within the estimator and the predictor. All models were designed to use the solver `Dassl` which is a variable step solver using a multiple step method that can handle stiff dynamics.

Model 1 was developed with the aim to simplify the model as much as possible while still retaining a near perfect representation of the plant. To minimize the number of states, the two hot water production lines (HWP1 and HWP2) were simplified to boundary components supplying hot water at a constant temperature. The main pulp line was modeled without the fiber content, using only a standard IF97 water medium, thus eliminating states related to mass fractions of the pulp. The resulting model has 12 states which were manually selected to be the temperature and level for the tanks and states for the integral part of the PI-controller in the hot water system. The final states are presented in Table 5.1. An overview of Model 1, as seen in Dymola, is presented in Figure 5.1.

Model 2 is based on Model 1, adding some parametric errors to reflect when one is not able to derive the correct parameters from the plant. Three different parametric errors were introduced. The first was that the pumps between the WT and HWT tanks in the model operate with 110 % capacity of the pumps in the plant. This could represent lowered performance of the pumps in the plant due to wear while modeling the pumps using their original specifications. The second error was giving the PI-controller (HWT_PID) different values for its parameters compared to the ones in the plant. The controller gain (K) was given a slightly higher value and the time constant for the integrator part (T_i) was given a lower value. The third error introduced was that the heat transfer between the tanks and the environment in the internal model is 150% of that in the plant. This is to reflect when the loss of heat in the tanks is overestimated in the internal model.

Table 5.1: State variables for the models

State	Modelica variable	
x1	PT.level	Level in pulp tank (PT)
x2	BL.T1.level	Level in bleachery tank 1 (BL.T1)
x3	BL.T2.level	Level in bleachery tank 2 (BL.T2)
x4	WT1.level	Level in return water tank 1 (WT1)
x5	WT1.medium.T	Temperature in return water tank 1 (WT1)
x6	WT2.level	Level in return water tank 2 (WT2)
x7	WT2.medium.T	Temperature in return water tank 2 (WT2)
x8	HWT1.level	Level in hot water tank 1 (HWT1)
x9	HWT1.medium.T	Temperature in hot water tank 1 (HWT1)
x10	HWT2.level	Level in hot water tank 2 (HWT2)
x11	HWT2.medium.T	Temperature in hot water tank 2 (HWT2)
x12	HWT_PID.I	Integrator value of PI-controller (HWT_PID)

Model 3 is based on Model 2, introducing a major modeling error by completely removing the second hot water production line (HWP2). Though it might seem unlikely that such a severe modeling error could be made, it is important to remember that real plants are a lot more complex than the plant used in this case study and include hundreds of streams. This makes it possible for streams to be overlooked during modeling.

5.1.1 Validation

To evaluate the developed models they were simulated and compared to the plant using the standard operating cycle (SOC) over a 12 h period. The results for the 12 states for the three different models are presented in Figure 5.2.

Model 1 manages to resemble the plant well for most of the states. A difference can be seen for the temperatures in WT1 and WT2. The difference originates from the replacement of the pulp medium. Without the fiber content in the pulp, the heat capacity of the pulp is different and the energy balance in the paper machine changes, which affects the water temperature.

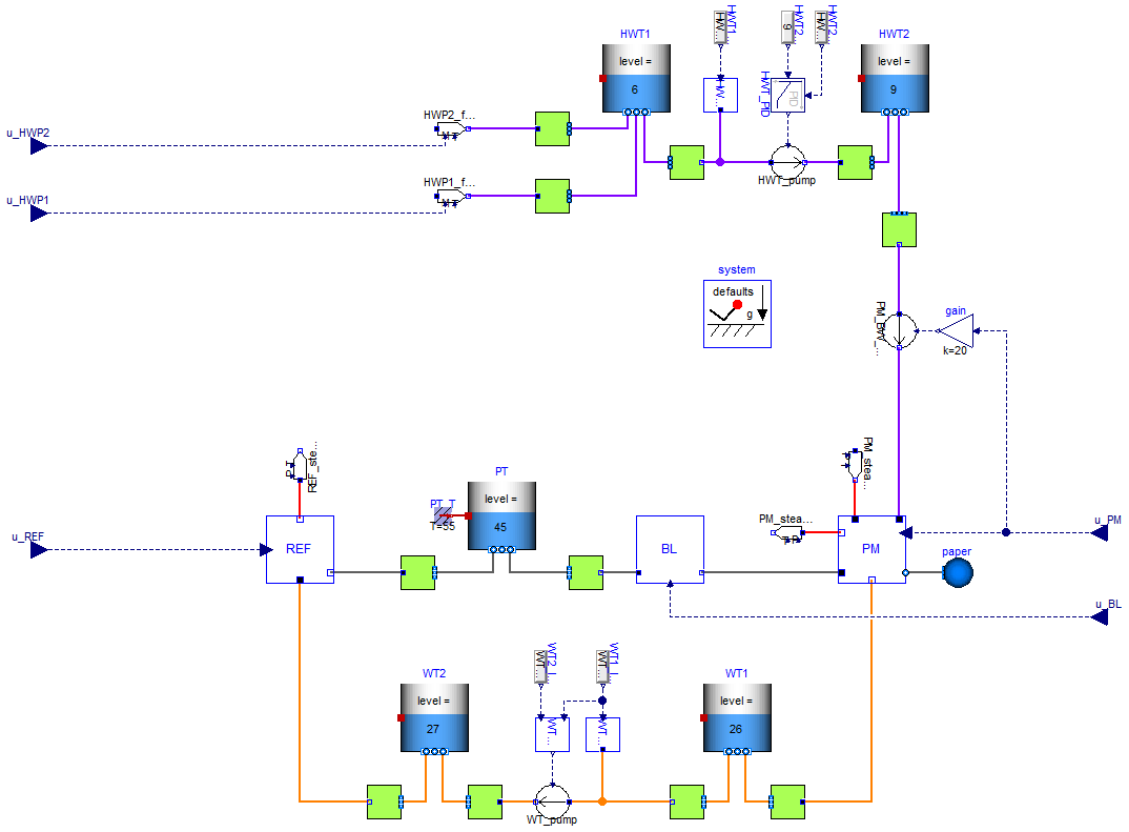

Figure 5.1: Overview of Model 1. The grey lines represents the pulp line, the orange lines the return water system, the purple lines the hot water production and the red lines connections to the steam network. Green boxes are adapters for connecting components from different Modelica libraries. Simplifications in the model can be seen by comparing it to Figure 3.2 which shows the plant model.

The parametric errors for the pump capacity introduced in Model 2 and 3 allow a faster water transfer between the tanks in the models compared to that in the plant. For the level of HWT2 no difference can be seen since it is controlled by the PI-controller. For the levels in WT1 and WT2 the result is different. The "zig-zag" pattern of the tank levels is changed but it is still centered around the same mean. The error in heat transfer is noticeable by the temperatures in WT and HWT tanks being lower for Model 2 and 3.

The effects of removing one of the hot water streams in Model 3 can be seen clearly from the tank level in HWT1. Since HWP2 supplies hot water at a slightly higher temperature than HWP1 the removal also has a negative effect on the temperatures in the tanks.

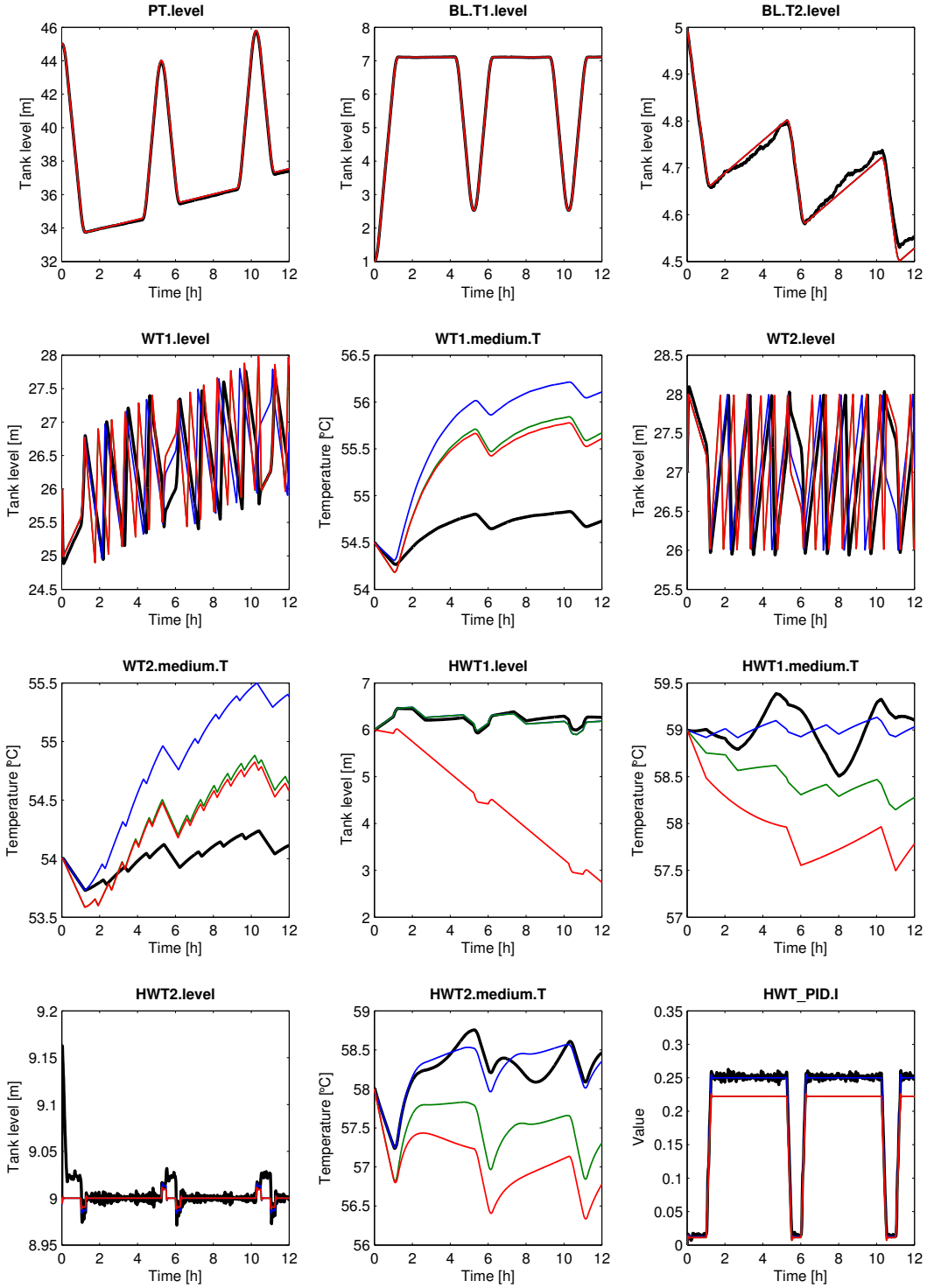

Figure 5.2: Comparison between plant and the three developed internal models. The thick black lines corresponds to the true value from the plant, the blue lines to Model 1, the green lines to Model 2 and the red lines to Model 3. Note that all lines are not visible in some graphs since they are covered by other lines.

5.2 Estimator

The objective of the estimator is to provide estimates for the states in the predictor model for the current state of the plant. Two different implementations of the estimator were studied for the paper mill. One case is when all states are measurable and another when state estimation is required.

5.2.1 All States Measurable

The studied system offers a particular good case for filtering since the system dynamics are slow compared to the measuring noise. A simple moving average filter was designed in MATLAB using the function `filter` to average the signals over 150 s using a 3 s sampling time. The result of the filtering is presented for two of the states in Figure 5.3.

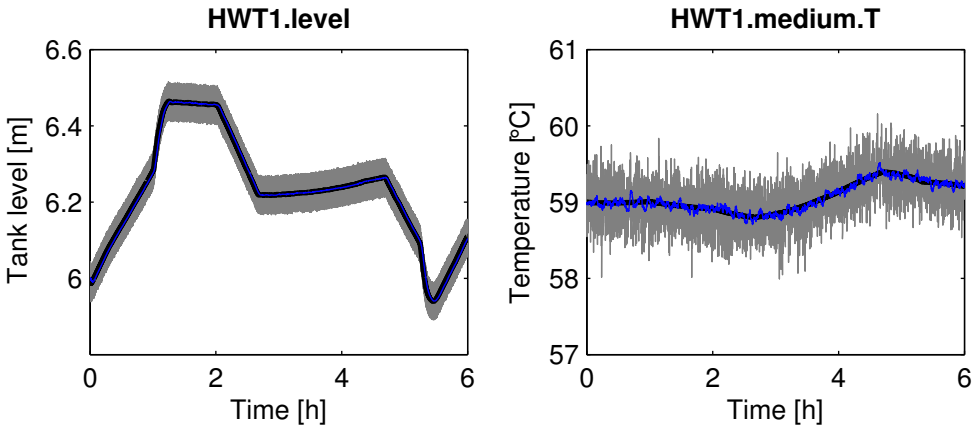

Figure 5.3: Example of filtered measurement signals for the state related to HWT1. The thick black line corresponds to the true value from the plant, the grey lines are the measurement signals and the blue lines are the signals after filtering.

As can be seen in Figure 5.3, the moving average filter manages well to filter out the measurement noise. As long as the measurements from the sensors does not have too large offset errors using simple filtering in the estimator is a viable approach.

5.2.2 State Estimation Required

As mentioned, direct measurements can in reality not be expected for all states and state estimation probably has to be performed. In order to demonstrate the use of state estimation using Modelica models, an EKF was implemented using each of the three developed models. 13 variables (see Table 5.2) were chosen to be measured as plant outputs such that the system becomes observable.

Table 5.2: Output signals from the plant.

Output signal	Measurement
y1	Level in pulp tank (PT)
y2	Level in bleachery tank 1 (BL.T1)
y3	Level in bleachery tank 2 (BL.T2)
y4	Level in return water tank 1 (WT1)
y5	Temperature in return water tank 1 (WT1)
y6	Level in return water tank 2 (WT2)
y7	Temperature in top of return water tank 2 (WT2)
y8	Temperature in bottom of return water tank 2 (WT2)
y9	Level in hot water tank 1 (HWT1)
y10	Temperature in hot water tank 1 (HWT1)
y11	Mass flow between hot water tanks (HWT1 and HWT2)
y12	Temperature of hot water entering paper machine (PM)
y13	Integrator value of PI-controller (HWT_PID)

Output measurements from the plant were generated using the standard operating cycle for 6 h with a sampling time of 30 s. The sampled outputs were imported into MATLAB where the EKF algorithm was implemented. Model 1-3 were exported as Dymosim and interfaced with the EKF using the file interface.

For implementing the EKF in MATLAB four special functions were created:

- `build_dsin`
  Function used to generate the dsin.mat file with start values for variables to the Dymosim simulation.
- `build_dsu`
  Function used to generate the dsu.mat file with values for the inputs to the Dymosim simulation.
- `check_inputs`
  The implemented EKF is a standard EKF not considering constraints. In order to avoid invalid state estimates `check_inputs` was created. The function makes sure that the state estimates ($\hat{x}$) and inputs (u) are within their allowed range.
- `ekf`
  Function that performs one iteration of the EKF.

The algorithm is described below:

1. Initiate state estimate $\hat{x}_{k-1}$ and error covariance matrix P_{k-1} to the initial guesses.
2. Get the input u_k and measured outputs $\tilde{y}_k$ from the plant.
3. Check $\hat{x}_{k-1}$ and u_k using `check_inputs`.
4. Create `dsu.mat` and `dsin.mat` with values for the state estimate and the input.
5. Linearize the model using the command `Dymosim -l` and load the matrices in `dslin.mat` into MATLAB.
6. Since the matrices are for continuous time behavior, discretize them using $F = e^{A_{cont}T_s}$ and $H = C$ (A_{cont} is the continuous time system matrix and T_s is the sampling time).
7. Check that the matrices are observable. If not, print out a warning.
8. Simulate the model using `Dymosim -s` for one sample and load `dsres.mat` into MATLAB. From the result, extract the values for $\hat{x}_{k|k-1}$ and y_k.
9. Use F to calculate $P_{k|k-1} = FP_{k-1|k-1}F^T + Q$.
10. Use H to calculate the Kalman gain $K_k = P_{k|k-1}H^T(HP_{k|k-1}H^T + R)^{-1}$.
11. Calculate the new state estimate $\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(\tilde{y}_k - y_k)$.
12. Calculate the new error covariance matrix $P_{k|k} = (I - K_kH)P_{k|k-1}$.
13. Iterate from step 2.

The MATLAB code for the implementation is presented in Appendix A.

The most important tuning parameters for the EKF are the estimates for the covariances Q and R. To receive the best performance it is important to get a good balance between the two. In practice, Q and R are often chosen as diagonal matrices, i.e. the noises are assumed to be uncorrelated. The values along the diagonal are then the variances for the different variables. Simply put, the different values in the matrices become weights on how much each variable (states and measurements) can be "trusted". A low value (low variance) corresponds to small noise for the particular variable meaning it should be trusted more by the Kalman filter. Values for R are normally the easiest to estimate since it relates to the measurement noise. Choosing values for Q is more difficult since it relates to model errors and process disturbances, which are difficult to assess.

For the EKF in this case study the choice of Q and R were initially chosen as:

$$\begin{aligned} Q = diag([&0.001\ 0.001\ 0.001\ 0.001 \\ &0.001\ 0.001\ 0.001\ 0.001 \\ &0.001\ 0.001\ 0.001\ 1000]); \end{aligned} \tag{5.1}$$

$$\begin{aligned} R = diag([&0.16\ 0.001\ 0.001\ 0.001 \\ &0.05\ 0.001\ 0.05\ 0.05 \\ &0.16\ 0.05\ 0.16\ 0.05\ 0.001]); \end{aligned} \tag{5.2}$$

The simulation result for the EKF is presented in Figure 5.4.

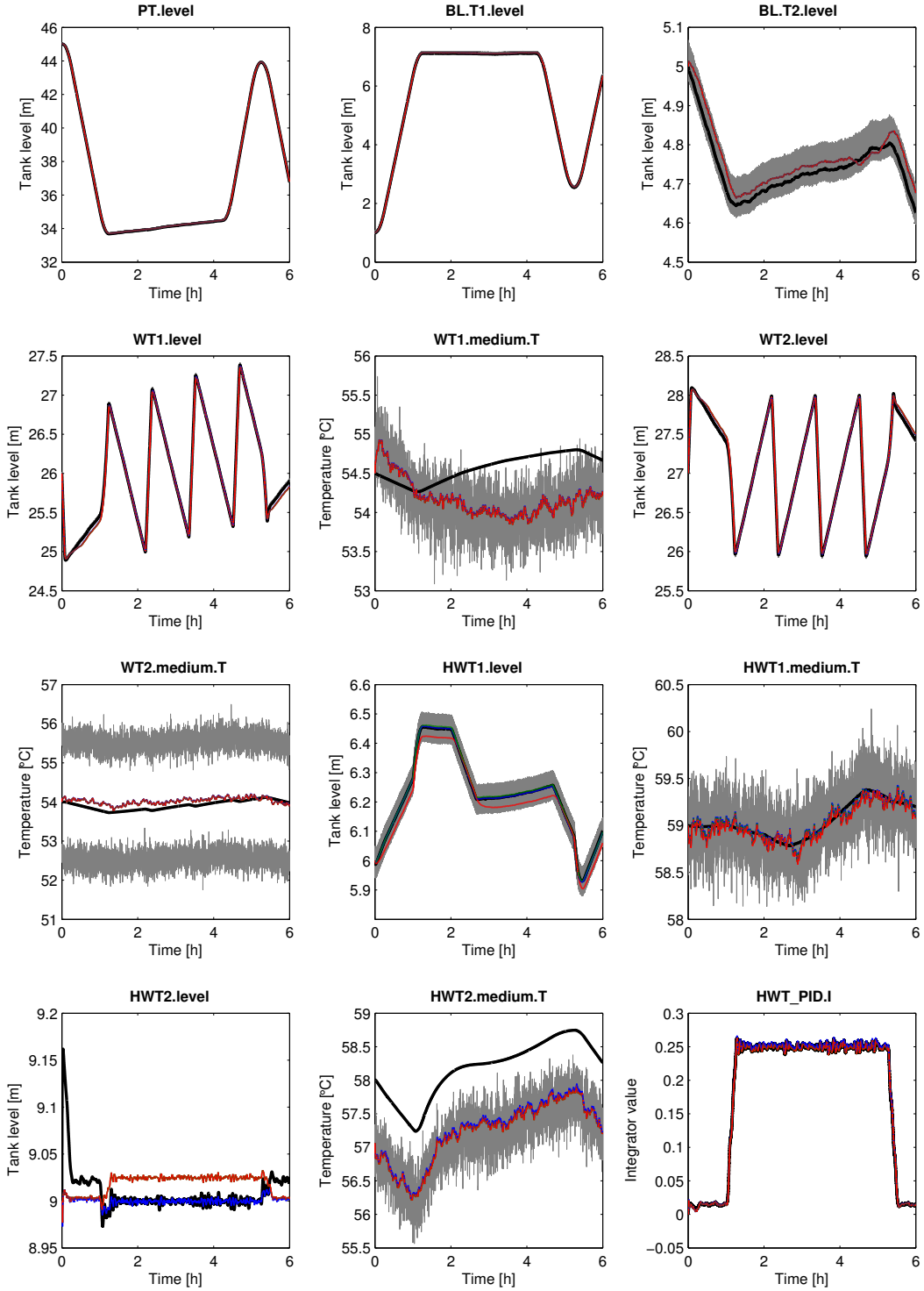

Figure 5.4: Result of the state estimation using the EKF. The thick black lines are the true values of the states from the plant, the grey lines sensor measurements, the blue lines the state estimates when Model 1 is used, the green lines when Model 2 is used and the red lines when Model 3 is used. Note that all lines are not visible in some graphs since they are covered by other lines.

As seen in Figure 5.4, the EKF manages to estimate most of the states well. The result also shows that the EKF perform almost the same for all models despite their different errors. The reason for this is because the state estimates are constantly corrected using measurement data. However, some errors can be seen in the estimation for the temperature in HWT2 (HWT2.medium.T), the temperature in WT1 (WT1.medium.T) and the level in HWT2 (HWT2.level) when using Model 2 and 3.

The estimation error for HWT2.medium.T is due to no direct measurement being available. To limit the number of states in the models, no pipe dynamics were modeled between HWT2 and the PM. Thus, the EKF sees the measurement of the inflow temperature to the PM as a direct measurement of the temperature in HWT2 and corrects the state estimates towards it. This can be seen in Figure 5.4 for HWT2.medium T where the temperature for the water entering the PM (y12) is plotted in gray. To avoid this type of errors, care has to be taken not to put too much trust in measurements for which model dynamics are missing. By adjusting Q and R so that less trust is put on y12, the estimate in HWT2.medium.T can be improved. This is presented in the left plot of Figure 5.5.

The estimation error in WT1.medium.T is related to sensor drift in measurement y5 (compare the grey measurement line with the black line showing the true value). Trying to use the same technique for correcting, as above (less trust in y5) will force the EKF to increase the trust on the model. Since there are modeling errors for the temperatures in WT1 (pulp modeled as water) this will impair the estimate as can be seen in the right plot of Figure 5.5. This type of error with sensor drift is best avoided by using information from more sensors in the plant.

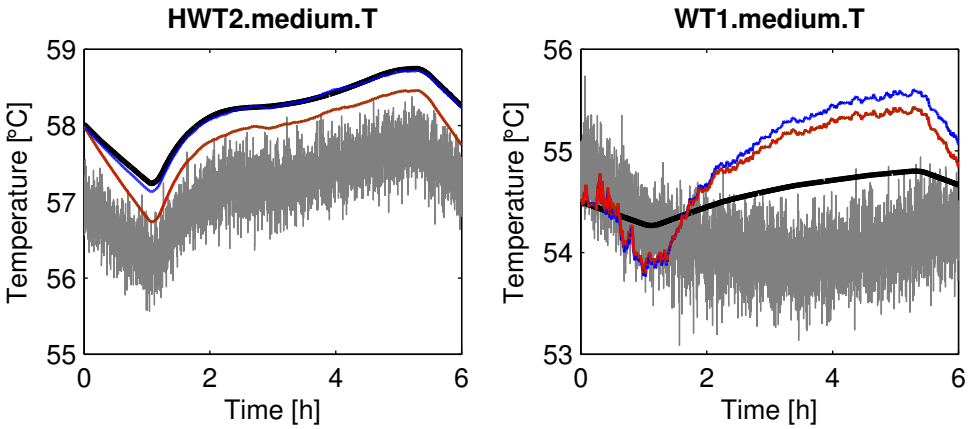

Figure 5.5: EKF compensation for missing model dynamics or sensor drift. The thick black lines are the true value of the states from the plant, the grey lines sensor measurements, the blue lines the state estimates when Model 1 is used, the green lines when Model 2 is used and the red lines when Model 3 is used.

The error in the estimate for HWT2.level for Model 2 and 3 is related to the level in HWT2 not being directly measured for the EKF. Since the flow between the tanks are measured ($y11$), it can be used together with the controller parameters (K, T_i) and the integrator value from HWT_PID ($y13$) to calculate the level in the tank. Since HWT_PID is a part of the plant control system, $y13$ is not a sensor measurement and was assumed to be noise free. The values of Q and R were therefore adapted to highly trust $y13$ (compare the high last value in Equation (5.1) to the low value in Equation (5.2)), giving the result presented in Figure 5.4. By adjusting the values of Q and R other results can be achieved. An example of this is seen in Figure 5.6

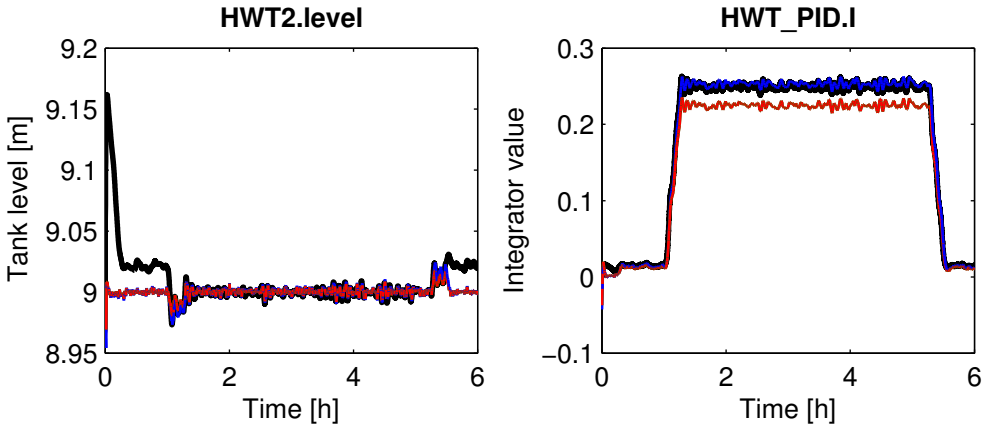

Figure 5.6: Result when trying to compensate for wrong controller parameters. The thick black line is the true value of the states from the plant, the blue lines the state estimates when Model 1 is used, the green lines when Model 2 is used and the red lines when Model 3 is used. As can be seen on the left, when the weights of Q and R are changed the estimation of HWT2.level becomes correct for all models. On the right, it can be seen how the change of weights instead moves the estimate error for Model 2 and 3 to the integrator state of HWT_PID.

5.2.3 Computation Time

The estimator must be able to perform its calculations between the sampling points. In the simple case where all states are measurable this will not be a problem since filtering can be done fast and efficiently. For the more complex case, where the EKF has been designed, the used sampling time was 30 s and this is therefore also the maximum allowed time for the EKF calculations.

With the implemented solution, one iteration of the EKF takes about 0.4 s, leaving a large margin for use of more complex models, use of more advanced estimator techniques or a shorter sampling time. Note that the calculation time would probably also be shorter if a more efficient way to interact with the model simulations was used. With the current implementation, a large portion of the 0.4 s is spent on reading and writing to files.

5.3 Predictor

As mentioned in Chapter 4, three main factors determine the accuracy of the predictions. To study the influence of these factors simulations were performed for the predictor.

5.3.1 Error Due to Model Inaccuracy

The first factor to affect the accuracy of the prediction is how well the model describes the system. The effect of the three developed models on the prediction has indirectly been shown in Figure 5.2. Generally the modeling errors and their respective impact on the prediction can be divided into two different categories:

The first category are model errors which have a less severe impact on the prediction since they only affect how energy and material are distributed in the system. An example of this type of error is difference in pump capacity between two tanks. For this type of error the prediction can give the wrong value for the individual content in the two tanks, but the sum of the contents in the two tanks might still be correct.

The second category of model errors are more severe when energy or material is lost from the system. Examples of this type of errors are missing streams or higher/lower heat transfer coefficients. The impact of these kind of errors will increase with simulation time.

5.3.2 Error in Starting Point from the Estimator

The second factor to affect the accuracy of the prediction is errors in the starting point supplied by the estimator. To study the impact of an initial error in the starting point, simulations were performed using Model 1 where three different values were given for the initial values of the tank level and temperature in HWT1. The results are presented in Figure 5.7.

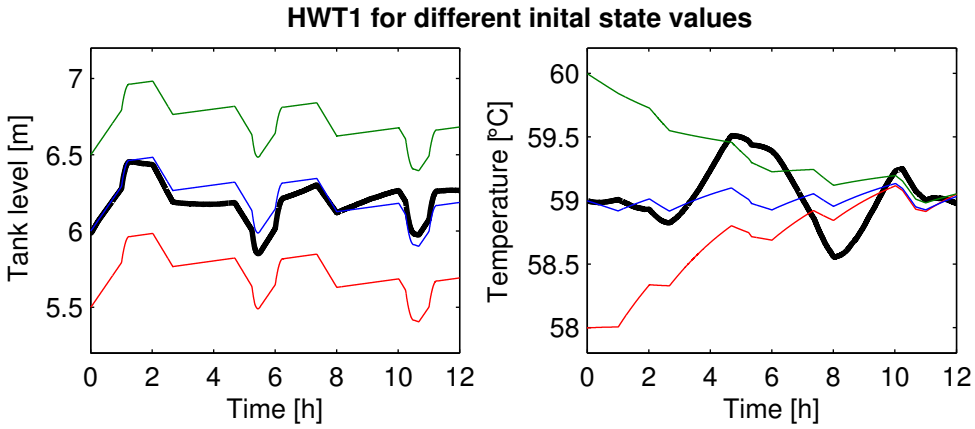

Figure 5.7: Simulations performed with Model 1 with different initial state values. The thick black lines represent the true value from the plant, the blue lines Model 1 with no initial error, the green lines Model 1 for a situation where the initial values for the states are too high and the red lines when the initial values are too low.

The supply to the tank is modeled correctly for flow and temperature. Therefore, the impact of the initial error in the tank level will remain constant for the whole prediction. The impact of an initial error in the temperature is different. In this case, the impact will decrease over time since new water is supplied and the tank temperature will converge to the temperature of the supplied water. Since the model has a correct supply water temperature, the tank temperature will converge towards the correct value.

5.3.3 Error in Estimate of Future Control Inputs

The third factor to affect the accuracy of the prediction is the estimates of future control inputs. As already mentioned in Chapter 4, the impact of this factor is very hard to determine since the decision to change the control inputs are made by human operators. Despite this, supply estimates of future inputs are necessary in order to make predictions longer than just to the next change in control. Simulations were made for two possible scenarios.

5.3.3.1 Delays in Operating Cycle

The first scenario covers unexpected delays in the operating cycle of the PM after a prediction was performed. Two different delays are possible, an extended production time or an extended stop time.

Simulations were made using the SOC with the two different delays. In the first case, the production of the PM was extended by 20 min at 1.5 h into the SOC. In the second case, the stop time of the PM was extended by 20 min at 30 min into the SOC. Results from the simulations are displayed for HWT1 in Figure 5.8.

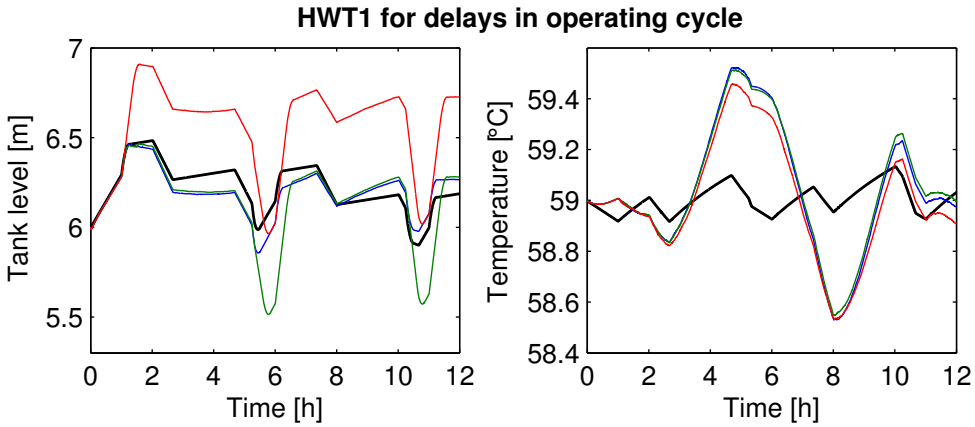

Figure 5.8: Impact on the accuracy of the prediction for HWT1 if the operating cycle is delayed. The black lines represent the prediction made using Model 1 according to the SOC, the blue lines the plant if the estimate of future control inputs was correct (i.e the estimates matched the SOC), the green lines a 20 min delay after 1.5 h and the red lines a 20 min delay after 30 min.

As can be seen, the impact of the delays is worst for the tank level. The reason is that the operating cycle used in the prediction has become out of sync with the plant. Depending on when the delay occur the impact is different. If the delay happens when the PM is running (extended running time, green lines), the cycle will get out of sync for the periods during when the PM should have been turned off. If the delay happens when the PM is stopped (extended stop time, red lines) the cycle will get out of sync during the period when the PM was supposed to run. For the temperature only a small difference can be seen, and compared to the modeling error, the impact of the delay is negligible.

5.3.3.2 Disturbances in Production Cycle

The second scenario considered is disturbances that cause unexpected stops in production. Simulations were performed when 5% and 10% of the scheduled production time are stops. The result is presented in Figure 5.9.

Again, the impact is worst for the tank level. The unexpected stops mean that the predicted water usage is much higher than in the plant, causing the tank levels to be underestimated. For the temperature, the result is similar to the scenario with delays where the change is so small that it can be neglected.

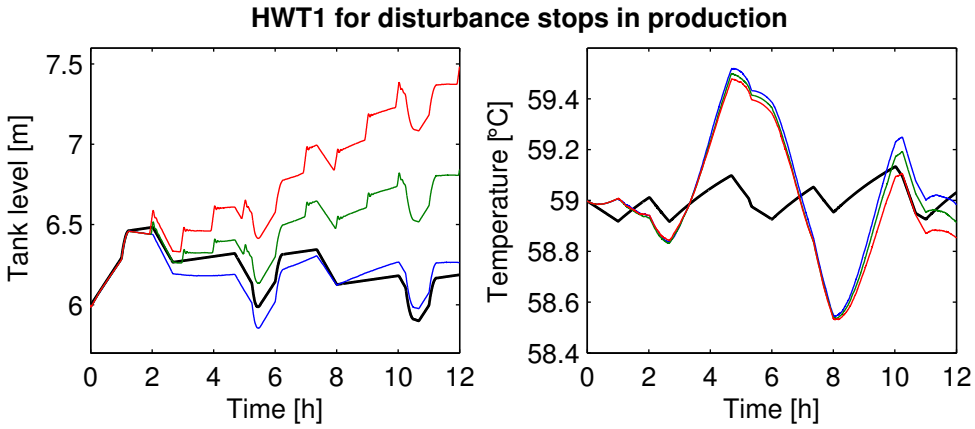

Figure 5.9: Impact on the accuracy of the prediction for disturbance stops in production. The black lines represent the prediction made using Model 1 according to the SOC, the blue lines the plant if the estimate of future control inputs were correct (i.e the estimates matched the SOC), the green lines the plant running according to the SOC but when 5% of each hour are unexpected stops and the red lines when 10% of each hour are unexpected stops.

5.3.4 Computation Time

As mentioned in Chapter 4, the DSS should be able to deliver a 4 h prediction within 1 min. To test the performance, the three models were simulated for 12 h. The simulation times were roughly the same for all the models within approximately 17 s. The models used in this case study are therefore well below the target, which leaves a large margin for more complex models to be used in the predictor.

6 | Discussion

In this chapter various aspects of the proposed DSS concept are discussed. Different previously presented issues are covered and some reflections on design choices and their consequences are made.

6.1 DSS Concept

The purpose of this thesis was to propose a concept for a decision support system (DSS). The proposed concept should be able to serve as a starting point for future development of DSS at Solvina. Development of a DSS is a large project and only a part of the required work has been covered in this thesis

The DSS concept is a combination of modules which have to work well together in order to provide good predictions for the operators. Since the modules are intertwined, the different design choices have to be balanced and compromises are necessary in order to achieve acceptable performance of the DSS.

6.2 Modelica in Internal Models

Important for the success of the DSS is to have good internal models. Since Solvina has previous knowledge of modeling in Dymola/Modelica the proposed concept was developed to use this type of models. The choice of using Modelica to construct the models offers a versatile and powerful modeling solution. The use of Modelica models in online applications is a relatively new area that has arisen in the last couple of years. The majority of the work presented in literature is related to nonlinear model predictive control (NMPC) [16, 17, 29]. Projects like this one, where Modelica models are used in DSS systems to provide operators with information, are not as common and references in literature are few [30].

When designing the models the most challenging task is to achieve a suitable model complexity. The models should be able to describe complex process systems on a plant-wide basis, but still remain relatively simple to meet the computational requirements set by their online use. The use of a simple model for the predictor will result in few states and a less complicated implementation of the estimator, but a simple model might not describe the behavior of the plant well enough, meaning

the prediction will only be accurate for a short period of time. On the other hand, using a too complex model for the predictor will result in a large number of states which will make the design of the estimator difficult. The models will also take much longer to simulate with an increased waiting time for the predictions.

The models developed for the case study used in this thesis are relatively simple. Most of the large components in the plant were simplified to a single set of mass and energy balance equations. It was decided to replace the pulp medium with regular water to reduce the states in the internal models. When examining the results, it is clear that this was not the optimal choice since it introduced one of the most significant error in Model 1. A better simplification would probably have been to keep the pulp media, but modify it to have a constant consistency. In this case, no states would be created for the mass fractions, but the media would have more pulp-like characteristics than plain water.

6.3 Design of the Estimator

When constructing a DSS with the proposed concept, the design of the estimator will most likely require the largest effort. The design of the estimator is closely related to the plant process and to the states of the model in the predictor. Several choices exist for the design. The best choice depends on the dynamics of the system.

In the case study two different designs for the estimator were presented; one where all states were directly measurable and one where state estimation using an EKF was implemented. For the studied system, the EKF manages rather well to estimate the states. However, if an EKF was implemented for a system with highly nonlinear dynamics the performance would likely be worse and other approaches such as the UKF or MHE should be considered.

The EKF was implemented for models of the whole plant. Therefore, in each time step the entire plant model had to be linearized and simulated. For small models with few states, such as the ones in the case study, this approach was not a problem. However, if an estimator should be constructed for a much larger model with hundreds of states it may prove difficult to include a model of the whole plant as one component. In this case, an option might be to separate the estimator into several sub-estimators that cover different states. The different state estimates could then be combined and used for initializing the predictor model which features all the states. Separation into sub-estimators would also simplify the tuning (selection of Q and R for the EKF). With smaller modules, the tuning parameters could be more easily selected and evaluated.

Important for the estimator is the interface used to communicate with the model. The estimator needs fast access to the model since it should be simulated repeatedly. In the case study, interfacing to the models was done using files (exported as Dymosim). This implementation is easy to implement, but the solution is slow and sensitive to faults. Therefore, this technique is only recommended for test purposes and not to be used in an active DSS. A better solution would be to use a more direct approach to interface with the models. A solution that appears promising for the future is to use FMI-CS 2.0. This gives a direct interface and the models can be developed using any tool supporting FMI. Disadvantages using a FMI based implementation are mostly related to it being a relatively new standard still under development (FMI 2.0 was released 2014-07-25). Native support for FMI is currently lacking in many common simulation tools (for example MATLAB/Simulink and LabVIEW). However, efforts are made to implement FMI support in future versions.

6.4 Accuracy of the Predictions

Even though the predictor is simple to design, the accuracy of the predictions are affected by a number of factors which can be difficult to influence. Three main factors have been mentioned. The two first factors, errors due to model inaccuracy and errors in initial state from the estimator, can likely be minimized by designing good models and using powerful estimator techniques.

Minimizing the effect of the third factor (errors in estimate of future control inputs) offers a totally different challenge. Estimating what the operators will do in the future can prove to be very difficult and depends upon the type of work tasks the operators perform within the actual plant process. Therefore, more information, experience and knowledge about how operators work in real plants have to be gathered and analyzed.

7 | Concluding Remarks

This chapters provides some concluding remarks about the work performed in this thesis, and gives suggestions for future development.

7.1 Conclusions

The work presented covers the design of a decision support system (DSS) concept to be used within the process industry. The DSS concept use simulations of internal models, in parallel to the real plant, to provide predictions to operators about the future behavior of the plant.

In the proposed concept, predictions are made by initiating a prediction model to a starting point representing the current state of the plant. The predictor model is then simulated, providing the predictions. The initial starting point is provided by an estimator module which continuously evaluate measurement data from the plant. To develop the models used within the DSS, Modelica was chosen for its powerful multi-domain modeling capabilities and its ability for simple reuse of model components.

To evaluate the proposed DSS concept, a case study was conducted where parts of the concept was evaluated for a fictitious thermomechanical paper mill. Three different models were developed to be used as internal models, each representing different levels of model accuracy. The estimator module was designed for two different cases; one simple, where all states were measurable, and one where state estimation was performed using an extended Kalman filter (EKF). Simulations were also performed to evaluate different factors affecting the accuracy of the predictions.

The results from the case study show that the proposed concept provides a possible solution though each implementation of a DSS has to be adapted for each specific process system. Nevertheless, the implementation technique and the knowledge gathered are reusable when developing DSS concepts for other systems.

Concluding, the development of a DSS is a large and challenging project and the presented DSS concept provides a far from finished solution. Further work have to be conducted in a number of areas before a DSS as the one proposed can be developed for use within active production. Despite this, the results gathered in this thesis show the development of a DSS to be promising.

7.2 Further Work

To further develop the DSS concept extended work have to be performed. The main areas of interest can be summarized into four points:

- **Development of online models for real process systems**
  Developing the models that are to be used in the DSS are a time consuming task which requires a good understanding of the system that should be modeled. In order to get a better understanding of Modelica models for online use, models have to be developed for real process systems where validation data is available. It is only then that it is possible to see what kind of problems might arise and how they are best handled.
- **Further research into estimators using Modelica models**
  A complex part of the DSS is the estimator. Since several different design options exist for the estimator further studies have to be performed to evaluate the performance of the different options. For this task, information gathered from nonlinear model predictive control (NMPC) using Modelica models can prove useful. Another thing to consider when designing estimators for large process systems is some kind of automated implementation technique to minimize the the manual work otherwise required.
- **Availability of future control inputs**
  As mentioned previously, an important factor for the accuracy of the predictions are the estimates of future control signals. At the moment, it is unknown how well these can be estimated. Therefore, further studies have to be performed on real process system to gain more knowledge.
- **User interface**
  While it is outside the scope for this thesis, development of a suitable user interface for the DSS has to be conducted which allows the operators to interact and work with the system.

Bibliography

[1] L. Ljung and T. Glad. *Modeling of Dynamic Systems*. Prentice Hall, 1994. ISBN: 9780135970973.

[2] *Modelica Association*. URL: `http://www.modelica.org`.

[3] *Modelica Language Specification 3.3*. Modelica Association. May 2012.

[4] S. E. Mattsson et al. "Initialization of Hybrid Differential-Algebraic Equations in Modelica 2". In: *Proceedings of the 2nd International Modelica Conference*. Oberpfaffenhofen, Germany, Mar. 18–19, 2002, pp. 9–15.

[5] *Dymola - User Manual Volume 1*. 15th ed. Dassault Systemes AB. Sept. 2013.

[6] *Dymola - User Manual Volume 2*. 15th ed. Dassault Systemes AB. Sept. 2013.

[7] *Functional Mock-up Interface for Model Exchange and Co-Simulation*. 2.0. Modelica Association. July 25, 2014. URL: `http://www.fmi-standard.org`.

[8] T. Glad and L. Ljung. *Control Theory: Multivariable and Nonlinear Methods*. CRC Press, 2000. ISBN: 9780748408788.

[9] K. J. Åström and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2008. ISBN: 9780691135762.

[10] S. Särkkä. *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013. ISBN: 9781107030657.

[11] *How a paper machine line works*. Metso. URL: `http://www.metso.com/corporation/info_eng.nsf/WebWID/WTB-060628-2256F-1BCC2`.

[12] M. Rodríguez Pascual. "Konceptstudiemodell av TMP". Internal document Solvina AB. 2013.

[13] J. Pettersson. "Energikartläggning för energiledningssystem. Holmen Paper, Hallsta". Luleå University of Technology, Applied Physics - Division of Energy Science, 2005.

[14] N. J. Sell and P. E. Chair. *Process Control Fundamentals For The Pulp & Paper Industry*. Tappi Press, 1995. ISBN: 978898522945.

[15] H. Elmqvist, H. Tummescheit, and M. Otter. "Object-Oriented Modeling of Thermo-Fluid Systems". In: *Proceedings of the 3rd International Modelica Conference*. Linköping, Sweden, Nov. 3–4, 2003, pp. 9–15.

[16] R. Franke and J. Doppelhamer. "Online Application of Modelica Models in the Industrial IT Extended Automation System 800xA". In: *5th Modelica Conference*. Sept. 4–5, 2006.

[17] L. Imsland, P. Kittilsen, and T. S. Schei. "Model-based optimizing control and estimation using Modelica models". In: *Modeling, Identification and Control* Vol. 31.No. 3 (2010).

[18] P. Kittilsen, S. O. Hauger, and S. O. Wasbø. "Designing models for online use with Modelica and FMI". In: *Proceedings of the 9th International Modelica Conference*. Munich, Germany, Sept. 3–5, 2012, pp. 197–204.

[19] S. Ungarala, E. Dolence, and K. Li. "Constrained Extended Kalman Filter for Nonlinear State Estimation". In: *8th International IFAC Symposium on Dynamics and Control of Process Systems*. Cancún, Mexico, June 6–8, 2007, pp. 63–68.

[20] S. Kolås, B. A. Foss, and T.S. Schei. "Constrained nonlinear state estimation based on the UKF approach". In: *Computers and Chemical Engineering* Vol. 33 (2009), 1386–1401.

[21] J. I. Videla and B. Lie. "Using Modelica/Matlab for parameter estimation in a bioethanol fermentation model". In: *Proceedings of the 6th International Modelica Conference*. Bielefeld, Germany, Mar. 3–4, 2008, pp. 287–299.

[22] M. Bonvini, M. Wetter, and M. D. Sohn. "An FMI-based Framework for State and Parameter Estimation". In: *Proceedings of the 10th International Modelica Conference*. Lund, Sweden, Mar. 10–12, 2014, pp. 647–656.

[23] J. Brembeck et al. "Nonlinear State Estimation with an Extended FMI 2.0 Co-Simulation Interface". In: *Proceedings of the 10th International Modelica Conference*. Lund, Sweden, Mar. 10–12, 2014, pp. 53–62.

[24] *FMI Library (FMIL)*. URL: `http://www.jmodelica.org/FMILibrary/`.

[25] *PyFMI*. URL: `http://pypi.python.org/pypi/PyFMI/`.

[26] *JModelica.org*. URL: `http://www.jmodelica.org`.

[27] *javaFMI*. URL: `http://bitbucket.org/siani/javafmi`.

[28] C. Schweers et al. "Automated Design of an Unscented Kalman Filter for State- and Parameter Estimation on unknown Models". In: *Control, Automation, Robotics and Embedded Systems (CARE), 2013 International Conference*. Jabalpur, India, Dec. 16–18, 2013.

[29] A. Johnsson. "Nonlinear Model Predictive Control for Combined Cycle Power Plants". MSc thesis. Lund University, Department of Automatic Control, 2013.

[30] E. Dahlquist, B. Widarsson, and E. Tomás-Aparicio. *Demand-based maintenance and operators support based on process models*. Project 1231. Värmeforsk, Feb. 2012.

A | MATLAB Code

A.1 build_dsin

```
function build_dsin(experiment,method,settings,initialName,initialValue,quiet)

    % Check that dsin_init exist, if not: create it.
    if ~exist('dsin_init.mat', 'file')
        dos('dymosim -i dsin_init.txt');
        dos('alist -b dsin_init.txt dsin_init.mat');
        delete('dsin_init.txt');
        if ~quiet
            disp('dsin.txt created!');
        end
    end

    % Load init dsin for default values.
    dsin = load('dsin_init.mat');

    % Set experiment
    if length(experiment) == 7
        dsin.experiment = experiment(:);
    elseif ~quiet
        disp('Experiment wrong size!');
    end

    % Set method
    if length(method) == 27
        dsin.method = method(:);
    elseif ~quiet
        disp('Method wrong size!');
    end

    % Set settings
    if length(settings) == 13
        dsin.settings = settings(:);
    elseif ~quiet
        disp('Settings wrong size!');
    end

    % Set initialValue for initialName
    if size(initialName,1) == length(initialValue)
        for i = 1:length(initialValue)
            n = tnindex(dsin.initialName,tnblank(initialName(i,:)));
            if n
                dsin.initialValue(n,2) = initialValue(i);
            elseif ~quiet
                disp('Failed to set value!');
            end
        end
    elseif ~quiet
       disp('Size of initialName and initialValue have to be the same!');
    end
```

```
    % Save changed dsin file.
    dsin = rmfield(dsin,'initialDescription');
    save('dsin.mat', '-struct', 'dsin','-v4');
    if ~quiet
        disp('dsin.mat created!');
    end

end
```

A.2 build_dsu

```
function build_dsu(names,data,quiet)

    % Build struct
    dsu.Aclass = char('Atrajectory','1.0','Generated by Matlab');
    dsu.names = char('time',names);
    dsu.data = [0 data(:)'];

    % Save dsu file.
    save('dsu.mat', '-struct', 'dsu','-v4');
    if ~quiet
        disp('dsu.mat created!');
    end

end
```

A.3 check_inputs

```
function x_ret = check_inputs(x,x_min,x_max,x_names)

    x_ret = zeros(size(x));

    for i = 1: length(x)
        x_ret(i) = min(max(x(i),x_min(i)),x_max(i));
        if ~isequal(x(i),x_ret(i))
            disp([tnblank(x_names(i,:)) ' limited to ' num2str(x_ret(i)) ' from '
                num2str(x(i)) '!']);
        end
    end

end
```

A.4 ekf

```
function [x2,P2]=ekf(model_path,x_init_names,x_names,x0,u_names,u0,y_names,y0,P0,Q,
    R,ts)

% Change path to model dymosim.exe
old_path = cd(model_path);

% Delete old files to avoid errors
delete('dsu.mat','dsres.mat','dslin.mat');

% Settings for Dymosim simulation
experiment = [0 ts 0 1 0.001 0 8];
settings = [0 1 0 1 1 0 0 0 0 0 1 0 1];

% Create files for Dymosim
build_dsin(experiment,0,settings,x_init_names,x0,1);
build_dsu(u_names,u0,1);

% Linearize and discretize
[~] = evalc('dos(''dymosim.exe -l -u dsu.mat dsin.mat'')');
[~,A,~,C,~]=evalc('tloadlin');

% Check system for observability. Halt if not observable!
O = obsv(A,C);
if ~rank(O,0.01) == length(x0)
    disp('System is not observable!');
    pause;
end

% Discretize system matrices
F = expm(A*ts);
H = C;

% Simulate to find x_k|k-1 and y_k
[~] = evalc('dos(''dymosim.exe -s -u dsu.mat dsin.mat'')');
[~,s,n] = evalc('tload(''dsres.mat'')');

% Extract values for x from the result file
x1 = zeros(size(x_names,1),1);
for i = 1:size(x_names,1)
    x1(i) = s(end,tnindex(n,x_names(i,:)));
end

% Extract values for y from the result file
y = zeros(size(y_names,1),1);
for i = 1:size(y_names,1)
    y(i) = s(end,tnindex(n,y_names(i,:)));
end

% Compute the covariance matrix
P1 = F*P0*F' + Q;

% Compute the Kalman gain
K = (P1*H')/(H*P1*H' + R);

% Calculate x_k|k and P_k|k
x2 = x1 + K*(y0(:) - y);
P2 = (eye(size(K*H)) - K*H)*P1;

% Change to old path
cd(old_path);
```