JASS: A Java Based Activated Sludge Process Simulator

Pär Samuelsson

Contents

1 Purpose of the Master Thesis

The goal of this work has been to build a simulator for the activated sludge process in a wastewater treatment plant. Some of the key ideas has been:

- Testing the programming language Java for an advanced and computationally demanding task.

- Construction of a simulator where different control strategies may be implemented and studied.

- To construct a simulator with a graphical users interface (GUI) where data from the simulations are presented pedagogically in real time.

- Development of a simulator that is easy to use, for example, personnel at a waste water treatment plant should after short training be able to use the simulator.

- To design a simulator that can be loaded via internet and run with a standard web browser, such as *Netscape Communicator* or *Internet Explorer*

2   Fundamentals of Wastewater Treatment

2.1   A Short Introduction to Wastewater Treatment

Modern wastewater treatment is a fairly complex process which includes several treatment steps before the water is released to the recipient. A very typical strategy is to have four different process steps, (see also Figure 1):

- Mechanical treatment

- Biological treatment

- Chemical treatment

- Sludge treatment

In the first step, the mechanical treatment, larger objects are collected on a grid and heavy particles like sand are trapped in a sand trap. Also, the mechanical treatment step often includes a primary sedimentation where particles are allowed to settle.

The second treatment step is the biological treatment, and one version of this biological treatment, *the activated sludge process* (ASP), is what is simulated in this master thesis. There exist several different biological processes for wastewater treatment, but the most common one is the activated sludge process. In the ASP, different microorganisms decompose organic matter. It is also possible to extend the ASP for nitrogen removal, see Section 2.2. The biological step also includes a settling tank, where microorganisms and particulate matter can settle. The sludge that accumulates on the bottom of the settling tank are partly removed, and partly recirculated back into the process in order to keep the concentration of microorganisms in the water on a sufficiently high level.

The most common purpose with the chemical treatment step is to remove phosphorus. This is done by adding precipitation chemicals to the wastewater. These chemicals will convert the solved phosphorus into insoluble compounds and also stimulate flocculation. The flocks may then be removed either by sedimentation or by flotation. Another possible chemical treatment is to remove pathogens by chlorinating the water, but this is not very common in Sweden.

Sludge from the three steps described above is fed to the sludge treatment, which includes several partial steps. The sludge that is removed from the different process blocks has a high water content and must thus be thickened. The next step is to stabilize the sludge to reduce odor and to kill potential pathogens. This can be done in an anaerobic digester, where the high temperature kills most microorganisms, and organic matter is degraded. See also (Henze *et al.* 1987).

2.2   Biological Nitrogen Removal

One may ask: Why should the nitrogen be removed from the wastewater? A simplified answer is that these substances may be toxic to animals, and also that they often are the limiting nutrients in the recipient. Below is a short overview of the different nitrogen substances and their impact on the recipient.
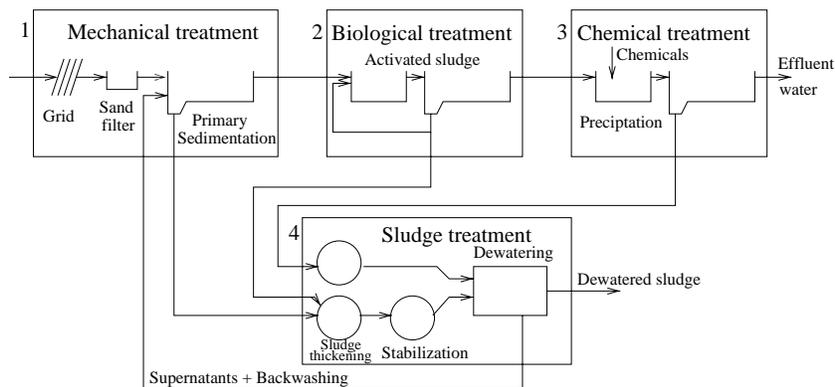
Figure 1: Typical layout of a wastewater treatment plant

- Ammonia (NH_3) is toxic to animals.

- Ammonium (NH_4^+) may be toxic, but is also often a limiting nutrient in the recipient. The fact that ammonium oxidizes to nitrate may cause oxygen depletion in the receiving water.

- Nitrite (NO_2^-) is toxic.

- Nitrate (NO_3^-) may be a limiting nutrient in the recipient.

- Presence of nitrogen in drinking water (which may be taken from the recipient) may cause high levels of bacteria in the water.

To remove these different nitrogen pollutions from the wastewater, one must use some microbiological knowledge. The main nitrogen pollution in the water that comes into a plant is generally ammonium. Ammonium can with the aid of a certain microorganism be oxidized (with nitrite as a intermediate product) to nitrate under aerobic conditions (nitrification). Nitrate may then under anoxic conditions be converted to N_2 gas with the help of another micro organism and organic matter (denitrification). The microorganisms of the denitrification process degrades the readily degradable organic matter, which is often limiting the denitrification process efficiency, and carbon from an external carbon source must thus be added in many cases. The facts above tell us that an activated sludge process needs both anoxic and aerobic basin zones to manage a high level of nitrogen removal.

2.3   Some Basic Process Descriptions

A very basic activated sludge process consists of two main parts: An aerated tank and a settler. Such a process is shown in Figure 2. As mentioned in previous sections the activated sludge process is a microbiological process where microorganisms decompose organic matter. All microorganisms comes into the plant with the influent water. The conditions in the aerated tank are such that the concentrations of microorganisms may grow. The air that is blown into the tank increases the oxygen concentration in the water, and the increased oxygen concentration gains microorganisms that oxidize organic matter. In the settler particulate matter and microorganisms can settle and
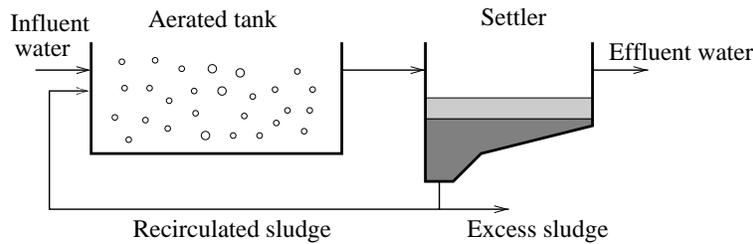
Figure 2: An activated sludge process with an aerated tank and a settler.

be removed from the process as excess sludge. To keep the concentration of microorganisms sufficiently high in the tank, sludge must also be recirculated from the settler.

As concluded in the previous subsection though, both anoxic and aerobic conditions are needed to remove nitrogen as from the wastewater (excluding the nitrogen removal obtained from assimilation to the sludge, see section 3.3. There are two main kinds of plant configurations, so called pre denitrifying and post denitrifying systems.

In a post denitrifying system, the aerobic zones are put first in the line and there is no recirculation of water from the last zone to the first. In this kind of system, readily degradable organic matter often limits the process efficiency. One may therefore have to add carbon from an external carbon source to the anoxic zones to keep a high efficiency on the denitrification that are depending on high carbon concentrations.

In a pre denitrifying system, the organic matter in the influent water is better used since the anoxic zones here are put first in the line. A pre denitrifying configuration is shown in Figure 3. The main drawback with this kind of configuration is that water must be recirculated from the last zone to the first. The reason for this recirculation is to transfer nitrate rich water to the anoxic zone.

Figure 3: An activated sludge process with pre denitrification.

3 The IAWQ Model

One of the most common models for the activated sludge process is the IAWQ Activated Sludge Model No. 1, and this is the model used in the simulator described in section 4. The m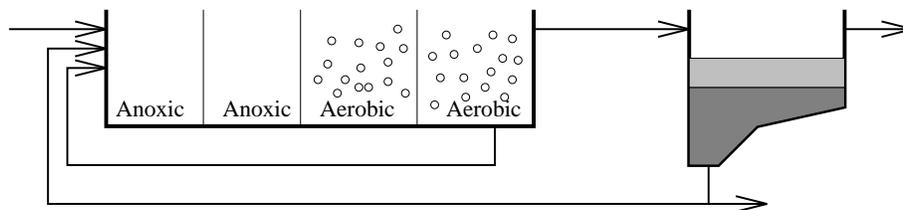odel describes the three processes described in the previous sections, i.e removal of organic matter, nitrification and denitrification. See also (Jeppsson 1996a) and (Henze et $al.$ 1987). An overview of the model and its components is presented below.

3.1 Organic Matter

In the model, the organic matter (the total COD) is assumed to consist of three main components that is further divided into different subcomponents.

- Biologically degradable organic matter.
  There are two different types of this material, readily degradable organic matter, S_S, and slowly degradable organic matter, X_S.

- Non biodegradable organic matter.
  This substance is divided into two fractions, soluble inert organic matter, S_I, and particulate inert organic matter. The particulate inert organic matter is divided into two subcomponents, X_P that derives from biomass decay, and X_I that derives from the incoming water.

- Active biomass (microorganisms).
  Two main kinds of microorganisms are taken into account in the model: Heterotrophic bacteria, $X_{B,H}$, which under aerobic conditions decompose the main part of the organic matter. These microorganisms may under anoxic conditions cause the desired denitrification process. There are also autotrophic bacteria, $X_{B,A}$ that under aerobic conditions convert ammonium into nitrate (nitrification).

The different reaction phases of the organic matter in the model are schematically described in Figure 4.

3.2 Different Fractions of Nitrogen and Other Compounds

The nitrogen pollutions are divided into the following groups:

- Ammonium, S_{NH}

- Nitrite and nitrate, S_{NO}

- Soluble organic nitrogen, S_{ND}

- Particulate organic nitrogen, X_{ND}

There are two more components included in the model, soluble oxygen, S_O and the alkalinity S_{ALK}. The soluble oxygen is important in many of the biochemical reactions of the model, while the alkalinity does not affect the reactions, it is just a measure of the pH sensitivity of the water.
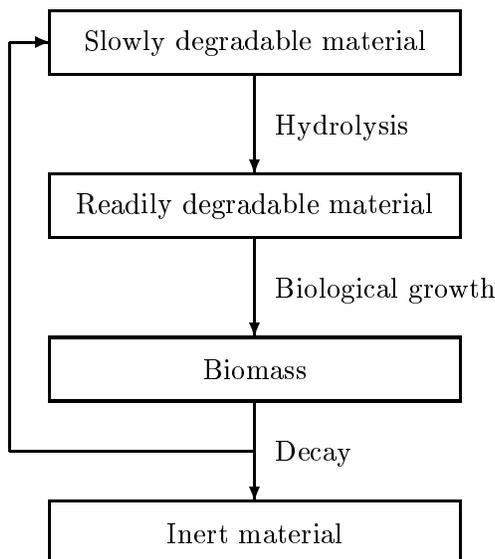
Figure 4: The biological decomposition chain

3.3   Biological and Chemical Reactions

The behavior of each of the substances described in the previous subsection is in the model described by nonlinear ordinary differential equations. These consist of two terms: An ordinary mass balance term and one reaction term. A general formula for such a differential equation in one specific basin zone is given below.

$$\frac{dZ}{dt} = \frac{Q}{V}(Z_{in} - Z) + R \tag{1}$$

Here, Z is the concentration of a certain substance in the zone, Z_{in} is the concentration of the substance in the water that flows into the zone, Q is the flow through a zone, V is the volume of the zone and R is the biochemical reaction rate for the specific substance in the zone. There is a number of biochemical reactions (R) that are taken into account in the model, and an overview is given below. In the model, these reactions is normally presented in matrix format. This process matrix is found in Appendix E.

- Aerobic growth of heterotrophs, $X_{B,H}$.
  This process degrades organic matter, and will take place under aerobic conditions if enough organic matter S_S is present. See row 1 in the process matrix.

- Anoxic growth of heterotrophs, $X_{B,H}$.
  This is the denitrification process, which will take place when the oxygen concentration is low and sufficient amounts of readily degradable organic matter, S_S, and nitrate S_{NO} are present. See row 2 in the process matrix.

- Aerobic growth of autotrophs, $X_{B,A}$.
  This is the nitrification process and takes place if there is enough oxygen, S_O, and ammonium, S_{NH}, in the system. The result is that ammonium is turned into nitrate. See row 3 in the process matrix.

- Decay of heterotrophs.
  Hereby, heterotrophs are turned into slowly degradable organic matter, X_S and inert organic matter X_P. Also, some particulate organic nitrogen, X_{ND} is created. See row 4 in the process matrix.

- Decay of autotrophs.
  See above and row 5 in the process matrix.

- Ammonification of soluble organic nitrogen, S_{ND}. See row 6 in the process matrix

- Hydrolysis of entrapped organics.
  This process describes the transfer of slowly degradable organic matter to readily degradable organic matter. See Figure 4 and row 7 in the process matrix.

- Hydrolysis of particulate organic nitrogen, X_{ND} to soluble organic nitrogen S_{ND}. See row 8 in the process matrix.

Below is a overview of how the different components of the model are affected by the reactions above. The differential equation for each substance is also presented. Note that the mass balance term of equation (1) has been excluded, the differential equations only present the reaction part, R. A short explanation of the different constants (or rather parameters) in the equations and their typical values are found in Appendix B. Some of these parameters show very little variation and can therefore be considered as constants while other are strongly dependent of temperature and other environmental factors.

- Readily degradable organic matter, S_S, is consumed by aerobic and anoxic growth of heterotrophs and generated by hydrolysis. The differential equation for the reaction part is presented below.

$$\frac{dS_S}{dt} = -\frac{1}{Y_H}\hat{\mu}_H\left(\frac{S_S}{K_S + S_S}\right)\left(\frac{S_O}{K_{O,H} + S_O}\right)X_{B,H}$$
$$-\frac{1}{Y_H}\hat{\mu}_H\left(\frac{S_S}{K_S + S_S}\right)\left(\frac{K_{O,H}}{K_{O,H} + S_O}\right)\left(\frac{S_{NO}}{K_{NO} + S_{NO}}\right)\eta_g X_{B,H}$$
$$+ k_h \frac{\frac{X_S}{X_{B,H}}}{K_X + \frac{X_S}{X_{B,H}}}\left(\left(\frac{S_O}{K_{O,H} + S_O}\right) + \eta_h\left(\frac{K_{O,H}}{K_{O,H} + S_O}\right)\left(\frac{S_{NO}}{K_{NO} + S_{NO}}\right)\right)X_{B,H}$$

- Slowly degradable organic matter X_S is generated by decay of microorganisms and consumed by hydrolysis. The reaction part of the differential equation is presented below.

$$\frac{dX_S}{dt} = (1 - f_P)(b_H X_{B,H} + b_A X_{B,A})$$
$$- k_h \frac{\frac{X_S}{X_{B,H}}}{K_X + \frac{X_S}{X_{B,H}}}\left(\left(\frac{S_O}{K_{O,H} + S_O}\right) + \eta_h\left(\frac{K_{O,H}}{K_{O,H} + S_O}\right)\left(\frac{S_{NO}}{K_{NO} + S_{NO}}\right)\right)X_{B,H}$$

- Autotrophic microorganisms, $X_{B,A}$, grows if oxygen and ammonium are present. The $-b_A X_{B,A}$ term in the equation is caused by decay.

$$\frac{dX_{B,A}}{dt} = \hat{\mu}_A\left(\frac{S_{NH}}{K_{NH} + S_{NH}}\right)\left(\frac{S_O}{K_{O,A} + S_O}\right)X_{B,A} - b_A X_{B,A}$$

- Heterotrophs, $X_{B,H}$, grows under aerobic conditions if enough substrate, S_S, is available. Heterotrophs may also grow under anoxic conditions if there is enough nitrate to replace the oxygen and enough substrate S_S is available. The $-b_H X_{B,H}$ term in the equation describes the decay.

$$\frac{dX_{B,H}}{dt} = \hat{\mu}_H (\frac{S_S}{K_S + S_S})(\frac{S_O}{K_{O,H} + S_O}) X_{B,H}$$
$$+ \hat{\mu}_H (\frac{S_S}{K_S + S_S})(\frac{K_{O,H}}{K_{O,H} + S_O})(\frac{S_{NO}}{K_{NO} + S_{NO}}) \eta_g X_{B,H} - b_H X_{B,H}$$

- Particulate inert organic matter, X_P, is generated by decay of microorganisms. No further terms are present in the equation since this material is inert and does not react further.

$$\frac{dX_P}{dt} = f_P (b_H X_{B,H} + b_A X_{B,A})$$

- Dissolved oxygen, S_O, is consumed by aerobic growth of heterotrophs and autotrophs.

$$\frac{dS_O}{dt} = -\frac{1 - Y_H}{Y_H} \hat{\mu}_H (\frac{S_S}{K_S + S_S})(\frac{S_O}{K_{O,H} + S_O}) X_{B,H}$$
$$- \frac{4.57 - Y_A}{Y_A} \hat{\mu}_A (\frac{S_{NH}}{K_{NH} + S_{NH}})(\frac{S_O}{K_{O,A} + S_O}) X_{B,A}$$

- Nitrate S_{NO} is turned into nitrogen gas by denitrification (anoxic growth of heterotrophs). Nitrate is formed by aerobic growth of autotrophs (nitrification).

$$\frac{dS_{NO}}{dt} = -\frac{1 - Y_H}{2.86 Y_H} \hat{\mu}_H (\frac{S_S}{K_S + S_S})(\frac{K_{O,H}}{K_{O,H} + S_O})(\frac{S_{NO}}{K_{NO} + S_{NO}}) \eta_g X_{B,H}$$
$$+ \frac{1}{Y_A} \hat{\mu}_A (\frac{S_{NH}}{K_{NH} + S_{NH}})(\frac{S_O}{K_{O,A} + S_O}) X_{B,A}$$

- Ammonium, S_{NH}, is turned into nitrate by aerobic growth of autotrophs (nitrification). Some of the ammonium is assimilated in the sludge. Ammonium is formed by ammonification of soluble organic nitrogen.

$$\frac{dS_{NH}}{dt} = -i_{XB} \hat{\mu}_H (\frac{S_S}{K_S + S_S})(\frac{S_O}{K_{O,H} + S_O}) X_{B,H}$$
$$- i_{XB} \hat{\mu}_H (\frac{S_S}{K_S + S_S})(\frac{K_{O,H}}{K_{O,H} + S_O})(\frac{S_{NO}}{K_{NO} + S_{NO}}) \eta_g X_{B,H}$$
$$- (i_{XB} + \frac{1}{Y_A}) \hat{\mu}_A (\frac{S_{NH}}{K_{NH} + S_{NH}})(\frac{S_O}{K_{O,A} + S_O}) X_{B,A} + k_a S_{ND} X_{B,H}$$

- Soluble organic nitrogen, S_{ND}, is consumed by ammonification and generated by hydrolysis of particulate organic nitrogen.

$$\frac{dS_{ND}}{dt} = -k_a S_{ND} X_{B,H}$$
$$+ \frac{X_{ND}}{X_S} k_h \frac{\frac{X_S}{X_{B,H}}}{K_X + \frac{X_S}{X_{B,H}}} \left((\frac{S_O}{K_{O,H} + S_O}) + \eta_h (\frac{K_{O,H}}{K_{O,H} + S_O})(\frac{S_{NO}}{K_{NO} + S_{NO}}) \right) X_{B,H}$$

- Particulate organic nitrogen, X_{ND}, is consumed by hydrolysis and generated by decay of microorganisms.

$$\frac{dX_{ND}}{dt} = (i_{XB} - f_P i_{XP})(b_H X_{B,H} + b_A X_{B,A})$$

$$- \frac{X_{ND}}{X_S} k_h \frac{\frac{X_S}{X_{B,H}}}{K_X + \frac{X_S}{X_{B,H}}} \left((\frac{S_O}{K_{O,H} + S_O}) + \eta_h (\frac{K_{O,H}}{K_{O,H} + S_O})(\frac{S_{NO}}{K_{NO} + S_{NO}}) \right) X_{B,H}$$

The careful reader may here note that the soluble inert material, S_I, the alkalinity, S_{alk}, and the particulate inert organic matter that derives from the incoming water, X_I, is not included in the overview above. These components are considered unaffected by the biological and chemical reactions and are therefore excluded. These are modeled as pure mass balances.

4 Design of the Simulator

There are many broad definitions of the meaning of the word simulation, but this project is concentrating on the following meaning: Simulations is a way of solving problems numerically with the aid of mathematical models. This is generally done when it is impossible or too expensive to solve the problem analytically or by testing different solutions in reality, i.e on the real process. In all simulation studies of this kind it is of fundamental importance to be aware of model approximations and the validity of the model. One must note that different needs often lead to different requirement of the accuracy of the underlying model. The general concept of simulations is also discussed in (Ljung and Glad 1991)

There are many reasons for designing simulators for the activated sludge process with the aid of mathematical models. The most common reasons are shown below.

- Testing different plant configurations.

- Testing different control strategies.

- Education of plant personnel and students.

- Optimization of the plant.

- Forecasting. By simulations, future plant performance can be predicted.

A further advantage with simulators for wastewater treatment plants is that the real system is very slow, the effect of a process change may take weeks to fully observe. With a simulator one hour of real time may be simulated in a fraction of a second.

4.1 Basic Simulator Setup

In the default version, the simulator simulates a post denitrification process with 10 zones and a simple settler. The model that has been implemented is the IAWQ model described in the previous section, except that for the two fractions of particulate organic matter, X_I and X_P has been gathered into one component, further denoted X_I. The simulator is controlled with a graphical users interface (GUI) where also data from the simulations are presented in real time. The number of zones as well as the placement of pumps are fixed. To run the simulator is fairly simple, and it is easy to change the plant configuration such as aeration of zones, flow rates, concentrations of incoming components etc. See the users manual in section 6 for a more detailed explanation on these practical aspects. It is possible to run the simulator with an ordinary web browser such as *Netscape Communicator 4.0*. Details on the implementation of the model is given in Appendix A.

4.2 The Modeling of One Mixed Zone

Since the programming language Java is object oriented one mixed zone can be looked upon as a separate block with its own characteristics such as volume, constants and flows (see also (Luttmer 1995)). Figure 5 shows a schematic overview of a zone. Here Qin is the inflow of wastewater to the zone, Qpre is the flow from the previous zone, Qreg is the flow recirculated from the settler, Qcar is the external carbon flow and Air is the air flow. Associated with each flow are of course concentrations of all the

components described in the previous section. Note that all zones has these properties, for instance each zone has an inflow of external carbon but it is only possible to set this flow to something else than zero in one of the zones. It may be of interest that
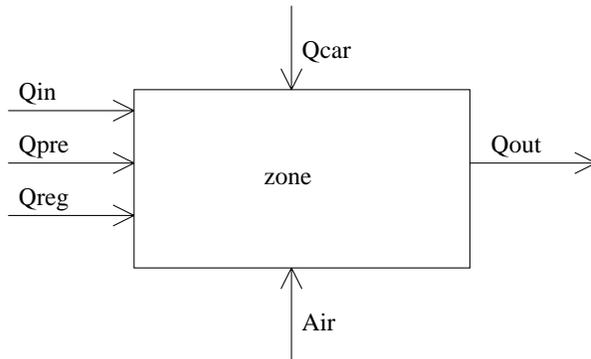
Figure 5: The flows into a zone.

the external carbon source in the simulator is ethanol and has a COD of 1200000 mg/l.

The effect of the external air flow is modeled as $K_L a(u)(S_{O,sat} - S_O)$. Here u is the air flow rate, the function $K_L a(u)$ describes the transfer function of the external aeration system, and $S_{O,sat}$ is the oxygen concentration at saturation. This equation comes in as a positive term in the differential equation for oxygen presented in section 3.3. The oxygen transfer function $K_L a(u)$ is here modeled as $K_L a(u) = ku$ where k is a constant that describes the efficiency of the process. The simulator will later be extended to allow for nonlinear and volume dependent $K_L a(u)$.

In the simulations the 12 differential equations of the form $\frac{dZ}{dt} = \frac{Q}{V}(Z_{in} - Z) + R$ described in section 3.3 are solved numerically for each zone with a fourth order Runge-Kutta method. The algorithm for this method is shown below.

$$
\begin{aligned}
\frac{dy_i}{dt} &= f_i(t, y_1, ..., y_N), i = 1....N \\
k_1^i &= f_i(t_k, y_1^k, ..., y_N^k) \\
k_2^i &= f_i(t_k + \frac{h}{2}, y_1^k + \frac{h}{2}k_1^i, ..., y_N^k + \frac{h}{2}k_1^i) \\
k_3^i &= f_i(t_k + \frac{h}{2}, y_1^k + \frac{h}{2}k_1^i, ..., y_N^k + \frac{h}{2}k_1^i) \\
k_4^i &= f_i(t_k + h, y_1^k + hk_3^i, ..., y_N^k + hk_3^i) \\
y_i^{k+1} &= y_i^k + \frac{h}{3}(\frac{1}{2}k_1^i + k_2^i + k_3^i + \frac{1}{2}k_1^i) \\
t_{k+1} &= t_k + h
\end{aligned}
$$

In our case h is the time step, y_i^k is the concentration of substance i at time step k and k_1^i, k_2^i, k_3^i, k_4^i are the four Runge-Kutta extrapolation coefficients of the i:th substance. The function $f_i(t, y_1, ..., y_N)$ is simply the process rate given in equation (1) in section 3.3, i.e the time derivative of the i:th substance concentration y_1. The

time step h is fixed at 0.01 h in the simulations. See also Appendix A.16 for a description of the implementation.

4.3   The Modeling of the Settler

For the implemented simple settler, two general assumptions are made:

- No reactions are assumed to take place in the settler. The behavior of the different model components is therefore modeled as a pure mass balance.

- Ideal settling, i.e all particulate matter settles, and the concentrations of the particulate components are therefore zero.

One may here note that the first assumption will lead to that the concentrations of the soluble components will be unchanged by the settler. A schematic overview of the settler is shown in Figure 6.
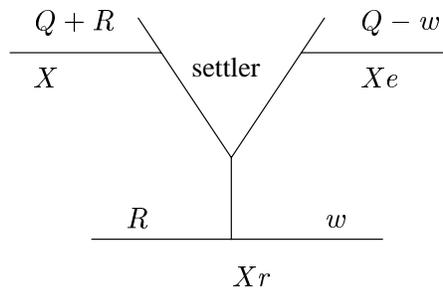
Figure 6: The settler.

Here, $Q + R$ is the inflow to the settler, X is the concentration of an arbitrary particulate component, R is the flow that is recirculated back into the zones, w is the excess sludge flow, X_r is the concentration of the particulate component in the excess sludge and in the recirculated sludge, $Q - w$ is the effluent water flow and X_e is the particulate concentration in the effluent water ($X_e = 0$ under the assumption of ideal settling). The mass balance differential equation to be solved can thus be written as:

$$\frac{dX_r}{dt} = \gamma \frac{Q + R}{V} X - \frac{Q + R}{V} X_r$$

Here V is the volume of the settler and $\gamma = \frac{Q+R}{R+w}$. The numerical solution is similar to the one described in the previous subsection. See also Appendix A.17 for details on the implementation.

4.4   The Controllers

4.4.1   The Oxygen PID Controllers

Ten oxygen PID controllers (one for each zone) are implemented in the simulator. The controlled outputs are the concentrations of dissolved oxygen, S_O, in each zone. The control signals that the controllers give out are the air flow rates to each zone

(u in the function $KL_a(u)$). These controllers are all tuned by hand. The output are sampled every 45:th time step in the ODE, i.e every 0.45 hour in simulated time.

4.4.2   The External Carbon Controllers

Two kinds of controllers have been implemented for control of external carbon source in the simulator. The controlled output is here the concentration of nitrate, S_{NO} in the last anoxic zone, and the control signal is an external carbon flow rate to the first anoxic zone. First, a PID controller of the same type used for oxygen was implemented. Secondly a model based polynomial LQG controller was evaluated, see section 5. This controller is the one used in the simulator. The sampling interval here is the same as for the oxygen PID controllers.

4.5   The Graphical Users Interface (GUI)

The GUI was constructed so that the simulator would be easy to run, and also so that simulated data could be presented in a simple and pedagogical manner.

The data from the simulations are presented in two different ways: There are three bar diagrams where the profiles of component concentrations, air flows and external carbon flow rates are plotted. These bar diagrams show the concentrations of a chosen component in the incoming water, all the ten zones and in the recirculated water. There is also a scrolling graph in which the user may plot two components or flows chosen from a specific zone.

During simulation, the user can easily change many of the simulation parameters such as reference values of oxygen, process constants, zone volume, flow rates etc. These changes are generally done by writing the new parameter value in a textfield, and then enter it to the simulations by pressing a corresponding *OK* button. Some of the textfields are located in so called dialog boxes that shows up when the user clicks a certain field or symbol. More details on how to run the simulator is given in the section *Users Manual*. Figure 7 shows the GUI. The bar diagrams are located to the upper left and the scrolling graph to the upper right. The GUI is also described in Appendix C.
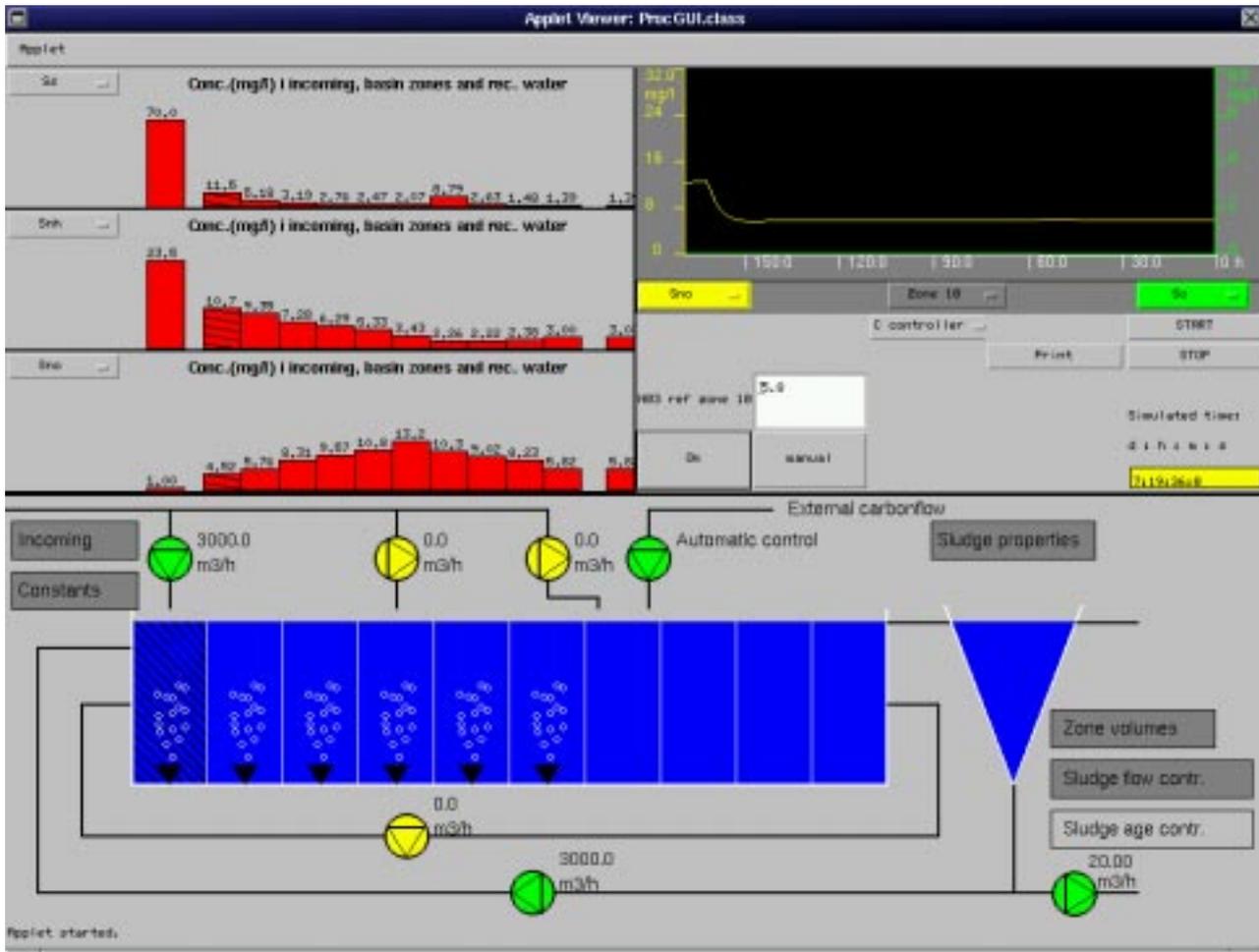
Figure 7: The GUI.

5   The External Carbon Flow Rate Controller

5.1   Motivations And Strategies for External Carbon Flow Rate Control

As mentioned earlier, the concentration of readily degradable organic matter, S_S, is of vital importance for the biological processes in a wastewater treatment plant. This is due to the fact that most microorganisms need carbon to build their biomass. The denitrification bacteria is particularly sensitive to the concentration of carbon substances in the water, and this motivates the addition of an external carbon source to the process, the process efficiency for the denitrification may hereby be raised.

If the concentrations of readily degradable organic matter in the influent water is low, it is generally preferable run the plant with a post denitrifying zone configuration. If high levels of readily degradable organic matter are present in the influent water, a pre denitrifying configuration may be an alternative since the carbon in the influent water then may be used in the denitrification process. The main drawback with a pre denitrifying system is that water must be recirculated from the last zone to the first, which may be expensive.

Independent of whether a pre or post denitrifying configuration is used, the controlled output is in general the concentration of soluble nitrate, S_{NO}, in the last anoxic zone. The control signal should be an external carbon flow rate to the first anoxic zone, this to give impact on all the anoxic zones. See (Lindberg 1997) for more information on this subject.

5.2   System Identification of the Denitrification Process

The purpose with system identification in this case is to obtain a simplyfied linear model of the highly nonlinear denitrification process. Since the model is based on sampled process data it will be discrete time. The model may then be used for designing a model based linear controller of the process. The fact that the model and the controller will be time discrete makes the controller very easy to implement on a computer, which would not be the case if a continuous time controller should be used. It has been suggested (Lindberg 1997) that it would be suitable to identify a model of the denitrification process in an ARX structure. The general structure of such a model is shown below.

$$A(q^{-1})y(t) = B(q^{-1})q^{-k}u(t) + n(t)$$

In the case when the mathematical model is used for the denitrification process, $y(t)$ corresponds to the concentration of S_{NO} in the last anoxic zone, $u(t)$ is the external carbon flow rate and $n(t)$ could be seen as process noise (for instance caused by variations in the concentrations of the components in the influent water). The model is here written in polynomial form using the backward shift operator.

Data for the identification was collected from a Matlab-C++ version of the simulator where the system was excited with a pseudo random binary sequence as input (carbon flow rate). The model was then identified in the ARX structure above with the Matlab command *arx*. In Figure 8, a cross validation of the model is shown. The plot shows simulated data, data from the real system and the input signal. Observe that the average values has been subtracted from the real data and the input respectively. For this case, an A polynomial of order 3, a B polynomial of order 1 and a time delay of $K = 2$ samples gave a model that behaved very similar to the real system. The subject of system identification for ARX models is described in detail by (Söderström and Stoica 1989).

5.3   LQG Controller Design

There exist several controller design methods for sampled systems. In this project, a LQG controller was designed and implemented. The LQG control problem is to find the optimal control $u(t)$ which minimizes

$$J = \lim_{N \to \infty} \frac{1}{2N} \sum_{t=0}^{N} \left[Ey(t)^2 + \rho E(\Delta u(t)^2) \right] \tag{2}$$

Here E is the expected value operator and Δ is in the q^{-1} notation written $1 - q^{-1}$. This criterion is very suitable in this case since it punishes the input increment, i.e
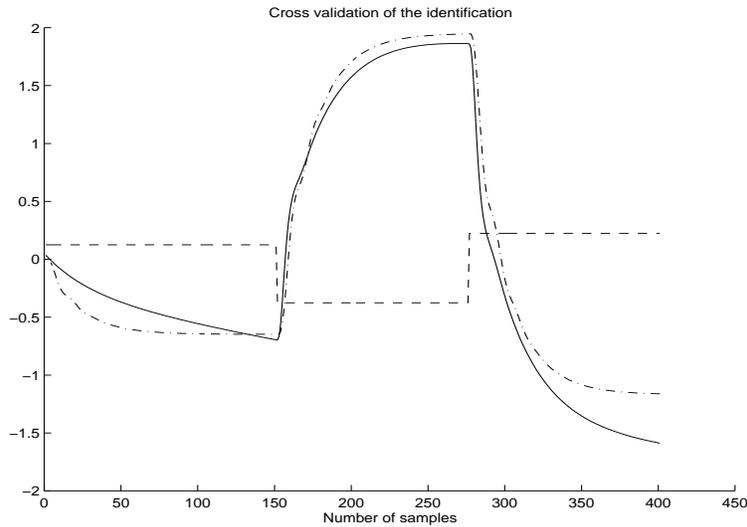
Figure 8: Cross validation of the estimated model. Dashed line shows the input (carbon flow rate) multiplied with 10, dashed-dotted shows the simulated system, full line shows the real system.

the input variations, and the variance of the output $y(t)$. Overshoots and peaks in the signals will thus be smoothed. The parameter ρ is the design variable that the designer may choose, a large ρ corresponds to a system with slowly varying input signals.

The controller was designed in a so called RST structure where the input signal $u(t)$ is shown below

$$\Delta u(t) = -\frac{S(q^{-1})}{R(q^{-1})}y(t) + \frac{T(q^{-1})}{R(q^{-1})}r(t) \tag{3}$$

$$u(t) = u(t-1) + \Delta u(t) \tag{4}$$

where r(t) in this case is the reference signal for S_{NO} in the last anoxic zone. The values of the control signal $u(t)$ is also constrained between 0 and 1. The lower constraint is very natural since a flow can not be negative, the upper bound is chosen so that under normal conditions, the controller will not exceed it. The R, S and T polynomials that minimize the LQG criterion are obtained by solving the two polynomial equations below.

$$r\beta\beta_* = BB_* + \rho A\Delta\Delta_* A_*$$

$$A\Delta R + q^{-k}BS = \beta C$$

The minimization of the criterion corresponds to a pole placement in βC. Here $\frac{1}{C}$ is often regarded as a design variable that should be chosen as a low pass filter. The

index * means reciprocal polynomial in this case. For a complete derivation of the polynomial LQG design see (Åström and Wittenmark 1990).

Controllers designed with some different ρ:s were simulated on the true nonlinear system in Matlab, Figure 9 shows two cases. It turned out that the control was very sensitive to the choice of ρ, and that the system could even be destabilized if ρ was chosen too small. This was probably an impact from input saturations. At time $t = 250$, the reference value of nitrate in the last anoxic zone is changed from 16 (The steady state value if no control applied) to 2. At time $t = 500$ the reference value is changed to 1. It is clearly seen from the figure that the controller with a small ρ tends to be faster and more oscillative than the other, which is in line with the theory. The closed loop system will in this case have poles closer to the origin.

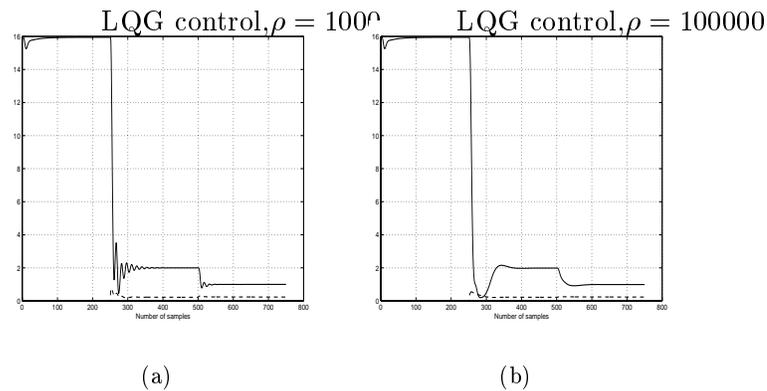Figure 9: (a) System simulated with LQG controller with $\rho = 1000$, full line shows the S_{NO} concentration in the last anoxic zone, dashed line shows the control signal (the external carbon flow rate). (b) The same plot for $\rho = 100000$.

6   Users Manual

Below is a short description on how the simulator is operated:

1. Start *Netscape Communicator* (or Explorer), version 4 or later.

2. Load the actual URL:

   www.syscon.uu.se/~psa/demo/ProcGUI.html

3. If the GUI is too big for the screen choose *hide* on the first three items in the menu *view* in the Communicator.

4. Click on the *Start* button.

5. In case the simulator needs to be reset, just press *STOP* in the GUI and then *shift+reload* in the Communicator menu.

6. To choose a certain compound to plot in the bar diagrams, click on the button to the left of a bar diagram. For instance, to plot the concentration of dissolved oxygen, choose S_O. The oxygen concentrations in incoming water, all basin zones and the settler are then plotted. It is also possible to plot the air flow rates and the external carbon flow rates, this is done by choosing the corresponding items in the menu.

7. In the scrolling graph it is possible to plot the concentrations of two compounds at the same time. Three menu bars are located under the graph. In the middle it is chosen whether to plot concentrations from incoming water, any of the zones, or the return sludge. Two compounds may then be chosen by clicking on the buttons to the left and to the right. By clicking these buttons the user may also choose to plot the air or external carbon flow rate in the specific zone.

8. To change process constants, click in the field *Constants* (upper left corner) in the process image. After a few seconds a new window where the constants may be entered shows up. To make the textfield writable, click the cursor in the textfield. When a new value has been chosen, push the *ok* button in the window. If the button *cancel* is chosen, no changes in the values are done. The dialog box for entering the new constants is shown in Figure 10. Note that the temperature can be chosen in the textfield to the lowest left. The user may enter process constants for 20 °C in the column to the left, the program then converts the process parameters to the actual simulation temperature when the ok button is pushed.

9. To change concentrations of the compounds in the incoming water, click in the field *Set incoming*. A window similar to the one showed in Figure 10 appears, and new values can be entered in the same way.

10. By clicking in the field *Zone Volumes*, a window where the volumes of all zones and the settler can be changed appears.

Figure 10: The dialog box for process parameters.

11. During simulation, it is possible to choose between running the simulator with a fixed excess sludge flow or with a fixed sludge age. In the default setup, the simulator is running with a fixed sludge flow. By clicking in the field *Sludge age control* and enter a desired fixed sludge age in the appearing window, the simulator will run with a fixed sludge age instead. Clicking on the field *Sludge flow control* and enter a desired flow value will now change the mode back so that the simulations run with a fixed excess sludge flow again.

12. When the field *Sludge properties* is clicked, a window that shows the actual sludge age, the excess sludge flow and the sludge concentration of the plant appears.

13. To change a flow value, click on the corresponding pump symbol in the process image and enter new values in the appearing window. An active pump will be colored green, and an inactive yellow. A dialog window for changing a flow is shown in Figure 11. It is possible to choose the flow rates in two different ways: Either as a constant or as a sinusoidal. In the textfield on top the user may choose the average flow rate, in the field in the middle is the magnitude of the sinusoidal to be chosen and on bottom the period time. If a constant flow rate is desired just set the magnitude to zero.

14. When the pump symbol marked *External Carbon flow* in the process image is clicked, a window where the user can enter an external ethanol flow rate (m^3/h)
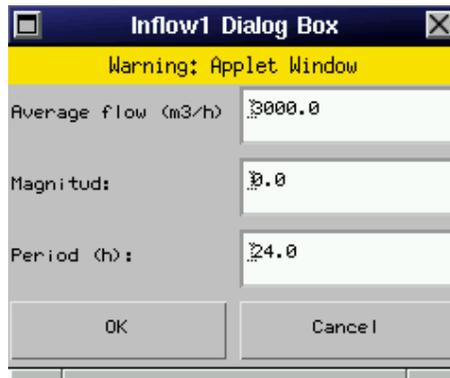
Figure 11: A dialog box for changing the flows.

to the seventh zone appears on the screen. The COD value of ethanol is 1200000 mg/l. Before entering a flow rate, note that you must enter *small* values for this flow rate (in the range 0-100 l/h) otherwise you will have to restart the simulator.

15. To change the reference values of dissolved oxygen, go to the panel under the scrolling graph (located under the menu bar *DO pid*. Here you may choose what zone to change the reference value in. Enter a value in the textfield and then push the *Set DOref* button. Note that the air bubbles in the zone will disappear when $DOref = 0$ is chosen.

16. To run the external carbon flow rate with an automatic controller (LQG), choose *C controller* in the menu *DO pid*. Enter a reference value for S_{NO} in the last anoxic zone in the textfield. The button *Ok* starts the automatic control, the button *manual* switches back to manual control.

7 Results and Conclusions

In the project, a simulator for the activated sludge process of a wastewater treatment plant has been developed. The simulator has been written in the programming language Java and can be loaded and runned with an ordinary web browser. A main aspect has been to try to develop a user friendly graphical user interface. Data from the simulation are shown in real-time using bar diagrams and a traditional time plot.

Many parameters can be changed during simulation. Flow rates are changed by clicking on the corresponding pump symbol and entering a new value in the appearing window. It is also possible to run the simulation with time varying flow rates. The process parameters of the IAWQ model No 1 may also be changed in the same way during simulation. A temperature dependence of the process parameters has been implemented so that the effect of different temperatures at the wastewater can be simulated. Volumes and composition of influent water may also be changed when running the simulation. The simulations can be performed with a fixed excess sludge flow rate or with a fixed sludge age.

A LQG controller for adjusting the external carbon flow rate was implemented. The controller could keep the effluent nitrate level low despite process disturbances.

Some main conclusions that can be drawn from this work are the following:

- The Java language was quite comfortable to work with. It was easy to write a large program with many interacting parts.

- It is fully possible to write computationally demanding differential equation solvers in Java, at least if the time step can be chosen long enough.

- The simulator has been proven to be a powerful tool for education purposes. It has been used in one academic course and one course for plant personnel from two wastewater treatment plants.

- A number of wastewater treatment plants have become interested in the simulator. We believe that two main reasons for this interest are: (i) The simulator is very easy to use. (ii) The simulator can be runned with an ordinary web browser. Currently, the simulator has been adapted to two wastewater treatment plants.

Interesting topics for further consideration could for instance be to implement a supervisory control of the dissolved oxygen concentration and to try more advanced controllers for the external carbon source. Multivariable control could be an efficient tool in optimizing the plant, and may therefore also be tried.

A   Implementation in Java Language

A.1   A Short Java Introduction

Java is an *object-oriented* programming language. For the programmer, this means that the focus is on the data in the application and methods to manipulate that data, rather than thinking in terms of procedures. In an object oriented language, a class is a collection of data and methods that operate on that data. All put together, the data and the methods describe the state and behavior of an object. Classes are arranged in a hierarchy, so that subclasses can inherit behavior from a superclass. This is good to know since Java has a large amount of predefined super classes that handles graphics, networking etc.

Java is an *interpreted* language: The Java compiler generates architecture neutral byte codes for the Java Virtual Machine (the interpreter and run-time system), rather than native machine code. To run a Java program, the computer uses the interpreter to execute the compiled byte-codes. Because of the architecture neutrality and platform independency of the byte-codes, Java programs can run on any computer that the Java Virtual Machine has been ported to. Of course, the interpretation makes Java slower than programs compiled to machine-code, but for many applications, the speed of Java is completely sufficient.

A running Java interpreter can load and create instances of any Java class at any time, therefore Java is said to be a *dynamic* language. The high-level support for networking also makes Java a *distributed language*.

A thing that the programmer appreciate with Java is the simplicity of the language. From the start, Java was designed to be easy to learn and to look familiar to a majority of programmers. Java reminds a lot of C and C++, but many features that causes a lot of trouble in these languages (for instance pointers) has been removed and Java is therefore quite robust.

A big strength with Java is its built-in functions for *multithreading*. In a lot of applications it is advantageous to execute different tasks at the same time, and compared to in C/C++, multithreading is very easy. More information on the subjects in this section can be found in (Flanagan 1990)

A.2   Overview of the program

This section has the purpose to help the reader to get a picture of the program structure and how the different parts of the program interact. To fill this purpose, all classes and their most important functions will be discussed below. Some fundamentals that can be useful to keep in mind when reading further are that the program is running in two separate threads, first we have the thread initiated in a class called the ProcGUI class. The main task of this thread is to handle different kind of events(communication between user and program). The second thread is the simulation thread initiated in the Simulator class that handles the solving of the differential equations. To illustrate the program threads, a block scheme with the most important program interactions is given in Figure 1. All classes that are part of the

program are listed in Table 1. This table may be used as a quick reference during the reading to help the reader to keep track of the program structure.

Name of class	Main function
ProcGUI	Starts up the program, defines the GUI
AWTContainer	Contains global declarations of graphic objects
ImageCanvas	Draws the process image
RTView	Draws the scrolling graph
Bar	Draws bar diagrams
UpdContainer	Contains other global class declarations
InflowDialog1-6	Classes for changing flow rates
IncomingDialog	Class for changing concentrations of incoming water
SludgeDialog	Class for showing some sludge parameters
CarFlowDialog	Class for entering an external carbon flow
ProcessDialog	Class for changing process constants
VolumeDialog	Class for entering new zone volumes
SludgeAgeDialog	Class for entering a desired value of the sludgeage
Iaw	Contains properties and equation solvers of a zone
IawSetta	Contains properties and equation solvers of a simple settler
Simulator	Starts up the actual simulation

Table 1: The code of the simulator consists of 16 classes. The main purpose of each class is described in the right column.
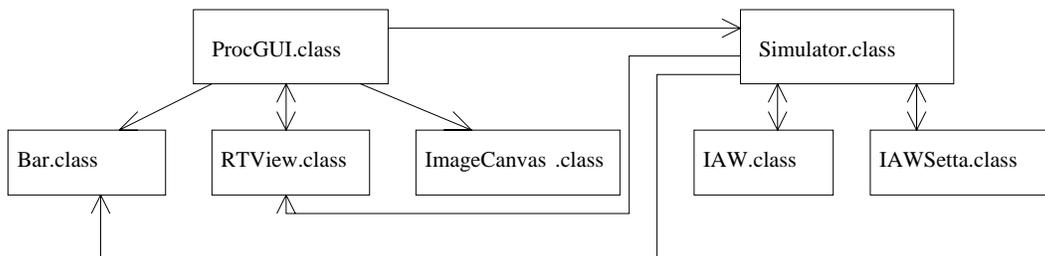
Figure 12: The main interactions in the program.

A.3   The ProcGUI Class

The class ProcGUI is a class that extends the predefined Java class Applet, in other words ProcGUI is the main program where all instances are initiated and where the simulation is started. The most significant member function is probably init(), which is the setup function for the whole applet. The function creates the layout of the users interface as well as instances of all necessary classes. init() also defines where to show all graphic objects in the interface. Apart from initializing the applet the most important thing about the ProcGUI class is its *action(Event e, object what)*, *handleEvent(Event e)* and *mouseDown(Event e, int x, int y)* functions. A pretty

good definition of what these functions do is that they take care of what should be done when for instance a certain button is pressed or a scrollbar is moved in the users interface, in other words they communicate with other parts of the program. The difference between these functions may be a little subtle, a press on a button or making a choice causes a call to the *action* function, while moving a scrollbar calls the *handleEvent*. The *mouseDown* function is (as the reader might have guessed) called when a mouse click is registrated in the interface. A small part of the *action* function can be seen below to illustrate what these functions look like.

```
public boolean action (Event e, Object what) {

    if(e.id == Event.ACTION_EVENT) {

      if (e.target == AWT.START) {

        try {
          simulation.Simulate();
        }
        catch(Exception fe){}

      }

      if (e.target == AWT.STOPP) {
        simulation.stop();
        System.exit(0);
      }
```

The small part of code seen above should be sufficient to get an idea of how the functions that handle events work. When a button or equivalent is pressed an instance of the class Event is created and the *action* function is called. In the function, different things are to be done depending on what button has been pressed, which is stored in the field target in the Event class. The small example above shows the cases where start and stop buttons are pressed. It might at this point be a bit difficult to understand the function calls, but simulation is an instance of the class Simulator, and its start and stop functions simply starts and stops the simulation. There is quite a large number of different events of this type that can occur due to the users demands during simulation, and to list them all here would probably not serve any purpose. A short examination of the code in combination with the following subsections of this thesis should give the interested reader a good view of what happens in different cases.

A.4   The AWTContainer Class

This class is a rather small class that contains declarations of graphic instances such as buttons, choices and textfields but also declarations of the classes for the scrolling graph (RTView), the bar diagrams (Bar) and the process image (ImageCanvas). To have these declarations in a separate class is advantageous because when the instance of the AWTContainer is declared in several of the other classes the declarations in it will be global. The class AWTContainer must be global since member functions of

RTView, Bar and ImageCanvas must be reached from a number of the other classes to handle different kinds of updatings of the users interface.

A.5   The ImageCanvas Class

The class ImageCanvas is the class that draws the process image. Since the process image is not completely static this class contains a few functions that update the image. The most important function of the class is the function *paint()* where the image is drawn. The picture has some parts that are always drawn, and some parts that are drawn according to different modes of the program. The static parts are the basin zones, the settler, the pipes, the valve pictures and all other clickable fields except for the fields where the alternatives excess sludge flow control or sludge age control are chosen. In the latter cases the active field is painted with dark grey color and the inactive with light grey. If sludge flow control is chosen, the value of the fixed flow is painted next to the valve, and equivalently when sludge age control is chosen, the fixed value of the age is painted. These features are simply handled by the function *setexFlowmode(int val)* which is shown below.

```
public void setexFlowmode(int val){
    if(val==1)exFlowControl=true;
    if(val==0){
     exFlowControl=false;
    }
    if(val==1)
      repaint();
  }
```

The function above is called from ProcGUI when the excess flow mode is changed. The integer *val* tells the function which mode that has been chosen by a click in either of the mentioned fields, and the boolean variable *exFlowControl* is set to tell the *paint()* function how to repaint the picture. The fact that the picture is repainted only when val=1 is due to the fact that when the simulator is changed to the sludge age mode, the user must enter a sludge age before the repaint. Therefore the call to *paint()* when val=0 is done in the function *setSludgeage(double value)* that is called from the SludgeAgeDialog frame class when the wanted sludge age has been entered. Of course, there is also a *setexFlow(double flow)* to change the sludge excess flow in the picture when the flow mode is active and the user changes the flow rate.

The function *setCarbonmode(int val)* works in a way similar to the function described above. In this function a boolean variable named *manualCarbonmode* tells *paint()* whether the program is running in manual or auto (regulator) mode for adjusting the flow rate of the external carbon source. If the simulation is chosen to be in manual mode the actual carbon flow is printed next to the carbon valve, if the mode is auto *paint()* simply draws "Auto mode" in the same place. A function that is also used for changing the carbon flow in the picture is *changeCarflow(double flode)* which is called when the value of the manual carbon flow is changed in simulation. This function is also called from the Simulator class to set the right carbon flow value when the

simulation mode is changed from auto to manual.

In the process image there is also five other clickable pump symbols. When the user click one of these, an applet dialog box where it is possible to change the corresponding flow rate appears. After a value has been entered, the process image is repainted and the new flow value is printed to the right of its pump symbol. This is done by the *changeInflow(double flode, int pumpnr)* function that is called from the InflowDialog1-6 frame classes. A dynamic feature that is coupled to each pump symbol is that when there is a flow through the valve, the color of the pump is green, while an inactive pump (where the flow is zero) is painted yellow.

There are two more "dynamic" features in the process image. The first one is the painting of bubbles and air valve in a zone when the oxygen reference value for that specific zone is larger than zero. This is done with two functions *setAeration(int zone, double value)* and *drawAeration(Graphics g, int x, int y)*. The first of these functions is called from ProcGUI when one of the ten oxygen reference values has been changed. The parameter *zone* tells the ImageCanvas in which zone the change has occurred, and the parameter *value* is simply the new reference value. If the new reference value is larger than zero the bubbles and valve is to be drawn in that zone. In the following repaint, the *drawAeration(Graphics g, int x, int y)* simply draws the "bubble picture" in all the zones with reference value larger than zero. The last dynamic feature is the possibility to click on a zone with the mouse, which marks the zone and the bars in the bar diagram that corresponds to that zone. This is achieved by having an integer which tells *paint()* where to draw the marks. If a zone is clicked, the integer changes and the process image is repainted.

There are also two more clickable fields in the process image, one to change the concentration of incoming substances and the other to change the process constants, but these fields are always unchanged when repainting is done.

A.6   The RTView class

RTView is the class that creates and updates the scrolling graph. In the graph, two concentrations of different compounds can be plotted at the same time. The most characteristic function of the class is *paint()*, which simply draws the axes and the graph itself (i.e lines and background). The scrolling is realized by copying the graph area, move it one pixel to the left and finally draw a line between the endpoints of the existing lines to the new points.

The class also contains functions to get different scalings, this because of the wide range of variation in substance concentrations. The change in scaling is done by the function *setLimit (int l1, int l2)* which is called from ProcGUI when one of the substance choices are changed. The parameters *l1* and *l2* are the maximal values for the two compounds chosen (The maximal value for each substance is specified in ProcGUI). The function *setLimit (int l1, int l2)* just set the limit values, the actual scaling of axes and drawings is done by another function, namely *scalePoint (double p, int maxy, int limit)*. This function is called by RTView:s own *paint()* each time

new points are appended to the graph.

The last function of big importance in the class is the function *append (double point1, double point2)*. *point1* and *point1* are the values of the points that are to be drawn, when these values have been read. *append (double point1, double point2)* simply repaint the graph. The calls to *append (double point1, double point2)* are made in the Simulator class, which sends the right values due to function calls to that class too when choices of basin zone and substances to plot are made. More on this subject when the class Simulator is treated.

A.7 The Bar Class

When simulation starts, three instances of the class Bar are created in ProcGUI. The task of the class is to plot bar diagrams of the concentrations of one substance in incoming water, the basin zones and the recirculated water. The substance to be plotted is chosen with a choice button next to each plot. When a choice is made a function call is made from ProcGUI to the Simulator, to get the Simulator to pass on the right values. The class is a rather simple one in the meaning that it only has two member functions. First there is the *append(double [] values)* that is shown below.

```
public void append(double values[]){
    int i;
    if(values[12]>3) k=10.0;
    if(values[12]==5) k=3.0;
    if(values[12]==7) k=3.0;
    if((values[12]==3)) k=0.005;
    if((values[12]==1)) k=0.1;
    if (values[12]==11) k=1;
    if (values[12]==2)  k=1;
    if((values[12]==0)) k=0.01;

    for (i=0;i<12;i++){
      v[i]=values[i];
      if ( (""+v[i]).length() == 1 ) {
        barvalue[i]=(""+v[i]+".000");
      }
      else {
        if(v[i]>=0.01)
          barvalue[i]=(""+v[i]+"000");
        else
          barvalue[i]=("0.0000");
      }
    }
    repaint();
  }
```

The function above is called from the Simulator with the parameter values which contains the concentrations that are to be plotted. Next to the bars the numerical

values of the concentrations are drawn, therefore the casts of the values to strings are done. The parameter k is a scaling factor depending on what substance the user want to plot. The last thing in the function is a call to the other member function, *paint()*, that handles the plotting.

A.8   The UpdContainer Class

UpdContainer is a class that, like the AWTContainer, contains declarations of global classes. The classes declared here have a lot in common, both in their structure and in what they do. The classes handle the communication between the users interface and the simulations, they all extend the superclass Frame and define applet dialog boxes when different fields are clicked in the process image. The 12 defined classes will be discussed in the following subsections.

A.9   The CarFlowDialog Class

CarFlowDialog is a class for a dialog box where a new flow of external carbon can be entered. When the carbon flow pump symbol in the process image is pressed, this dialog box, which contains one textfield and two buttons "ok" and "cancel", is shown. The most important function is the event handler *action (Event e, Object what)*, which is called when the user pushes one of the buttons. The code for this function is shown below.

```
public boolean action (Event e, Object what) {

    if(e.id == Event.ACTION_EVENT) {
      if (e.target == OK) {
        try{
          flode = new Double(flow.getText().trim()).doubleValue();
        }catch(NumberFormatException a) {flow.setText("Incorrect input");}
        newCarFlow=true;

        AWT.process_image.changeCarflow(flode);//update processimage
        this.hide();
      }
      if (e.target == CANCEL) {
        this.hide();
      }
      return true;
    }
    return(false);
  }
```

When the "ok" button is pushed this function reads the new manual carbon flow from the textfield and converts it to a double. When this is done, the boolean variable *newCarFlow* is set to true and the *changeCarflow(double flow)* of the class Image-Canvas is called with the new flow to get a correct process image. Besides *action*

(Event e, Object what), there are three more functions in the class, *checkUpdate()*, *getnewCarFlow()* and *setText(double value)*. *checkUpdate()* returns the boolean variable *newCarFlow* and is called from the Simulator to see if a new carbon flow has been set. If this is the case Simulator calls *getnewCarFlow()* that sets the boolean variable to false and returns the new flow to the simulations. *setText(double value)* is called from the Simulator when the carbon flow mode is changed from auto to manual, this to set the right flow in the textfield.

A.10   The ProcessDialog Class

The ProcessDialog has the same function as the class in the previous subsection. It contains a dialog box shown when the field "Process constants" in the process image is clicked. In this dialog box the user can enter new biochemical and stochiometrical constants at the temperature 20C for the processes of the IAWQ-model, and the dialog box therefore contains 19 textfields for these constants. There are also a textfield where the user may enter the actual temperature, and 19 more textfields that jaust shows the process parameters at the actual temperature.

The class has three member functions, function *action (Event e, Object what)* that reads and converts the textfields and estimates the process constants at the actual temperature, and finally sets a boolean variable true. The function *checkProcessUpdate()* that returns the boolean variable to the Simulator. The function *getProcessConstants()* resets the boolean variable and returns the array of new process constant to the Simulator when *checkProcessUpdate()* has returned the value true.

A.11   The InflowDialog1-6 Classes

The task of these classes is to tell the Simulator class when it is time to update the any of the six flow rates and and pass on the new values to the simulations. Three textfields are defined in each these frames, one where the user may enter the average flowrate, one for the magnitude of the flowrate and one for the period time. There are three functions in each these classes. The most significant one is the function *action(Event E, Object what)*. This function is called when the user pushes the *ok* or the *cancel* buttons of the frame. If the *ok* button is pushed a boolean variable *newInflow* is set to true to registrate that a new flow has been set. The *checkUpdate()* function returns *updflow* and is called continuously from the Simulator to check if a new flow value has been entered. If this is the case the Simulator class calls the *getnewFlow ()* that sets *newInflow* to false and returns the new flow.

A.12   The IncomingDialog Class

The updating of the incoming concentrations works in the same way as the updatings of the flowrates. The class contains 12 textfields where the user may enter new values of the concentrations of the different substances in the incoming water. When a *ok* button is pushed a boolean variable *IncomingUpdate* is set to true. Simulator calls the *checkIncomingUpdate()* that returns the boolean variable, and if the boolean

variable is true, Simulator also calls *getIncoming ()* that sets *updIncoming* to false
and returns the array of new incoming concentrations to the simulations.

A.13   The SludgeDialog Class

Class SludgeDialog is simply a dialog box that is shown when the "Sludge Properties"
field is clicked in the process image. The class contains three textfields, and it has
only one significant member function, *setSludgeText(double age, double conc, double
flow)*. This function is called continuously from Simulator and it writes the sludge
age, the sludge concentration and the excess sludge flow to the textfields. It also calls
the *setexFlow(double flow)* of the class ImageCanavas since a correct process image
then can be drawn when the excess sludge flow mode is changed from age control to
direct flow control.

A.14   The SludegAgeDialog Class

This class is structurally similar to the two classes before. In the class there is a
textfield for giving a new desired sludge age when the simulator is running in the age
control mode.

Four member functions are incorporated, *checkSludgeUpdate()* that returns a boolean
variable to the Simulator, *getnewAge()* that resets the boolean variable and returns
the new sludge age to the Simulator, *setText(double newsludge)* that writes the actual
sludge age to the textfield when the user chooses the age mode and finally there is
action (Event e, Object what) for reading and converting of the textfields.

A.15   The VolumeDialog class

This class is also very similar to the classes above. It extends the superclass Frame,
and when a certain field is clicked, a window where the user can enter new zone
volumes are shown. The class contains 11 textfields and a boolean variable to tell
Simulator when new volumes have been set. All together this class works exactly as
the previously described ProcessDialog class.

A.16   The Iaw Class

Ten instances of this class are created in the Simulator class, and a proper description
is that an instance of Iaw is simply a basin zone with all its characteristics. The most
important data fields needed to solve the differential equations of the IAWQ model
are shown below.

```
double [] inflow;
double [] state;
double [] previous;
double [] regler;
double [] outflow;
double Qin;
```

```
    double Qpre;
    double Qreg;
    double Qcar;
    double Car;
    double Air;
    double V;
```

The array *inflow* contains the substance concentrations of the influent water, *state* contains the actual concentrations in a basin, *previous* contains substance concentrations in the water coming from a previous zone, *regler* contains the concentrations in the excess sludge and *outflow* the concentrations in the water going out of the basin zone. Similarly, *Qin* is the external inflow to the basin zone, *Qpre* is the flow from the previous zone, *Qreg* is the flow of the water that is recycled from the settler and *Qcar* is the flow of external carbon source into a zone. Note that there is no datafield *Qout* since the outflow of the zone is calculated as the sum of the inflows in each timestep. The parameter *Car* is the COD of the water from the external carbon source, *Air* is the airflow into the zone and *V* is simply the volume of the zone. A few things here need some further explanation: First, the reason for having an array *regler* with substance concentrations of the recirculated sludge in all zones instead of just the first where it is needed is just for program simplicity. While all flows *Qreg* except for the one in the first zone is set to zero this will not affect the simulations at all. Second, in this version of the program all carbon flows, *Qcar*, except for the one in the sixth zone is always zero, i.e the sixth zone is the only one where carbon can be added, but due to the object oriented structure of the program, a user just needs to change a few lines of code in the Simulator class to add carbon to an arbitrary zone. Third, in the first zone the flow *Qpre* and concentration array *previous* is used when a predenitrifying system is simulated, that is when water is recirculated from the last zone to the first. In the default version *Qpre* of the first zone is set to zero, i.e a postdenitrifying system is simulated. During simulation, the user can easily change the process to be predenitrifying.

Iaw has four member functions, they will here be discussed in detail. To begin with the simplest one, the function *setNewConst()*, is called when stochiometric or biologic constants has been changed in the users interface. This function calls the *getProcessConstants()* of the class ProcessDialog to get an array of the new process constants and sets the new values in the simulation. *setNewConst()* also calls another member function in its own class, *CalcHelpConst()*, that calculates some help constant to make the solving of the differential equations a bit faster. The most important functions in the class are those for solving the differential equations, *newcalcIawqRungeKutta()* and *deltaF(double newstate[], double ystate[])*. The *newcalcIawqRungeKutta()* takes one time step (0.01h) in the differential equations for all the substances with the fourth order RungeKutta method. The code of the implemented Runge-Kutta method is shown below.

```
public void newcalcIawqRungeKutta(){
    int i = 0;
    if(upd.newProcConst.checkProcessUpdate()==true)
      setNewConst();
```

```
      deltaF(k1, state); //computes R-K parameters
    for(i=0; i<11; i++) {
      y_between[i] = state[i] + (__deltaT/2)*k1[i];
      if (y_between[i] < 0.0) y_between[i] = 0.0;
    }
    y_between[_SS] = state[_SS] + (__deltaT/2)*k1[_SS];
    if (y_between[_SS] < 0.0) y_between[_SS] = 0.0;


    deltaF(k2, y_between);
    for(i=0; i<11; i++) {
      y_between[i] = state[i] + (__deltaT/2)*k2[i];
    if (y_between[i] < 0.0) y_between[i] = 0.0;
    }
    y_between[_SS] = state[_SS] + (__deltaT/2)*k2[_SS];
    if (y_between[_SS] < 0.0) y_between[_SS] = 0.0;


    deltaF(k3, y_between);
    for(i=0; i<11; i++) {
      y_between[i] = state[i] +__deltaT*k3[i];
      if (y_between[i] < 0.0) y_between[i] = 0.0;
    }
    y_between[_SS] = state[_SS] + (__deltaT/2)*k3[_SS];
    if (y_between[_SS] < 0.0) y_between[_SS] = 0.0;


    deltaF(k4, y_between);

    for(i=0; i<11; i++) {   //computes new states
      state[i] = state[i] + (__deltaT/3)*(0.5*k1[i] + k2[i] + k3[i] + 0.5*k4[i]);
      if (state[i] < 0.0) state[i] = 0.0;
    }


  }
```

As seen, the implementation is very similar to the given algorithm. The most significant things in the code are the calls to the last member function of the class, *deltaF(double newstate[], double ystate[])*. This function is in principle equivalent to the $f_i(t, y_1, ..., y_N)$ function described in the algorithm, i.e it computes the process rates (mass balances as well as biological and stochiometrical process rates). The function is called with the parameter *ystate[])* which answers to $y_1, ..., y_N$ in the algorithm description. The values of the Runge-Kutta coefficients are stored in the array *newstate[]*.

A.17 The IawSetta Class

The IawSetta Class is a class for solving the differential equations of a simple settler, i.e the reader may consider an instance of this class as a settler. The class has got data fields very similar to the Iaw class, there are two concentration arrays *inflow* and *state*, and two flows *Qin* (inflow to the settler) and *Qreg* (flow taken out in the bottom of the settler). The IawSetta class has only one member function, *newcal-*

cRungeKutta() which is called from class Simulator. The function solves the mass balances for the settler in each time step, its code will not be shown here while it is pretty similar to the function with the same name in the previous class. The difference between these is that the time derivatives here are calculated directly in the function while the differential equations are much simpler due to less complicated processrates.

A.18   The Simulator Class

This class is the head class of the actual simulation. Here, ten instances of class Iaw and one instance of the class IawSetta are created and all datafields are initialized in the constructor. The class implements runnable, which means that it has a function *run()* where a separate thread of the program is running. The main task of this thread is to iterate and call the differential equation solvers of Iaw and IawSetta to calculate the new states of the model. In the function there are also calls to the append() functions of classes RTView and Bar to update the graphics. The last important thing in this function is the function calls to the classes declared in UpdContainer to get values of different constants when these have been changed.

In class Simulator there are also two regulator functions implemented, *CarbonRST()* and *doPID()*. *CarbonRST()* is a model based LQG regulator for the flow of the external carbonsource, and *doPID()* contains ten PID regulators for air flow rates, one for each zone. A few other functions to change reference values and PID parameters are also connected with the regulators. These functions are all called from the ProcGUI class.

Besides the functions described above there are a few more functions used for communication between Simulator and ProcGUI. The purpose of these functions are to tell Simulator which values to send to the RTView and Bar classes for updatings of the interface, and to tell Simulator which program modes (sludge age control, external carbon source etc) that should be active.

B Process Parameters in the Simulator

Table 2 shows explanations of the various process parameters and their typical values.

Symbol	Simulator	Value	Explanation
Y_A	Ya	0.24	yield for autotrophic biomass
Y_H	Yh	0.67	yield for heterotrophic biomass
i_{XB}	Ixb	0.086 g N(g COD)$^{-1}$	mass of nitrogen per mass of COD in biomass
i_{XP}	Ixp	0.06 g N(g COD)$^{-1}$	mass of nitrogen per mass of COD in products from biomass in endogenous mass
f_P	fp	0.08	fraction of biomass yielding particulate products
$\hat{\mu}_A$	mya	0.033 h^{-1}	maximum specific growth rate for autotrophic biomass
$\hat{\mu}_H$	myh	0.25 h^{-1}	maximum specific growth rate for heterotrophic biomass
K_S	Ks	20 g COD m^{-3}	half saturation coefficient for heterotrophic biomass
$K_{O,H}$	Koh	0.2 g O$_2$ m^{-3}	oxygen half saturation coefficient for heterotrophic biomass
K_{NO}	Kno	0.5 g NO$_3$-N m^{-3}	nitrate half saturation coefficient for denitrifying heterotrophic biomass
K_{NH}	Knh	1.0 g NH$_3$-N m^{-3}	ammonium half saturation coefficient for autotrophic biomass
$K_{O,A}$	Koa	0.4 g O$_2$ m^{-3}	oxygen half saturation coefficient for autotrophic biomass
b_A	Ba	0.0833 h^{-1}	decay rate coefficient for autotrophic biomass.
b_H	Bh	0.026 h^{-1}	decay rate coefficient for heterotrophic biomass
η_g	etag	0.8	correction factor for μ_H under anoxic conditions
η_h	etah	0.4	correction factor for hydrolysis under anoxic conditions
k_a	Ka	0.0033 m^3 COD (g h)$^{-1}$	ammonification rate
k_h	Kh	0.125 g COD (g COD h)$^{-1}$	maximum specific hydrolysis rate
K_X	Kx	0.03 g COD (g COD)$^{-1}$	half saturation coefficient for hydrolysis of slowly biodegradable substrate
$K_L a$			oxygen transfer function
$S_{O,sat}$		8.67 mg/l	saturated oxygen concentration

Table 2: The parameter values for the implemented IAWQ model at 20°C.

C   Example Laboration

C.1   Introduction

Two versions of the simulator will be used in this lab:

- A pre-denitrification ASP with the possibility to add an external carbon source in zone 1.

- A post-denitrifying system. An external carbon source can be added in zone 7.

The JASS simulator

C.1   Flows and Volumes

The simulated process consists of a basin divided in 10 completely mixed zones and a settler. A schematic layout of the process is shown in Figure 13 (lower half).
The (default) configuration of the zones is given in Table 3.

Zone no.	Volume of zone (m^3)	Operation
1	235	Anoxic
2	235	Anoxic
3	235	Anoxic
4	235	Anoxic
5	110	Anoxic
6	110	Anoxic
7	470	Aerated
8	235	Aerated
9	235	Aerated
10	235	Aerated

Table 3: The basin consists of 10 zones, zones 1 to 6 are non-aerated (anoxic) and zones 7-10 are aerated.

Each zone is modeled with the IAWQ model no 1. The values of the parameters and concentrations in the model can be changed. The default values of some typical process constants are given in Appendix B. The default concentrations of different compounds in the influent wastewater are given in Table 4.
The nominal flow rates of the pumps are given in Table 5.

C.2   The Sedimentation Unit

The sedimentation unit is assumed ideal, and it is modeled with the following simple mass balance

$$S_{inset} = S_{eff} = S_{rec}$$

$$X_{inset}Q_{inset} = X_{rec}(Q_{rec} + Q_w)$$

Standard abbrev.	Abbrev. in simulator	Variable	Conc. (mg/l)
$X_{B,H}$	Xbh	Heterotrophic biomass	0
$X_{B,A}$	Xba	Autotrophic biomass	0
X_S	Xs	Slowly biodegradable substrate	82
X_I	Xi	Part. inert org. matter	43
$X_{N,D}$	Xnd	Part. biodegradable org. nitrogen	5.8
$S_{N,H}$	Snh	NH_4-NH_3 nitrogen	23.5
$S_{N,D}$	Snd	Soluble biodegradable org. nitrogen	2.1
$S_{N,O}$	Sno	Nitrate and nitrite nitrogen	1
S_I	Si	Soluble inert org. matter	37
S_O	So	Oxygen	0.1
S_S	Ss	Readily biodegradable substrate	70

Table 4: Influent wastewater composition. The values approximately correspond to the composition of pre-sedimented wastewater at the main municipal plant in Uppsala

Flow	Rate [m^3/h]
Influent	250
Excess sludge	4
Return sludge	250
Internal recirculation	0
External carbon flow rate	0

Table 5: Nominal values of the flow rates in the the simulation model. If the value 0 is altered the pump will be marked yellow (off).

where subindex *inset* denotes influent components to the settler, X_{rec} is the concentration of particulate matter in the settler, Q_{rec} is the internal recirculation flow rate and Q_w is the excess sludge flow rate. Hence, the concentration of all dissolved components S are unaffected by the sedimentation dynamics and no suspended solids (X components) is assumed in the effluent.

Since no biomass is assumed in the effluent, the sludge age is then simply

$$\theta_s = \frac{V * X}{Q_w * X_{rec}} \quad [days] \tag{5}$$

where Q_w is the excess sludge flow rate (expressed in m^3/day), X is the mean sludge concentration in the aeration basin, X_{rec} is the concentration of the sedimented sludge and V is the volume of the bioreactor. Note that it is often of interest to calculate the *aerobic sludge age*, given as

$$\theta_{s,aerobic} = \theta_s \frac{V_{aerobic}}{V} \tag{6}$$

C.3   The Graphical User Interface (GUI)

The simulator is operated with a GUI where also the status of the plant is shown in real time. As seen from Figure 13, the plant is schematically drawn in the GUI. The

simulated flow rates of the pumps are changed by clicking on the pump symbol, then clicking on the "value box" and altering the desired flow rate in the window that appears. Note that the cursor must be located inside the value box before a value can be entered. To the right in the GUI, the simulation time is presented during the simulation. Note, that some changes may take hours or days before they are fully observable.

Important process parameters can also be shown in the GUI. For example, if the field *Sludge Properties* is clicked, a window that shows the sludge age, the sludge concentration and the excess sludge flow appears. The sludge concentration is here defined as the average concentration of particulate compounds in the ten zones. To further improve the presentation of the status of the plant, three bar diagrams are used. Each diagram shows the concentration of a selectable compound in incoming water, all zones and the settler. In these diagrams, the numerical values of the concentrations are also shown. In the GUI an ordinary graph is also used. Here, the concentrations of two compounds in a specific zone can be plotted versus time.
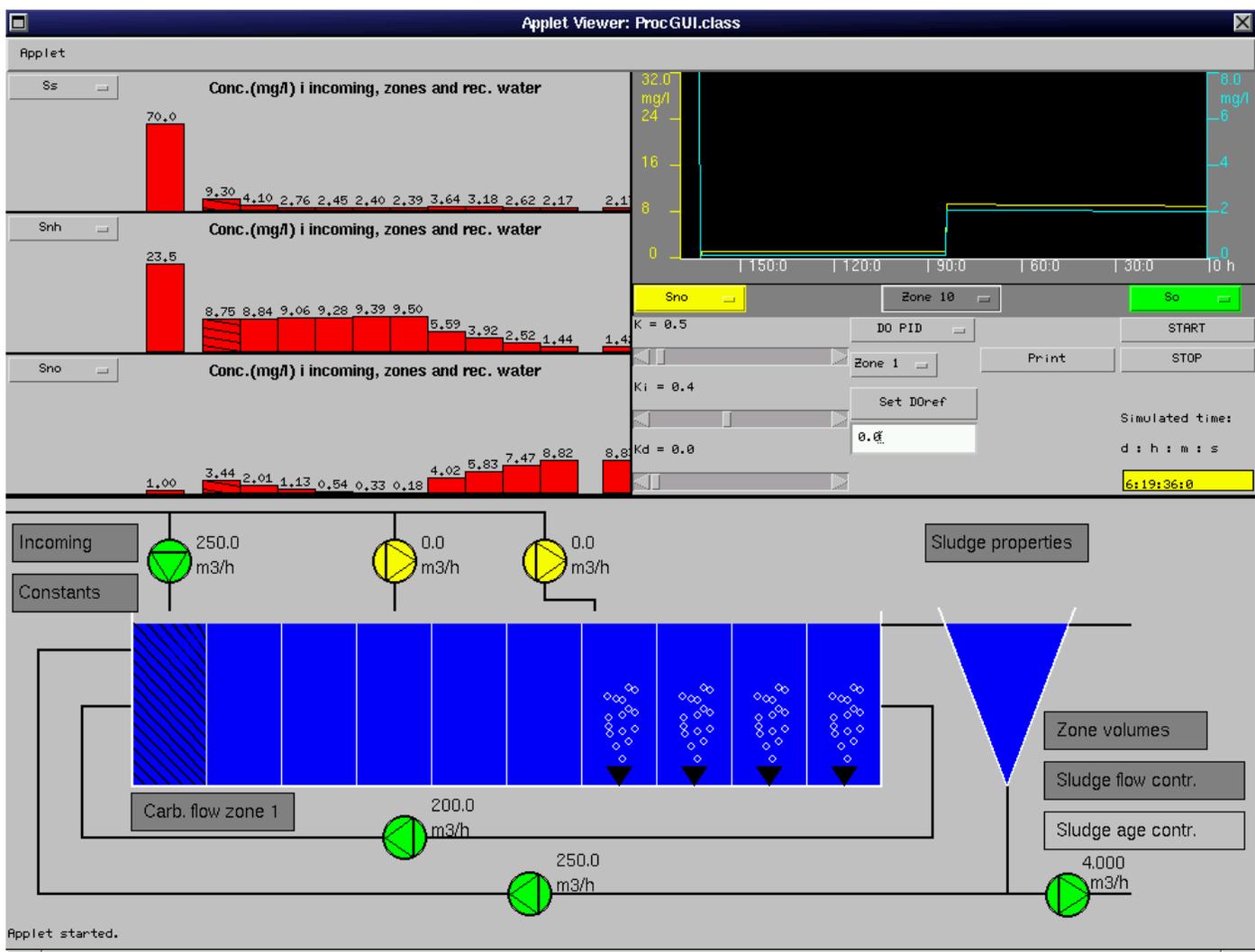

Figure 13: The graphical user interface in the simulator. The pre-denitrifying version.

C.4   Exercises - A Pre-Denitrification Process

You will here study the effect on the process when changing different process parameters. Some supporting calculations will also be done.

Exercise 1, Preparations

Start Netscape (or Explorer) and type the following URL address (or load a bookmarked version if possible).

www.syscon.uu.se/~psa/lab1e/ProcGUI.html

Start the simulator and check that:

- The three staple diagrams show (*air flow*), ammonium (*Snh*) and nitrate (*Sno*)

- Incoming flow = 250 m^3/h. Internal recirculation flow = 0 m^3/h. Excess sludge flow = 4 m^3/h.

- The DO level is around 2 mg/l in the last four zones and 0 in the others. This can be done by selecting *So* in one of the staple diagrams.

Let the simulator reach steady state values (this may take several days) and check:

- Effluent ammonia [1]=...................... mg/l

- Effluent nitrate=...................... mg/l

- Nitrate in last anoxic zone=...................... mg/l

Exercise 2, Effect of internal recirculation rate

An increase of the internal recirculation rate reduces the effluent nitrate concentration if the denitrification is complete or almost complete.

Assume we have complete denitrification ($S_{N,O} = 0$ in the last anoxic zone), influent flow is Q, internal recirculation is rQ, recirculated sludge is sQ. To derive the effluent nitrate concentration from the last aerated zone, the following mass balance is valid

$$Q \ S_{N,H,in} = (Q + rQ + sQ)S_{N,O,eff} \qquad (7)$$

which can be written as

$$S_{N,O,eff} = \frac{1}{1 + r + s} S_{N,H,in} \qquad (8)$$

---

[1] The effluent concentration of a soluble compound is the same as the concentration in the last zone

If $s = 1$ and $r = 0$ in the beginning and then the internal recirculation is increased to $r = 1$, by how many percent is the effluent nitrate reduced?

ANSWER:

Increase the internal recirculation to 250 m^3/h ($r = 1$). Write down the values of the nitrate concentration before and after the change in internal recirculation. Comment on the result and compare with (8).

ANSWER:

Exercise 3, Effect of incoming biodegradable substrate
Now, set the recirculation back to zero and change the concentration of easily degradable carbon S_S in the influent water to 25 mg/l. Wait for the process to reach steady state.

What happens with the ammonium and nitrate levels in the last anoxic zone and in the effluent water? Write down your values and compare these with Exercise 1. What microbiological reaction is primarily affected by the lack of carbon in influent water?

ANSWER:

What happens in the last anoxic zone and in the effluent water if you now increase the internal recirculation flow rate to 250 m^3/h and why?

ANSWER:

Exercise 4, Adding an external carbon source

An easy way of getting a better working denitrification when there is a lack of easily degradable organic matter in the anoxic zones is simply to add some ethanol in the first anoxic zone. Do this by clicking in the field *Carbon Zone 1* and enter a ethanol flow rate (m^3/h) in the window that appears. Try to make the nitrogen removal as good as possible without adding more carbon than is needed to get complete denitrification in the last anoxic zone (a good start value of the ethanol flow rate may be 0.01 (m^3/h). Write down your values of ammonium and nitrate in effluent water and in the last anoxic zone below. Comment on your results! Compare your results with Exercise 2.

ANSWER:

Exercise 5, Change of the DO

First of all, turn off the external carbon flow and set the concentration of *SS* back to 70 mg/l. Wait for the simulations to reach steady state and write down the values of: air flow rates (choose *airflow* in one of the bar diagrams), nitrate concentrations, and effluent ammonia. The dissolved oxygen concentration has a large impact on the activated sludge process. By default, it is set to 2 mg/l in all the five aerated zones. Change the DO set-point (the reference value of dissolved oxygen) to 0.5 mg/l in all these zones and study the effect on the ammonium and nitrate concentrations. How to do this is described in the section, *running the simulator*. Write down the values of: air flow rates (choose *airflow* in one of the bar diagrams), nitrate concentrations, and effluent ammonia after you change DO set-point. Comment on the results.

ANSWER:

Exercise 6, Increase of influent ammonia.

First, change all five (zone 6-10) oxygen reference values back to 2.0 mg/l. (Check that internal recirculation flow rate is 250 m^3/h and influent flow rate is 250 m^3/h).

When the process has reached steady state, write down the values of air flow rates, nitrate concentrations, and effluent ammonia. Then increase the ammonium concentration by 10 mg/l (to 33.5 mg/l) and study the effect in ammonium and nitrate concentrations, and airflow rates. Try to explain what you see.

ANSWER:

Exercise 7, Tuning of DO controller

First, change the value of influent ammonia back to 23.5 mg/l. Study how the DO controller in zone 7 is tuned by changing the set point (for example to 3 mg/l). Choose So in the time graph (upper right corner) and air flow in the bar diagrams. Try to improve the performance by adjusting the PID parameters. This is done by moving the scrollbars under the menu bar *DO PID*. Remember to choose the right zone before changing your parameters. What parameters did you select?

ANSWER:

Exercise 8. Increase of excess sludge flow

The biomass in the bioreactor affects the whole process. What usually is most critical to maintain is the nitrification process, why? Change the excess sludge flow rate to 40 l/h and study the ammonium conc. nitrate conc. and air flow rates. Explain what you see.

ANSWER:

Now you have made a wash-out! Be thankful it was only done on a simulator.

C.5   Exercises -Post Denitrification Process

First of all, load a version of the simulator that simulates the post denitrification process for the same influent water as before. This simulator is found on the URL

http://www.syscon.uu.se/ psa/lab2e/ProcGUI.html

(may be bookmarked on your Communicator). Start the simulations in the same way as before.

Exercise 9, Preparations
After you have started the simulator, let it reach steady state values (this may take 5-10 days) and check:

- Effluent ammonia=....................... mg/l

- Effluent nitrate=....................... mg/l

- Substrate Ss i zone 7....................... mg/l

What can you say about the levels of ammonia and nitrate in the effluent water compared with Exercise 1. Explain the differences! What would you do to improve the nitrogen removal?

ANSWER:

Exercise 10, Adding an external carbon source
External carbon will now be added in order to improve the denitrification and reduce the nitrate level in the anoxic zones. This time the ethanol will be added in zone 7 (first anoxic zone). To add ethanol in zone 7, click on the pump symbol above the zone and enter a value of the flow rate in the window that appears. A reasonable flow rate to start with would be 0.01 m^3/h. Try to make the nitrate removal work as good as possible without an unnecessary high ethanol flow rate. Write down your final values of ammonia and nitrate in the effluent water, and also your choice of external carbon flow rate.

ANSWER:

Exercise 11, Control of external carbon source
Of course, in the long run manual control of the external carbon source is not a very cost efficient way of getting a good nitrate removal. The purpose with this exercise is to show you that it is possible to design a controller that controls the ethanol flow rate. Go to the menu bar *DO PID* and choose *Cregulator* instead. The panel with oxygen PID regulators is now replaced with a external carbon source regulator. To switch to automatic carbon control, push the *auto* button. To give a new reference value of the nitrate level in the effluent water, write your new value in the textfield to the right of the text *No3 ref zone 10:* and push the *set* button. By pushing the button *manual*, the control of the ethanol is switched back to the manual mode. Now, use automatic control of the ethanol and choose the desired value of nitrate in the effluent water to 1 mg/l. Compare the automatic control in this exercise with the manual control of Exercise 10. What advantages can you see?

ANSWER:

D   How to Change the Simulator Setup

This appendix serves as a quick reference of how to change the default setup of the
simulator. Table 6 shows the most usual changes and how to perform them.

Setup Change	How to perform it
Default zone volumes	Change the variables vz1-10 in the file Simulator.java, also change the default values in VolumeDialog.java to get right values in the textfields.
Default concentrations in influent water	These are changed by changing the values in the array initarr1 in the file Simulator.java. The order of the components is the same as in all the menu items.
Initial values for the concentrations in the zones.	These are changed by changing the values in the array initarr in the file Simulator.java. The order of the components is the same as in all the menu items.
Default flow rates	The average values of the inflow rates, the recirculated waterflow, the recirculated sludge flow rate and the excess sludge flow rate are changed by changing the variable medelv1-6. The corresponding flow magnitudes and period times are changed by the vaiables magn1-6 and freq1-6.
Default external carbon flow rate	Changed by giving a new initial value of the matrix element inmatris[15][7] in Simulator.java. The new value must also be entered in the file CarFlowDialog.java. To get the right carbon flow rate value from the start in the process image, the variable CarFlow in ImageCanvas.java must also be set to the new value.
Default process parameters	These are changed in the file Iaw.java and in the file ProcessDialog.java.
Default oxygen reference values	These are stored in the array r in Simulator.java. The values must also be set in the array Aeration in the file ImageCanvas.java, this to get the bubble pictures right.
Default parameters in PID controllers for the oxygen inblow	The PID parameters are stored in the arrays K, KI and KD.
The placement of objects	All placement of objects (such as menu bars, the bar diagrams, the graph etc) is done in ProcGUI.java.
The scrolling graph	Everything that has to do with the looks of the scrolling graph (colors, time scale, scaling, etc) is changed in the file RTView.java.

Setup Change	How to perform it
Time step of the ODE:s	This is changed by changing the variable deltaT in the file Iaw.java. Observe that this change leads to that the timescale in the graph must be changed (done in RTView.java) and that some parameters in TimeCanvas.java should be changed to show the right simulated time. The controllers must also be changed.
The $K_L a$	$K_L a$ is changed by the parameter KLa in the file Iaw.java.

Table 6: How to change the default setup for the simulator

E   The Process Matrix

See (Jeppsson 1996*b*).

References

Åström, K. J. and B. Wittenmark (1990). *Computer-Controlled Systems*. Prentice-Hall International Editions.

Flanagan, D. (1990). *Java In A Nutshell*. O'Reilly.

Henze, M., C. P. L. Grady Jr., W. Gujer, G. v. R. Marais and T. Matsuo (1987). Activated sludge model no. 1. Scientific and Technical Report No. 1. IAWPRC, London.

Jeppsson, U. (1996*a*). A general description of the iawq activated sludge model no1. Technical report. Lund Institute of Technology.

Jeppsson, U. (1996*b*). Modelling Aspects of Wastewater treatment plants. PhD thesis. Department of Industrial Electrical Engineering and Automation, Lund Institute of Technology.

Lindberg, C. F. (1997). Control And Estimation Strategies Applied To The Activated Sludge Process. PhD thesis. Systems and Control Group, Uppsala University.

Ljung, L. and T. Glad (1991). *Modellbygge och simulering*. Studentlitteratur.

Luttmer, J. (1995). Design of a simulator for an activated sludge process. Master's thesis. Systems and Control Group, Uppsala University. UPTEC 95149E.

Söderström, T. and P. Stoica (1989). *System Identification*. Prentice-Hall International Editions.