CODEN:LUTEDX/(TEIE-5244)/1-126/(2006)

Industrial Electrical Engineering and Automation

BENCHMARKING the STAR Controller using Matlab

Josep Carrasco Martínez

Dept. of Industrial Electrical Engineering and Automation
Lund University

BENCHMARKING the STAR Controller using Matlab

A MSC THESIS PRESENTED AT THE DEPARTMENT OF **Industrial Electrical Engineering and Automation** (IEA) of LTH at ***LUND UNIVERSITY***

Author:	**Josep Carrasco Martínez**
For the Degree of:	**Master of Science**
Supervisor:	**Dr Christian Rosen**
Examiner:	**Dr Ulf Jeppsson**
Date:	**September 2006**

Abstract

Along the years there have been large advances in Wastewater Treatment Plants (WWTP since now), from the basic treatment until applying the last technologies in automatic control systems, concentration measurements and mathematical modelling of wastewater treatments. About the last technologies it has to be stood out the fast advanced of computer systems, that let not only implement the most complex models of WWTP to check the evolution of the system in a short time, but also the possibility of integrating more complex controls, and it has to be stood the advanced of electronic systems too, that let apply the last sensor technologies, increasing the reliability and the device life in the abrasive ambient as it is the waste water.

The difficulty in WWTP is the biological processes that take part within the vessels. Principally the growth of biological microorganisms that are the responsible of removing the organic matter dissolved in the wastewater. The main organic matter that is removed from wastewater is the carbon, the nitrogen and the phosphorus, although the phosphorus can be removed sometimes by chemistry processes, which are faster than biological processes. The problem in biological processes is its slow dynamic and the external perturbations, which can affect considerably the growth of biological microorganisms.

Currently, the companies involved in WWTP offer automation tools and packages that let to apply automation techniques in WWTP easily. One of the most important devices offered is the controller, which will try to lead the progress of the wastewater by the right values and to ensure the process is getting the goals. It means that the controller is going to control the biological processes that take part in wastewater treatment applying the different control strategies programmed.

Here there are three concepts all together; the controller, the biological process and the control strategy. First, the control strategy is tested using a model of the WWTP. After, it is implemented in the controller to apply it to the real WWTP to control the biological process. The problem appears when the control strategy is applied to the real process. It is impossible to predict all that is going to happen in WWTP, because there are unpredictable external disturbances (rain, storms, temperature, dry weather, snow, ice, etc) that can affect the biological processes. The biological processes are sensible to the external changes of the weather. It means that a control strategy tested in a simulation within ideal conditions can not be the suitable.

The problem rises when the control strategy is not the suitable. The biological processes dynamic is really slow, a time constant of days, even weeks some times. Hence, it is possible to know when a control strategy is wrong after days or weeks, and then, is too late to try to control the process and one more week is needed to test another control strategy. All this time checking the control strategy means to spend a lot of resources.

There is one possible solution that would save a lot time and would let to test more control strategies using less resources. The solution consists in building a complete model of the WWTP that is going to be controlled (it would be necessary a particular model for each WWTP), and to simulate it in real time at the same time as the evolution of the real WWTP. It would let the controller to have two kinds of data, the data

measured in the WWTP and the data from the simulation of the model. With these data the controller or the operator can compare and test the control strategies, so that if the data measured in the WWTP are far from the data calculated in the simulation, it is possible to guess that there is something wrong with the control strategy.

Of course, this solution forces to implement a perfect model of the WWTP because the controller is comparing the data of the simulation of the model and the data of the WWTP, considering that the data calculated with the model are always right.

Preface

Before has been introduced one of the current problems in WWTP, a problem responsible of spending a lot of resources and time when testing control strategies to improve the efficiency in WWTP.

The realization of the present thesis is one of the results of a task group work of the "krüger" (in Copenhagen, Denmark, from "Veolia Water Systems" group, http://www.kruger.dk) in collaboration with the department of "Industrial Electrical Engineering and Automation" (IEA, at Lund University, Sweden). There has been created a research group with workers of the company and researchers of the IEA department, which purposes are to improve modelling WWTP and the control strategies minimizing the necessary resources.

Acknowledgements:

At last, the finalization of this thesis was possible not only to the work realized by the author, but also to the IEA department and the krüger company. There were involved more people, directly and indirectly related with the thesis.

I would like to express my special gratitude to all the people that have helped me in one or another way finishing the thesis.

Thank you to Christian Rosen (research associated of the IEA department), he has followed and supervised the progress of the thesis, despite of all the work that he had. He is part of the research group in collaboration with the company "Veolia Water Systems". It was inestimable his help to introduce me in the topic of waste water treatment; it was absolutely new for me at the beginning of the thesis.

Thank you to Ulf Jeppsson, the responsible of the IEA department, who has supervised the thesis too, specially the part of modelling batch process. His work about modelling settling and clarification and the model programmed in Matlab by Krist V. Gernaey (CAPEC, Dept. of Chemical Engineering, DTU, Denmark) and him, are the key model implemented in this thesis.

Thank you to Eric Login from "Veolia Water System" (krüger, in Copenhagen, Denmark). The communication of the main application was developed and discussed in collaboration with him.

Thank you to Sergey Litvinov too. I never met him personally, but his help was essential to resolve the creation of XML files with Matlab.

Finally, my gratitude to Dalius Misiunas, Fredrick Roos, Carlos Álbarez, Daniel Aguado, Jon Bolmstedt, Gunnar Lindstedt and all the department of "Industrial Electrical Engineering and Automation" (IEA, LTH, Lund, Sweden), and specially to *Lunds Universitet* and the *Polytechnic University of Valencia* (Universidad Politécnica de Valencia –UPV-, Valencia, Spain) who gave me the possibility of taking part in this foreign experience and realise the final degree thesis abroad.

Contents

PART I

TITLE: *Introduction*

CHAPTER 1

Introduction

This chapter is a brief sight of the thesis; motivations, contributions, structure and objectives.
After reading this chapter the reader knows the contents of each point in this thesis, so it's important to know how to read it and where is each topic studied.

1.1 MOTIVATION

The treatment of the waste water is a complex process, process compound of several biological processes sensitive to the external disturbances, as can be the storm, the rain, the snow, the ice; all the climatological changes can affect the wastewater treatment process. The sensitive biological processes and the disturbances of the weather make difficult the control of the wastewater treatment process.

Through the new technologies the most avant-garde companies have tried to solve the problem of the complex control needed and the existing disturbances within the process. With the new sensors it is easier to measure the state of the process and try to design suitable control strategies. The possibilities of the new computers let this and more, but there is still one restriction: the dynamic of the biological processes.

With the new advances of the techniques sometimes it is forgot that the new techniques cant change the process, they are only developed to control it, to get a stable and controllable process. The dynamic of some biological process are so slow, it means that the results of an action control will be observed a long time after applying it. This difficult the control of the process more than it seems at the beginning, because when the controller tries to solve the mistakes of the strategies or the effect of the disturbances it is too late, and the process need double time to became stable again.

The task group formed by “Veolia Water System” (krüger, in Copenhagen, Denmark) and the department of “Industrial Electrical Engineering and Automation (LTH, Lund, Sweden) was formed to improve the wastewater treatment processes using sophisticate techniques to model the whole process including the sensors model, the controller model and the plant model. Within the group rose the idea of simulating the wastewater process plant in real time. What does this idea mean? How can the real time simulation of the WWTP improve the behaviour or the control of the plant?

The answer to these questions is the motivation of this thesis. If it is possible to implement a perfect model of the WWTP and the controller that is being applied in the real WWTP, and to start the simulation of the process from the values of the data measured in the real plant, it would be possible to have the progress of the WWTP at the same time as the progress of the model implemented, and it would be easy to compare the values of the state variables of the simulation and the state variables of the real process. Comparing these values would be suitable to find disturbances in the process or important features of the plant that are not included in the model used usually to check the control strategies in simulation time. By the other side, the values of the simulation in real time would let to see the progress of some state variables that the real measurements can not let due to the characteristics of the sensor: the noise, the error, the precision, etc.

1.2 OBJECTIVES

In the point before has been discussed the motivation of the present thesis, but there is not enough with motivations to developed a thesis. The motivation is the beginning, the main idea to solve a problem, to improve a process, to do anything.

The motivation says that it would be a good idea to simulate the model of the WWTP at the same time that the plant is working, with the same conditions, the same value of the state variables, the same control strategy, and after, to compare the results of the simulation and the measurements of the process. But now the motivation has to be translated to objectives, which will represent a concrete aims.

If we start to analyze the motivation, the purpose is to simulate a model while the WWTP is working. So the first objective will be the implementation of an application to simulate the model of the WWTP.

The controller has to compare the results of the simulation and the measurements of the WWTP. It points that it is need a communication between the application developed and the controller of the WWTP.

The application is simulating the model of the WWTP at the same time that the real plant is working, and the controller is transferring data from the real process to ensure the model of the simulation and the real plant have the same conditions. So not is only needed a communication between the application and the controller, the synchronization is needed too.

At last, the most important is the model. The model of the WWTP has to be developed. There are a lot of possibilities to implement a model of the WWTP, and only to model one part of the plant can be a master thesis. Here in this thesis in implemented the model of the batch process (see *chapter 2* to know what a batch process is) in one vessel. From the model exposed is possible to implement the full model of the secondary treatment in WWTP, but it will not be the aim of this thesis.

Summarising, the objectives of the present thesis are:

- Implementation of an automatic application to simulate a model of the WWTP in real time.
- To program the communication and the synchronization with the controller.
- To model batch process in one simple vessel and implement it.

1.3 CONTRIBUTIONS

There have been named the objectives of the thesis to get improve the control in wastewater treatment plants. To get these objectives is join to the techniques and applications that are going to be used for getting them.

The whole code programmed in this thesis and the user interface of the application has been programmed within Matlab package toolbox. Matlab is a powerful application that let the researchers to implement, simulate, check, program, control near all the things. There are a big number of packages to complement Matlab and let the user to execute different things or processes sharing the same workspace, thing that allows interconnect implemented applications in Matlab and simulations that are running at the same time, without translating the data from one application to another.

The thesis has each chapter divided in two virtual parts; the first one explain the abstract discussion of the topic that is going to be developed. After the abstract discussion, the second part is the implementation of the topic discussed in a real and operative code.

Starting from the paragraphs before, the contributions can be understood as:

- The design of an automatic application that simulate the model of the real WWTP in real time.
- The implementation of the automatic application in Matlab.
- The communication of the application with the controller through the Ethernet network. The basis and the implementation in Matlab.
- The synchronization of the controller and the implemented application, the design of several possibilities and the posterior implementation of it.
- The design of the batch process model, with the evolution of the equations along all the phases, and the translation of the equations into code to implement the model.

The contributions have been presented as independent works, but they can be read as a main big contribution, the contribution of implementing an application to get the combination between the abstract part of the control and the real part of the control in the process working together.

The abstract part would be the implementation of the model of the process that is going to be controlled, all the part of the code programmed dedicated to model and simulate the WWTP plants. And the real part would represent all the structures and devises that form the WWTP, the controller, the sensors, the actuators.

The implementation of this application is not the substitute for the first simulation and testing of the control strategies before applying it, it is like an automatic test of the control strategy in real time, a test to verify the control of the WWTP while it is working.

1.4 OUTLINE OF THE THESIS

The distribution of the thesis try to get the better way to understand the topic developed in. It tries to introduced and explain all the concepts used along the thesis, and to give the necessary bibliography to look for extra information if it is necessary for the reader, not only to better understanding the thesis, but also to increase the given information in the present work.

The thesis is divided in four parts:

1. Introduction.
2. Modelling the Wastewater Treatment Plant in Matlab.
3. Appendixes.
4. Bibliography.

Each part is divided in chapters that expose the topics developed along the thesis. The first and the second part form the work of the thesis, and the third and fourth part are complementary to read more about the topics developed in the first and second part.

The first part is an introduction about wastewater treatment processes, what they are, how to model, the parts of a WWTP plant, the previous knowledge to better understand the thesis. This part is divided in two chapters: *chapter 1* and *chapter 2*. The *chapter 1* is the current chapter. The *chapter 2* is an introduction about wastewater treatment plants, the description of the process, automation in WWTP and the basis of modelling WWTP.

The second part is that we can call the work developed. There are three chapters: *chapter 3*, *chapter 4* and *chapter 5*. The *chapter 3* contains the design and implementation of the main application and the user interface. In *chapter 4* is discussed and programmed the communication with the controller and the database and the synchronization with the controller. In *chapter 5* is designed and programmed the batch process model in Matlab.

The third part contains the appendixes to the chapters of the first and second part, and the last part is the bibliography used to develope the thesis.

CHAPTER 2

Wastewater Treatment Processes

Previous to the full report of the project presented in this thesis it is essential a brief explanation about wastewater treatment processes, what is that, basic modelling and automation within it.
In the pages below it is made a description about wastewater treatment processes, giving more details of the second treatment in the plant. After it, a brief introduction about automation in wastewater treatment and finally the basic modelling in wastewater treatment plants is presented.
In this chapter is presented only a basic idea about description, automation and modelling wastewater treatment, for further knowledge it is the possibility of checking the *chapter 7* ("Appendix of CHAPTER 2") and the bibliography.

2.1 PROCESS DESCRIPTION

There are four disciplines involved in wastewater treatment processes, and we need all of them to make work the wastewater plant. In the figure below there is a schema of these disciplines:

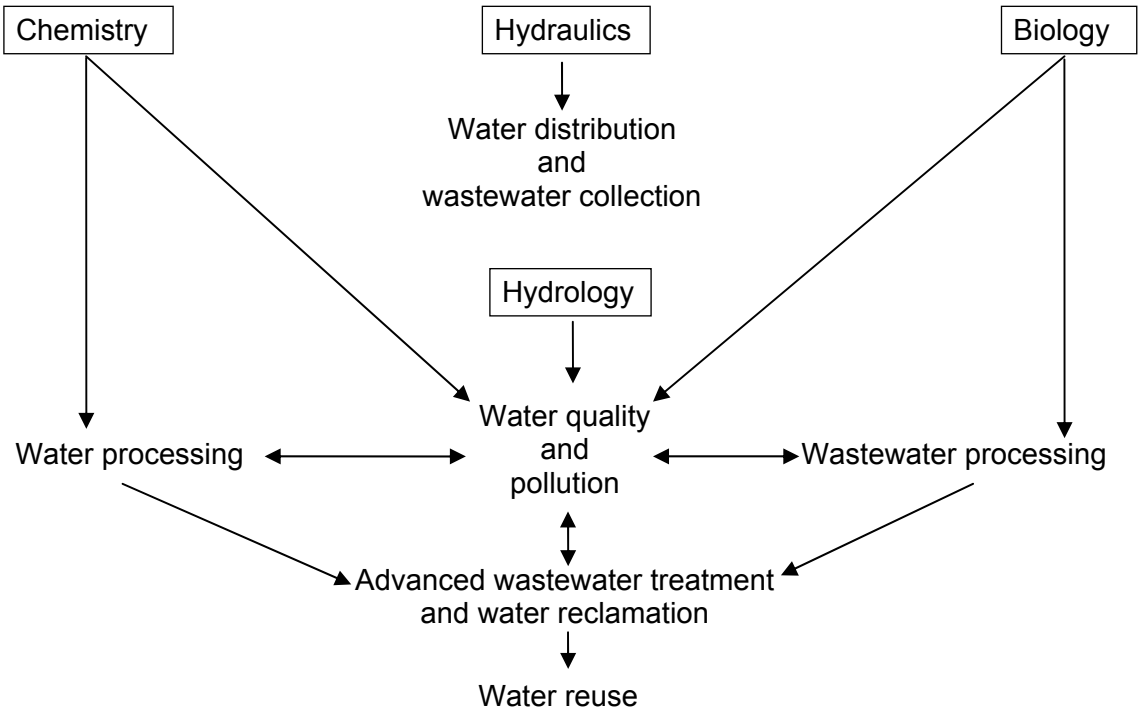


figure 1: disciplines involved in wastewater treatment

The process starts when wastewater is collected in a sewer system and transported to a plant for treatment prior to disposal (hydraulic discipline). Once the wastewater arrives to the treatment plant it has to pass through three main treatments:

figure 2: main treatments in wastewater treatment plants

The *primary treatment* is the simplest one, and the most of times it consists in a group of screens and primary settler to remove the largest rejects (plastic bags, food and another solid products) and to skims off floating greases and oils. After this first treatment it is the *secondary treatment*, where it takes part the biological reactions, chemical precipitation, nutrient mass transfer, biomass growth and settling and clarification; it is called Biological Nutrient Removal (BNR). The last treatment, *tertiary treatment*, is a group of filters and membrane techniques and, this part of the plant can change considerably, it depends of the final use of the e-fluent water (human consume, industry use, river recirculation, etc); this treatment attempts to limit the micro-organisms and other pathogens in the treated water.

All the work of this thesis has been developed about modelling of *secondary treatment*, and below it is showed the scheme of this treatment:

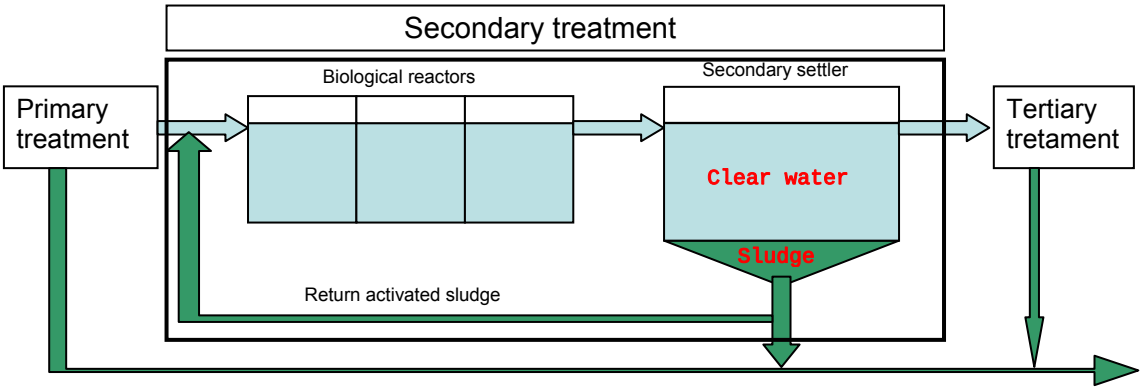
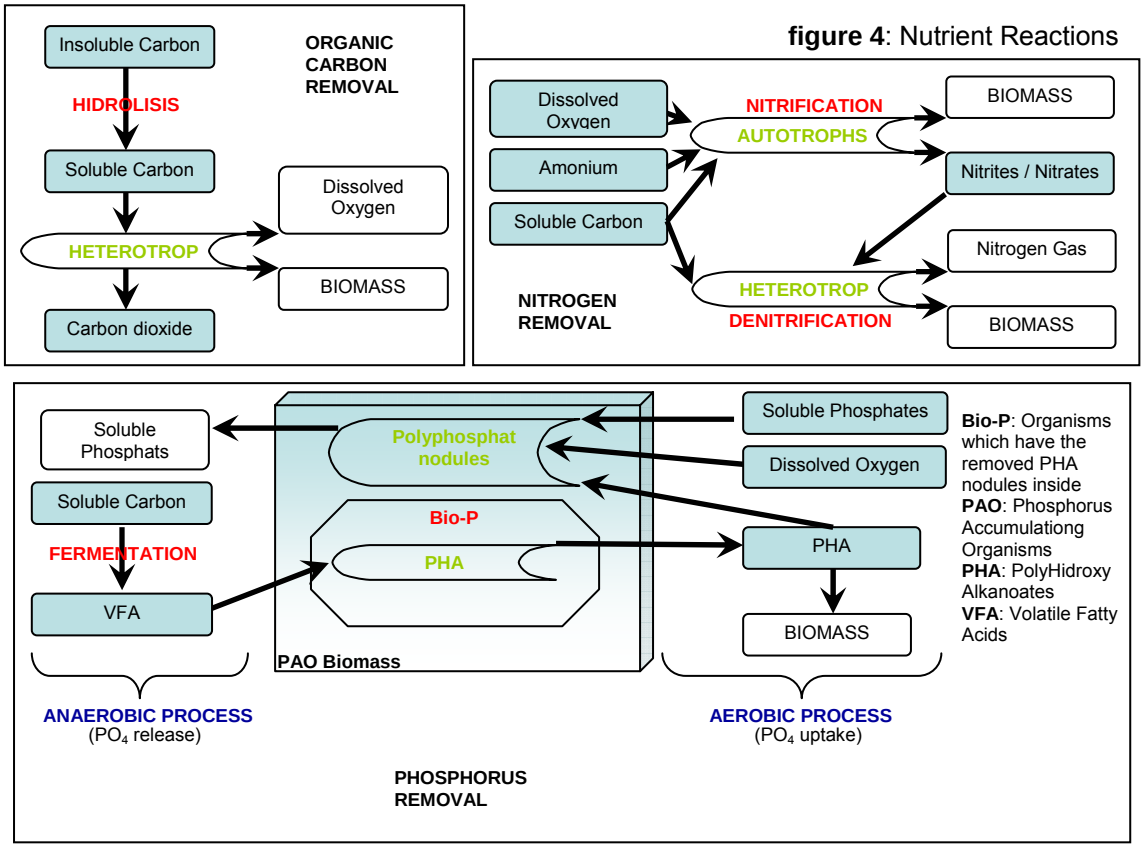


figure 3: Layout for the secondary treatment in a continuous wastewater treatment plant

We can find in the secondary treatment the following mechanisms:

- **HYDRAULICS**: Wastewater flows through the treatment plant by the force of gravity if it is possible, and pumping the wastewater to lift it from the sewer to the plant inlet. The mechanisms which take part are the vessels, which are usually concrete tanks of various shapes and sizes with controlled levels, and pumps which are almost used to recycle mixed liquor and sludge and the small sludge wastage streams.
- **NUTRIENT REACTIONS**: The biological reactions involved are: *removal of organic carbon*, *removal of nitrogen* (**nitrification** and **denitrification**) and *removal of phosphorus*. The reactions are showed below:

figure 4: Nutrient Reactions



Carbon removal and nitrogen removal are the most common process modelled and used in wastewater treatment. The first one is easily understood, but the complexity of nitrogen removal makes it was necessary a brief explanation of it.

Predenitrification AS process

Many modern treatment plants utilising AS have biological nitrogen removal. Biological nitrogen removal relies on nitrifying and denitrifying bacteria for removal of nitrogen in two steps (as it is showed in figure 4): *nitrification* and *denitrification*. Two different types of bacteria cultures are used to achieve nitrification and denitrification:

- autotrophic bacteria uses inorganic carbon as carbon source.
- heterotrophic bacteria uses organic carbon as carbon source.

In the nitrification step, ammonium is oxidised to nitrite and then nitrate (nitrification) by autotrophs. In the second step, nitrate is reduced to nitrogen gas (denitrification) by heterotrophs. A difficulty with this procedure is that the two steps require different ambient conditions to function effectively. The nitrification step needs dissolved oxygen, whereas the denitrification step requires an oxygen free environment. A solution to this is to divide the reaction volume into separate compartments in which the conditions are different.
A relatively common configuration for nitrogen removal is the predenitrification process. The first reactor is anoxic, that is no dissolved oxygen is present, and is followed by an aerated volume. This may appear somewhat backwards as the nitrification is done after the denitrification. However, the denitrification process requires readily biodegradable organic substrate and this is normally present in the influent wastewater. If the denitrificaiton has to take place after the nitrification (i.e. a post-denitrification configuration), most of the organic substrate will have been consumed and external carbon will have to be added. Thus, to provide the anoxic reactor with nitrate, a recirculation stream is introduced from the last reactor to the first reactor (sometimes the sludge recirculation is sufficient and no internal recirculation is needed).

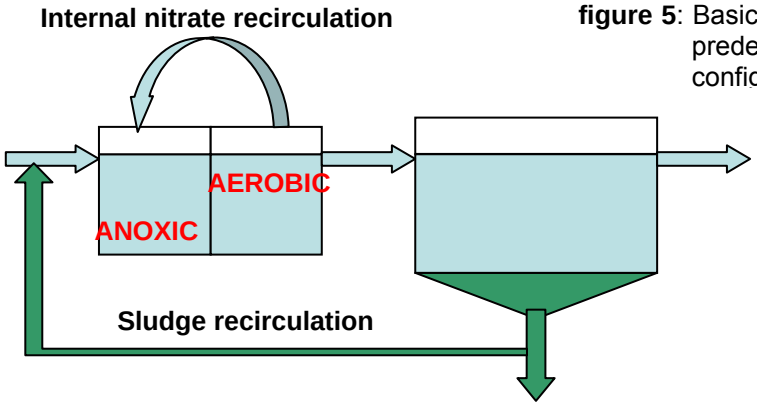


figure 5: Basic principle of the predenitrification configuration

- **CHEMICAL PRECIPITATION**: Sometimes it is possible to remove the phosphorus using *chemical precipitation* instead of *nutrient reactions*; it is faster and more economically favourable. This mechanism is made by the addition of aluminium or iron compounds such as alum, pickle liquor, ferric hydroxide or ferric chloride.

- **NUTRIENT MASS TRANSFER:** this mechanism occurs in biological reactions and the purpose is to get oxygen from gaseous air into water, and it is promoted in stirred vessels.

- **BIOMASS GROWTH:** It means the maintenance of the right proportions of different species of organisms within the sludge in an active state. The aim is to avoid the growth of harmful organisms.

- **SETTLING AND CLARIFICATION:** Finally it takes part the separation of insoluble particles form water. It is used the gravity force, thus the insoluble particles which are denser than the water go down the settler and they form the sludge (see figure 3).

2.2 AUTOMATION IN WASTEWATER TREATMENT PLANTS

Online measuring and data collection systems

The number of measurable entities increases as research on and development of instrumentation and sensors progress. A difficulty in online measuring is the aggressive environment in which the sensors must function and that many of the interesting entities must be derived from reaction analysis in batch or continuous experiments.
Interesting development in the sensor area involves new types of sensors such as sensor arrays or soft sensors, where variables are deduced from a number of measurements, biosensors that utilise (immobilised) cultures of bacteria, and microbial sensors for measuring inhibitors and substrates.

In the last few years, a wide variety of biosensors have been developed. Within this large field, microbial sensors stand out because of their multi-receptor behaviour. In fact, these results in rather poor selectivity, but a great variety of substances can be detected simultaneously with one sensor. It is for this reason that microbial sensors are preferred for measuring environmentally relevant summary parameters (such as Biochemical Oxygen Demand) or for detecting inhibiting and toxic effects.
Compared to enzyme sensors, their higher stability and lower production costs are additional arguments for using microbial sensors for environmental analyses.
With rapid, direct, and exact detection of these parameters with microbial sensors, it will be possible to make the momentary situation in a water or wastewater stream visible. This results in new possibilities for the control of sewage plants.

The data collecting systems differ from plant to plant and from supplier to supplier but common sampling rates (in Sweden) are 10 and 12 per hour, i.e. every sixth and fifth minute, respectively. The sample values are often an average over the sampling period, during which some sensors continuously deliver values and others perhaps only once a

minute. All sensors are afflicted with time lags, but normally these are short in comparison to the dominant time constants of the process.

The most often measurements in wastewater treatment are *aerobic growth of heterotrophs*, *anoxic growth of heterotrophs*, *aerobic growth of autotrophs*, *'decay' of heterotrophs*, *'decay' of autotrophs*, *ammonification of soluble organic nitrogen*, *'hydrolysis' of entrapped organics* and *'hydrolysis' of entrapped organic nitrogen* in biological processes, and *settler velocity*, *suspend solids concentration*, *feed volumetric flow rate*, *effluent volumetric flow rate* and *underflow volumetric flow rate* in settler processes.

Control handles

There are limitations to what can be controlled in a wastewater treatment plant. This is due to a lack of powerful control handles in comparison to the relatively severe disturbances that varying influent wastewater characteristics impose on the system.
A majority of the manipulated variables are macro variables (DO and some of the chemical additions excepted) whereas some of the major mechanisms that drive the processes are on the micro level. Moreover, these mechanisms are often coupled. Thus, most control handles must be considered rather blunt and often a combination of control handles is required to reach a certain control objective.

Environmental control increasingly requires the use of analytical methods, which should be uncomplicated and thus able to operate at low cost and with minimal maintenance. Miniaturized components should make the devices suitable for mobile investigations. They should also be capable of carrying out rapid measurements. Biosensors fulfil these requirements due to their simple principle of operation.

In biological process, *soluble inert organic matter* (S_I), *readily biodegradable substrate* (S_S), *particulate inert organic* (X_I), *slowly biodegradable substrate* (X_S), *active heterotrophic biomass* ($X_{B,H}$), *active autotrophic biomass* ($X_{B,A}$), *particulate products arising from biomass decay* (X_P), *oxygen* (S_O), *nitrate and nitrite nitrogen* (S_{NO}), *NH4+NH3 nitrogen* (S_{NH}), *soluble biodegradable organic nitrogen* (S_{ND}), *particulate biodegradable organic nitrogen* (X_{ND}) and *alkalinity-molar units* (S_{ALK}), and in activated sludge, *suspend solids concentration* and *sludge concentration* are examples of the state variables we can control in a wastewater treatment plant.

STAR controller (Superior Tuning And Reporting) is an example of commercial controller for wastewater treatment plants distributed by Krüger (VEOLIA group); the one used to developed the thesis. Its main performances are:

- Supervisory control system for WWTP
- Control modules for different unit processes
- Rule-based control
- Data screening and reconciliation system
- Web-based interface

Below it is showed the network interface of the STAR controller. Through this software interface it is possible to check the values measured by the controller from every place if there is an Ethernet connection.

figure 6: Ethernet interface of the STAR controller

Process dynamics

A wastewater treatment process consists of many sub-processes with dynamics of different time scales. Some variations are slow, for instance sludge dynamics and temperature, with time scales of days, week and even months. The daily variation in influent flow rate and substance concentrations is perhaps the most dominant variation. However, there are even faster dynamics present, such as dissolved oxygen (DO) dynamics and hydraulic shocks. The different time scales make it difficult to analyse the cause-effect relationships, especially when recirculation and other feedback loops are present. Therefore, it is important to establish the dynamic behaviour of the involved processes and adapt the analysis methods in accordance to the dynamics.

It is important to project out the different dynamics within the sub-processes in Biological Nutrient Removal. It is possible to divide the process dynamic in six sub-processes as well as it was showed before in "*process description*". Thus these are different dynamics:

- **HYDRAULIC DYNAMIC**: Commonly the exponential lag characterizes the hydraulic dynamic, and the size of the lag depends on the weir geometry.
  In spite of this, it is usual to find different time constants. In instance intensive variable, concentration or temperature, dynamics for flow through vessels are characterised by the mean residence time (vessel volume divided by flow rate). This parameter is commonly known as the hydraulic residence time or HRT. The spread of the residence time depends on the flow patterns within the vessel which will fall somewhere between plug flow, no spread giving a dead time response, and perfectly mixed a first-order lag with time constant equal to the residence time. The time constant is typically of the order of minutes to hours.
  Flow down pipes causes a transport delay for intensive variables such as nutrient concentrations. The dead time is equal to pipe length divided by liquid velocity. These are unlikely to be significant with the possible exception of return activated sludge which could experience dead times of a few minutes, and poorly designed analyser sampling systems which have been known to have dead times of hours.
  Recycle flows slow down the dynamics of intensive variables. For a well-mixed vessel, the time constant of the dynamic response is (R+1) times the mean residence time of the vessel, where R is the recycle ratio (recycle flow rate divided by inlet flow rate).

- **NUTRIENT REACTIONS:** Nutrient removal by the biomass in a well-mixed reactor effectively speeds up the dynamics for that nutrient concentration compared to the residence time of the tank. If the effluent concentration is one tenth the influent concentration, the time constant is one tenth the reactor residence time. It follows that time constants for nutrient concentrations will be within an order of magnitude of the reactor residence times, and generally smaller, depending on nutrient removal ratios and recycle ratios. Thus time constants of the order of 1 to 10 hours could be expected. Recirculation of nitrate in a pre-denitrification system is a fast process, since the flow rate is so high. Consequently the nitrate concentration can be changed within minutes.

- **CHEMICAL PRECIPITATION**: The most important aspect of chemical precipitation is to achieve rapid and complete mixing. Once the chemicals are mixed with the wastewater, the dynamics are very fast, responding in a matter of seconds.

- **MASS TRANSFER:** The transfer of oxygen from a gaseous form to a dissolved form takes place within a time scale of 15-30 minutes. A change in air flow rates therefore does not immediately affect the dissolved oxygen concentration in the aerator. The respiration rate may change within minutes due to changes in substrate loading or toxic inputs. This will result in DO changes that take place in the time scale determined by the DO dynamics.

- **BIOMASS GROWTH**: If biomass growth and sludge wastage are in balance, the sludge dynamics time constant is equal to the solids residence time (SRT). The SRT is the solids hold-up divided by the solids wastage, where the suspended solids in the effluent should be neglected with great care. This is typically of the order of several days for the activated-sludge process. Significant changes in the proportions of different species within the biomass could take longer, maybe several weeks for slow growing species.

- **CLARIFICATION:** The flow patterns in a clarifier will approximate plug flow upwards for clarified effluent and plug flow downwards for return activated sludge. This would typically involve a dead time of 1-4 hours in each stream.

Summarising, it is possible to distinguish three kind of dynamics in wastewater processes attending to the speed; *fast dynamic* (flow dynamic and dissolved oxygen dynamic), *medium dynamic* (concentration dynamic and nutrient removal) and *slow dynamic* (biomass growth).

2.3 MODELLING OF WASTEWATER TREATMENT PROCESSES

First, it is necessary to leave clear some questions about the nomenclature in going to be used in this point, in order to understand the equations shown below:

S, used for concentrations of components dissolved in wastewater
X, used for concentrations of solid components in wastewater
V, used to refer to the tank volume (it is assumed is constant)
q_{in} or q_{out}, referred to the input and output flow rate

After this brief introduction, how modelling wastewater treatment plants is exposed and, as it was shown before, can be divided in different groups:

Modelling mass transfer

Mass transfer is defined as the movement of a component from one phase to another. The main mass transfer in wastewater treatment is the **transfer of oxygen** from the air into the water to be used by the biomass.
The mass transfer is represented as:

$$r_a = K_L a(S_{O,sat} - S_O) \qquad (2.1)$$

, where: 'r_a' is rate of oxygen transfer
'K_L' is the mass transfer coefficient
'a' is the air water surface area
, and '$S_{O,sat} - S_O$' is the concentration difference

In that way, if we consider a well mixed tank with constant volume, chemical oxygen demand (COD) and water, without taking care about the transfer of oxygen, the component mass balance is the one below:

$$\frac{d(VS_O)}{dt} = q_{in}S_{O,in} - q_{out}S_O \tag{2.2}$$

, but now the new oxygen mass balance in the tank is:

$$\frac{d(VS_O)}{dt} = q_{in}S_{O,in} - q_{out}S_O - K_L a(S_{O,sat} - S_O)V \tag{2.3}$$

Modelling biological nutrient removal

In biological nutrient removal there are involved different processes; carbon removal, nitrogen removal and phosphorus removal. And it is pertinent to add the modelling of a single nutrient, because it has to be modelled not only the nutrient removal but also the biomass present in wastewater. By the other side it is assumed a well-mixed tank for modelling biological nutrient removal.

- **MODELLING A SINGLE NUTRIENT**: In the simplest example there are involved three components; water, nutrient and biomass. The microorganisms absorb the nutrient from the wastewater using it to grow and to produce more biomass. We assume constant volume tank and constant density so:

  Mass balance: $q_{out} = q_{in}$ (2.4)

  Biomass balance: $\frac{d(VX_B)}{dt} = q_{in}X_{B,in} - q_{out}X_{B,out} + r_B V$ (2.5)

  Nutrient mass balance: $\frac{d(VS_N)}{dt} = q_{in}S_{N,in} - q_{out}S_{N,out} - r_N V$ (2.6)

  , where: r_B is reaction rate for biomass growth
  r_N is nutrient reaction rate
  X_B is biomass concentration
  S_N is nutrient concentration

- **MODELLING CARBON REMOVAL**: Now let to go further, and to consider four components; water, soluble carbon, oxygen and heterotrophic biomass. It is the simplest example of biological treatment of wastewater. There are the balances shown below:

  Heterotrophic biomass balance: $\frac{d(VX_H)}{dt} = q_{in}X_{H,in} - q_{out}X_{H,out} + r_H V$ (2.7)

  Carbon mass balance: $\frac{d(VS_S)}{dt} = q_{in}S_{S,in} - q_{out}S_{S,out} - r_S V$ (2.8)

  Oxygen mass balance: $\frac{d(VS_O)}{dt} = q_{in}S_{O,in} - q_{out}S_O - K_L a(S_{O,sat} - S_O)V$ (2.3)

, where: r_H is reaction rate for biomass growth
r_S is nutrient reaction rate
X_H is biomass concentration
S_S is soluble carbon nutrient concentration

- **MODELLING NITROGEN REMOVAL**: The main description of this process was made before, and it was exposed there are two steps for nitrogen removal:

1. Aerobic growth of autotrophs
2. Anoxic growth of heterotrophs

Now there are two biological processes; carbon removal and nitrogen removal. The balances of the different concentrations to model the nitrogen removal are:

Autotrophic biomass balance: $\frac{d(VX_A)}{dt} = q_{in}X_{A,in} - q_{out}X_{A,out} + r_A V$ (2.9)

Ammonia nutrient mass balance: $\frac{d(VS_{NH})}{dt} = q_{in}S_{NH,in} - q_{out}S_{NH,out} - r_{NH}V$ (2.10)

Nitrate mass balance: $\frac{d(VS_{NO})}{dt} = q_{in}S_{NO,in} - q_{out}S_{NO} - r_{NO}V$ (2.11)

, where: r_A is reaction rate for autotrophic biomass growth
r_{NH} is reaction rate of ammonia nutrient
r_{NO} is reaction rate of nitrate
X_A is biomass concentration
S_{NH} is soluble ammonia nutrient concentration
S_{NO} is nitrate concentration

- **MODELLING PHOSPHORUS REMOVAL**: In the *figure 4* it is shown the process of phosphorus removal. It is evident the phosphorus removal is a complex process for modelling, but there are for basic mechanisms to describe it:

1. Fermentation of fermentable COD, S_F, to volatile fatty acids (VFA), S_A, which can be utilised by the PAO microorganisms to store carbon as polyhydroxyl-alkanoates (PHA), X_{PHA}.
2. Phosphorus release from poly-phosphate (PP), X_{PP}, into solution at the same time as the VFA is converted to PHA.
3. Phosphorus uptake from solution to PP utilising the PHA and dissolved oxygen, S_O.
4. Growth of the PAO biomass, X_{PAO}, also utilising the PHA and dissolved oxygen.

And this model of the phosphorus removal process introduces five more mass balances and states: S_A, S_{PO4}, X_{PHA}, X_{PP}, X_{PAO}.

The additional mass balance is integrated with the nitrogen removal model, but it is replaced the single component balance (S_S) by two component balances (S_A and S_F) and using the kinetic expressions shown in the tabular forms in *appendix I* (appendixes to CHAPTER 2).

- **ACTIVATED SLUDGE MODEL Nº1 (ASM1)**: The Activated Sludge Model No.1 (ASM1) was developed by a task group work in collaboration with the International Water Association (IWA, formerly IAWQ and IAWPRC) in 1983 and published in 1987.
  The state variables included in the ASM1 are listed in the table below:

The state variables of the ASM1 model

Symbol	Variable
S_I	Inert organic matter
S_S	Readily biodegradable substrate
X_I	Particulate inert organic matter
X_S	Slowly biodegradable substrate
$X_{B,H}$	Active heterotrophic biomass
$X_{B,A}$	Active autotrophic biomass
X_P	Particulate product from biomass decay
S_O	Dissolved oxygen
S_{NO}	Nitrate and nitrite nitrogen
S_{NH}	Ammonia nitrogen
S_{ND}	Biodegradable organic nitrogen
X_{ND}	Particulate biodegradable organic nitrogen
S_{ALK}	Alkalinity

Table 1: state variables of the ASM1 model.

And to model the total suspended solids (TSS), which is a normally measured, in wastewater treatment plants, it is used the following conversion:

$$TSS = 0,75(X_I + X_P + X_S) + 0,9(X_{B,H} + X_{B,A}) \qquad (2,12)$$

There are eight different dynamic processes in the ASM1 model for describing the dynamics:

1. Aerobic growth of heterotrophs
2. Anoxic growth of heterotrophs
3. Aerobic growths of autotrophs
4. Decay of heterotrophs
5. Decay of autotrophs
6. Ammonification of soluble organic nitrogen
7. Hydrolysis of entrapped organics
8. Hydrolysis of entrapped organic nitrogen

Phosphorus removal is not modelled in ASM1 model, the one used in this thesis for programming batch process with Matlab as it will be shown later.

Modelling hydraulics

To begin, it is assumed there are no waves on the surface, so the total flow rate over all the weirs with an arbitrary shape is:

$$q_{out} = N\int_0^h b(z)v_z(z)dz \qquad (2.13)$$

, where: $v_z(z)$, is the velocity of the flow rate
$b(z)$, is the shape of the weir
z, is the translocation from the surface of the weir until where we are measuring the velocity

In the flow-sheet below it is shown all of them:

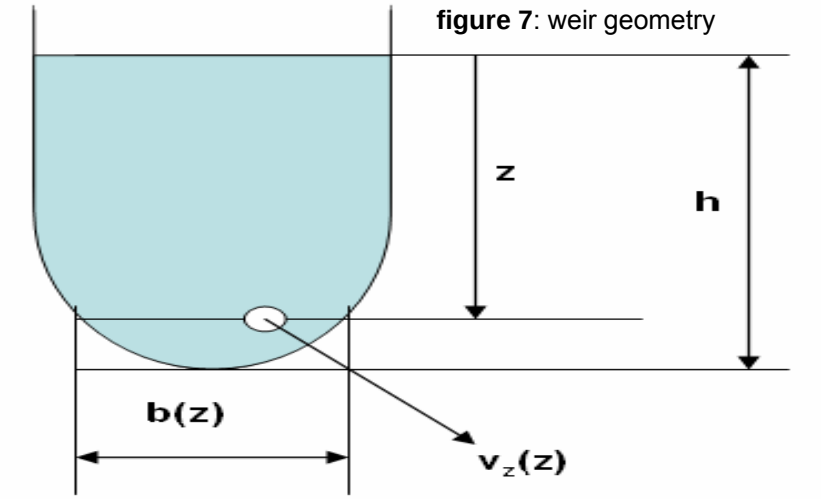


figure 7: weir geometry

The liquid velocity is calculated from Bernoulli equation:

$$p_{stat} = p_0 + z\rho g - \frac{1}{2}\rho v_z^2 \qquad (2.14)$$

, where 'ρ' is the density and 'g' is the gravity acceleration. It is assumed the pressures are equals, so $p_{stat} = p_0$ and the velocity of the flow rate is:

$$v_z(z) = \sqrt{2gz} \qquad (2.15)$$

If the equation 2.15 is inserted into equation 2.13 the total flow rate becomes:

$$q_{out} = N\sqrt{2g}\int_0^h b(z)\sqrt{z}dz \qquad (2.16)$$

Modelling settling and clarification

For modelling settling and clarification is used the one-dimensional layer model. It only describes the settling process along the vertical axis, leaving cross-sectional area and depth as design parameters.
The main assumptions in this model were made by Stenstrom in 1975:

- The continuous thickener does not exhibit vertical dispersion.
- The concentration of suspended solids is completely uniform within any horizontal plane within the settler.
- The bottom of the solids-liquid separator represents a physical boundary to separation and the solids flux due to gravitational settling is zero at the bottom.
- There is no significant biological reaction affecting the solid mass concentration within the separator.
- The mass flux into a differential volume cannot exceed the mass flux, the volume is capable of passing, nor can it exceed the mass flux which the volume immediately below it is capable of passing.
- The gravitational settling velocity is a function only of the suspended solids concentration except when the assumption immediately above is violated.

The model was extended to include the clarification zone (not included by Stenstrom) by Vitasovic in 1985. The settler was divided into *'n'* layers with the feed entering in layer *'m'*. It is assumed the feed is instantaneously and completely distributed throughout the feed layer.
Thus, the region below the feed level is modelled according to Strenstrom's approach and the clarification zone according Vitasovic's extension of the model. The equations of the model are exposed below:

Concentration in each layer (not valid for feed layer and thickening layer) in the settling zone:

$$\frac{dX_i}{dt} = \frac{v_{dn}\left(X_{i-1} - X_i\right) + \min\left(J_{s,i}, J_{s,i-1}\right) - \min\left(J_{s,i}, J_{s,i+1}\right)}{z_i} \qquad (2.17)$$

, where: J_s, is the settling flux
z_i, is the height of layer i
v_{dn} is defined by: $v_{dn} = \frac{Q_u}{A}$, where 'Q_u' is the underflow volumetric flow rate and 'A' is the cross –sectional area of the settler.

The concentration for the bottom layer is:

$$\frac{dX_n}{dt} = \frac{v_{dn}\left(X_{n-1} - X_n\right) - \min\left(J_{s,n-1}, J_{s,n}\right)}{z_n} \qquad (2.18)$$

In the clarification zone, from layer *'2'* to layer *'m-1'* (where *'m'* is the feed layer), it is given the following equation:

$$\frac{dX_i}{dt} = \frac{J_{up,i+1} + J_{clar,i-1} - J_{up,i} - J_{clar,i}}{z_i} \tag{2.19}$$

, where the fluxes are defined as:

$$J_{up,i} = v_{up} X_i \tag{2.20}$$

$$v_{up} = \frac{Q_e}{A} \tag{2.21}$$

$$J_{clar,i} = \begin{cases} J_{s,i} & if\ X_{i+1} \le X_t \\ \min\left(J_{s,i}, J_{s,i+1}\right) & if\ X_{i+1} > X_t \end{cases} \tag{2.22}$$

And the equation for the feed layer is:

$$\frac{dX_m}{dt} = \frac{\frac{Q_f X_f}{A} + J_{clar,m-1} - \left(v_{up} - v_{dn}\right)X_m - \min\left(J_{s,m}, J_{s,m+1}\right)}{z_m} \tag{2.23}$$

, where: *'Q_f'* is the feed volumetric flow rate to the settler
'X_f' is the suspended solids concentration of the feed.

The equation to describe the top layer becomes:

$$\frac{dX_1}{dt} = \frac{J_{up,2} - J_{up,1} - J_{clar,1}}{z_1} \tag{2.24}$$

In *figure 7* it is shown graphically a general description of the one dimensional layer model.

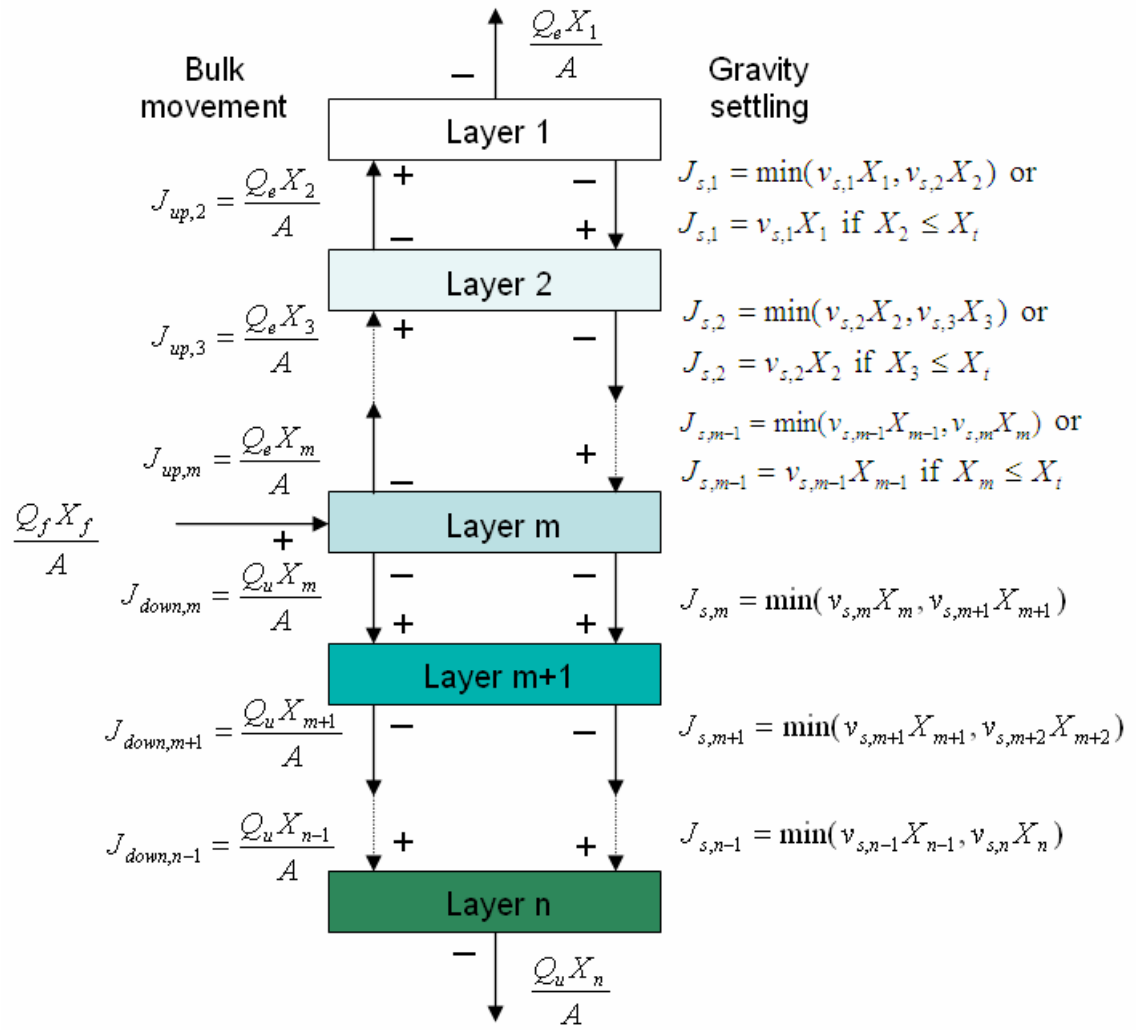


figure 8: General description of the traditional one-dimensional layer settler model (Jeppsson; 1996)

It has been shown the one dimensional layer settler model and all the equations to define it, but it hasn't said anything about the settling velocity. The determination of an appropriate settling velocity model is indispensable for modelling the secondary clarifier using the solids flux theory. In the model used in this thesis it is chosen the empirical double-exponential settling velocity function proposed by Takács et al. (1991):

$$v_s = \max\left(0, \min\left(v_0', v_0\left(e^{-r_h(X-X_{\min})} - e^{-r_p(X-X_{\min})}\right)\right)\right) \qquad (2.25)$$

, where: v_0' is the maximum settling velocity
r_h is a settling parameter characteristic of the hindered settling zone
r_p is a parameter associated with the settling behaviour at low solids concentrations
X_{min} is the minimum attainable concentration of suspended solids in the effluent and it is possible to express it as:

The minimum attainable concentration of suspended solids is defined by the equation:

$$X_{\min} = f_{ns} X_f \tag{2.26}$$

, where f_{ns} is the non-settleable fraction of X_f (suspended solid concentration of the feed)

In *figure 8* it is shown the function of the double-exponential settling velocity where it is possible to see how the velocity would increase if it was defined by a single exponential function:

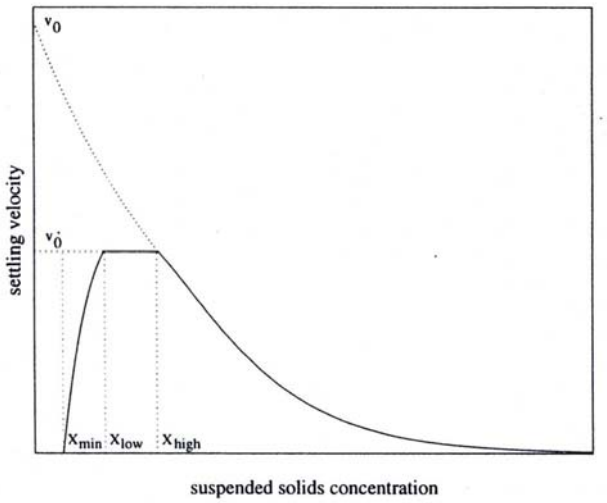


figure 9: schematic description of the double-exponential settling velocity (equation 2.25) model suggested by Takács et al. (1991)

Modelling batch processes

Moreover continuous modelling processes there is another possibility which has increased in popularity the last years and it is used in many wastewater treatment plants. This one is Modelling ***Batch Processes***.

It means until now, wastewater treatment processes have been model as a continuous system, with biological treatment tanks before the settler, so the wastewater goes through the anoxic tank, after the aerobic and finally to vessel where it takes part the settling and clarification. All of it happens always with constant volume and constant flow rate.

In batch processes there is no constant volume and there is no constant flow rate always. The aim of batch processes is to model all biological process in wastewater treatment, including settling and clarification, in the same tank.

There is no a tank for a specific biological process, there will be several tanks with a number of distinct time phases. In the *figure 9* below is shown the typical activated sludge sequenced batch reactor:

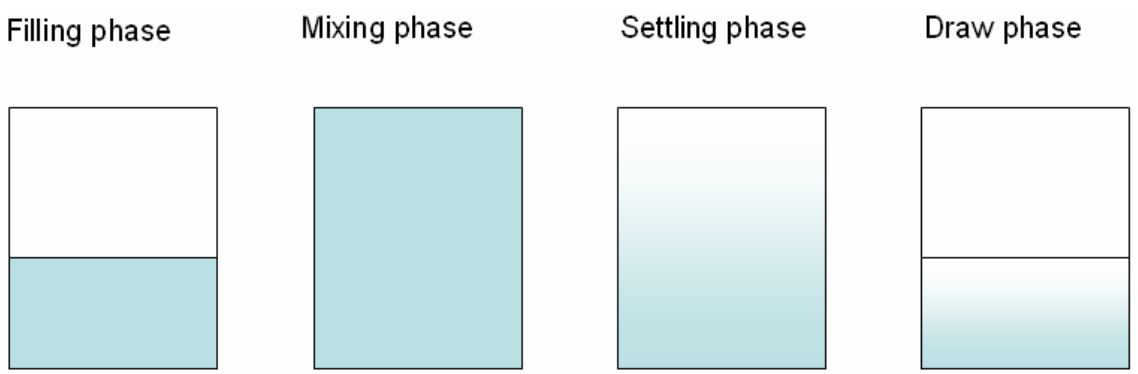


figure 10: main sequenced batch reactor (SBR)

It is possible to see in *figure 9* de difference between the phases:

- Filling phase: The wastewater is going in the tank, so the height of the tank is increasing. It is assumed a well mixed vessel.
- Mixing phase: The tank is full, the height is equal to its maximum value and it is assumed a well mixed vessel. Now it is taking part the biological processes.
- Settling phase: The height is still constant and with the maximum value, but in this phase the solid particulates are settling on the bottom of the tank.
- Draw phase: Finally when the settling process has finished it starts to draw the tank and the cleaned water returns to the weirs.

This is the simplest example of batch process. It can have as much phases as the designer needs. For instance it is possible to divide the 'mixing phase' in two phases, one the 'anoxic phase' and the other the 'aerobic phase'.

Notice that the balances are the same in all phases but the equations for each one are different depending of the phase where the process is.

PART II

TITLE: *Modelling the Wastewater Treatment Plant with Matlab*

CHAPTER 3

Application Design

Now that modelling wastewater treatment plants has been introduced it is time to expose the application designed.
This chapter is dedicated to the design of the main application with Matlab, which is connected to the STAR controller. It is included the design of the main code, the structure of the programme, the flow-sheet of the code and the functions which ones it is divided to make easier update the code in the future.
Nevertheless, it is not included the data-exchange and synchronization code and the simulink model, the following chapters (chapter 4 and chapter 5) are dedicated to those parts.

3.1 INTRODUCTION

In the chapter before it has been introduce how to model and to control wastewater treatment processes to understand the contributions developed in this thesis. But now, it is going to be described the main objective of the thesis.

This thesis is the result of the collaboration of the "industrial electric engineering and automation" department (Lund University) and "VEOLIA Water Systems" company. The project pretends to develop an automatic application able to communicate with the STAR controller (developed by the company), simulate a model of the wastewater treatment plat which is controlled by STAR and returns the results from the simulation to the STAR controller database.

The objective of this project is to improve the control handles. The STAR controller gives to the application some initial values (update the state variables of the plant model), the application simulates the plant model and returns another values to the STAR controller. The purpose of this data exchange is to make possible to compare the expected data (data from the simulation) with the real data measured from the wastewater treatment plant. Then it is possible to check differences between the expected results and the real results, and it makes easy to find out mistakes in the control strategy. And the controller is going to ask for new data from the simulation process to compare with the real measurements periodically, it means there is one part of the application is going to be executed regularly when it is running.

One of the worst problems in wastewater treatment plants is the dynamic of biological processes, it is very slow. It makes a mistake in control strategy is detected after days, even weeks sometimes. With this kind of control, comparing the real data with expected data, it is possible to detected mistakes faster, and it makes possible to reduce the waste resources along the application process of a new control strategy.

Now it is defined the problem, it has to be defined the connection between the database and the application and the platform to developed the application. As STAR controller has web-based interface, the application has to be able to connect with the database through Ethernet. The platform chosen to develop the application is Matlab, which is be able to exchange data through the network and to simulate a complex model necessary to simulate wastewater treatment processes (with Simulink features).

3.2 MAIN FLOWSHEET

Once we know what the application has to do, it is necessary to set the different tasks within the application and the order of them. In the figure below is represented graphically the main tasks of the process that the application has to realise and the order of them:

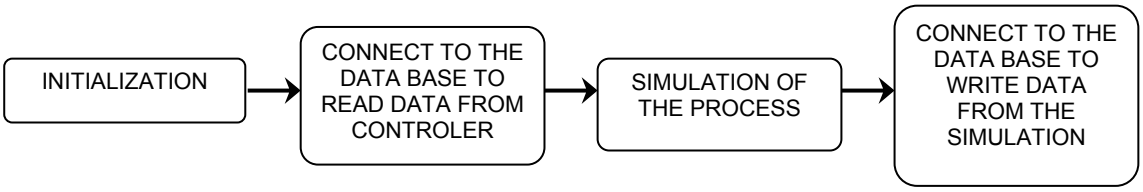


figure 11: flow-sheet diagram

As it is shown there are two tasks with the name of "connect to the database", but one is for reading data from the database and the other to write data in the database, so they are different tasks, although it is necessary to connect to the database in both of them. It means there is one similar feature in these tasks.

They are defined the main tasks of the application, but there is one feature very important and it is necessary to take care of it, because it will change the configuration of all the application. This feature is the recursive characteristic of some tasks. Therefore there is one 'main loop' in the process where the recursive tasks will be included and there is the 'initialization block', necessary to start running the program and the simulation of the wastewater treatment plant.

The main structure for the design is shown below:

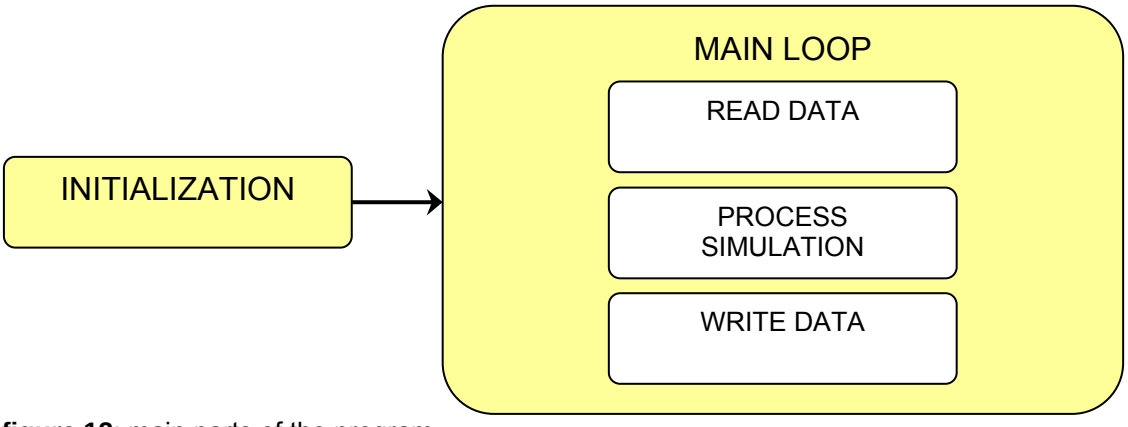


figure 12: main parts of the program

So there are the tasks are going to be executed only at the beginning of the application (*"initialization block" in figure 12*) and the tasks are going to be executed each period defined by the controller (*"main loop block" in figure 12*).

Finally there are summarized the main tasks that the application has to do, but now it is necessary to expose the specifications imposed by the company:

- The simulation background for the prototype is Matlab/Simulink.

- The communication with the database and the STAR controller through Ethernet (XML files).
- The period of the main loop is 2 minutes approximately.

3.3 STRUCTURE OF THE PROGRAM

In the point before (*3.2 Main Flow-Sheet*) it has been analyzed the main structure of the application is going to be developed, but more over the main tasks of the application, in this point it is discussed and presented the full structure of the program with its functions and all the features needed to program it.

If we remember again the main diagram of the process to program (see figure 13):

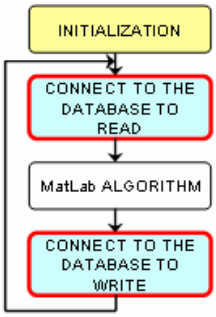


figure 13: diagram of the process

There are two tasks that they need to connect with the database of the STAR controller, well for reading data, well for writing data. It means the application is acceding to another program that is being used by the STAR controller too.
It is evident a synchronization is needed to avoid problems between both applications; the controller and the verification data application.

Thus, a synchronization block must be included within "connect to the database" block, but which one of them? To ask this question make us to see there is more than one possibility of synchronization with the STAR controller. Below are presented some possibilities:

- ***Function of synchronization inside the block "connect to the database to read":*** As it is shown in *figure 14*, the synchronization takes part when the application is going to read data from the database, after this first synchronization, the program read the data, simulate the process and write the results in the database. When all the process is finished it waits for another synchronization to start again the "main loop".

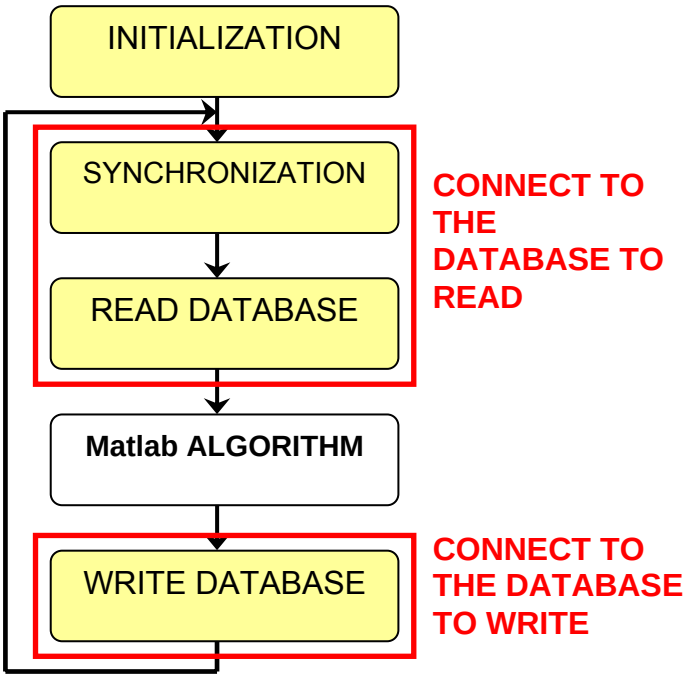


figure 14: synchronize when reading

- ***Function of synchronization inside the block "connect to the database to write":*** This time, the synchronization is made just before the application is going to connect to the database to write data, that is, the application starts to run, initialize all the variables, connect to the database to read data, simulate the process, synchronize with the controller and connect to the database to write the results. Then, it repeats the process again until the synchronization step. It means it has taken part the first access to the database without any control when the application has read the data to simulate the process. The process is shown in *figure 15*. Of course it is possible to program some kind of control to avoid this first access to the database without control the first time the application starts to run.

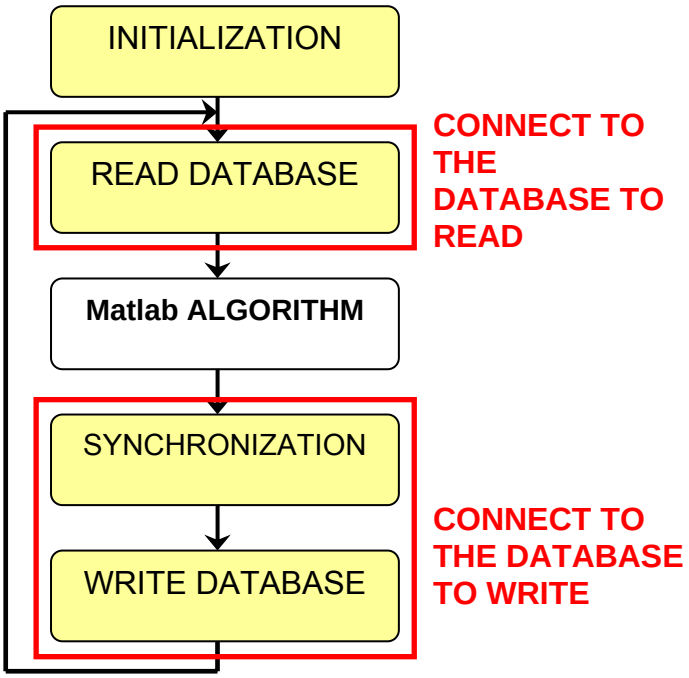


figure 15: synchronize when writing

- ***Function of synchronization in both blocks, that is, there are two synchronizations:*** This third possibility, although longer than the others, its complexity let the application synchronize each time it is going to connect with the database. Now, there is synchronization in both blocks that they connect to the data base, as much as the application is going to read from the database as it is going to write in the database. The structure of the process now it is shown in *figure 16*, on the right of these lines. First there is synchronization before connecting to the database, the simulation of the process follows that step and finally there is another synchronization before writing the results in the database. Then the main loop starts again.

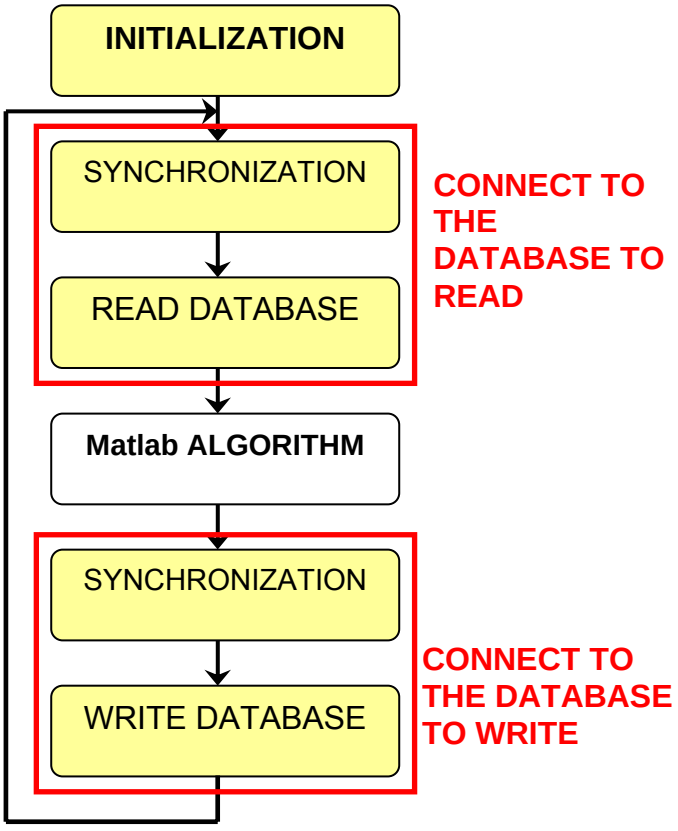
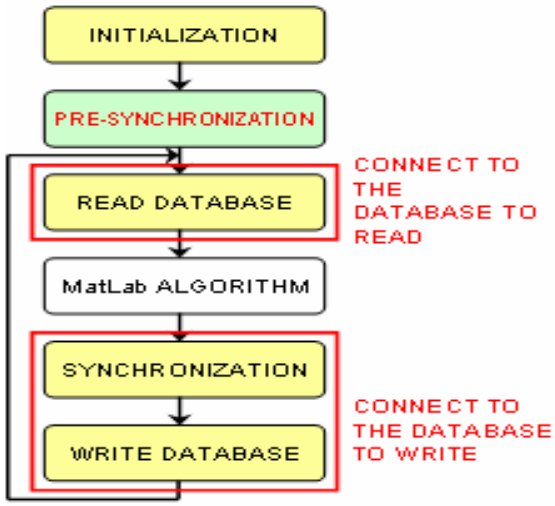


figure 16: double synchronization

All the ways of synchronization explained before are correct to program the application, but the way to "synchronize when writing" has one difficult, it is necessary to make a pre-synchronization in the initialization block before starting the main loop, then we avoid the application accesses to the database without the controller allows it.



The new flow-sheet is shown in *figure 17*. There is a pre-synchronization function after the initialization block, so now the application doesn't start to access to the database until it is allow to it. This pre-synchronization just would take part once at the beginning of the applications, each time it starts to run.

figure 17: synchronize when writing with pre-synchronization

By the other side, for "synchronizing when reading" or the last way (synchronizing always) it is no needed any extra-function to pre-synchronize the application because there is always a synchronization before the first reading time, something never happens in "synchronizing when writing" the first time that the *main loop* is executed.

The synchronization chosen is the third case: ***double synchronization*** (before reading and before writing in the database). In the chapter after (chapter 4: "data exchange") it is explained in detail the synchronization function, and in the next point; *"3.4 program code"* the code for both synchronization ways (synchronization when reading and synchronization always) is implemented, because it is the same code and the same function to synchronize but, this function is used once in "synchronization when reading" and twice in the other case. Thus they will be shown two solutions for the synchronization problem.

Now it is going to be exposed the function of each blocks in the points bellow, following the main division made before:

- Initialization of the application.
- Main loop of the application.

3.3.1 Initialization

This part of the code includes all data which need to be initialized and it is divided in several kinds of initializations:

- ***Model parameters initialization:*** It means, to establish the value of the constants, the initial value of the parameters for the simulations, the value for the parameters of simulation, the initial conditions of the simulation background.
- ***Model state variables initialization:*** This part seems unnecessary, but it is a mistake to avoid this point. Simulink always starts all state variables from cero value if there are no specific initial values for them. Thus, with a first simulation of the model it is possible to establish the initial value for the state variables trying to approximate the initial value of the state variables in the simulation process to the initial value of the state variables in the real wastewater treatment process.
- ***Control parameters initialization*** (to start the loop and the simulation)***:*** They have been included within the application the possibility of stopping the simulation when the application is running and the possibility to stop all the process of data acquisition, simulation and data writing (to make possible break the main loop). In this part of the code the "control variables" are initialized to make the main loop and the simulation run.
- ***Synchronization parameters initialization:*** This is another critic point. The variables used for the synchronization are modified by the STAR controller, which allows the application to read and write in the database, but at the beginning of the application running it is not possible to know the value of the "synchronization variables" stored in the memory, so it is suitable to initialize the variables to the expected values.

3.3.2 Main loop

The main loop implementation is more complex than the initialization, here is where all processes take part, the connection with the database to read, the simulation process, the second connection with the database to write. After making reference to all theses processes it is evident the main loop is divided in three parts (see *figure 6* shown before). Below these parts are defined:

Connection to the database to read

As it was shown before there are two steps to connect to the database, first the synchronization and after the access to the database to read the data. The two steps are explained below:

- ***Synchronization***: The synchronization is important for two main reasons: to avoid the access of two applications at the same time to the same file and to ensure there is no repetition of data, it means, that the application does not simulate the process more than once when the controller needs the data only once, or there are new results of the simulation when the controller asks for them and they are not the ones of the past simulation. Summarizing, there are new

results when the controller asks for them, there is no simulation of the process when the controller does not ask for it.
Hence, the synchronization allows to forbid the application accesses to the database until the controller needs the results of the simulation, then it let the application gets the data from the database, simulate the process and write the results.
To get the purpose exposed before it has been implemented one auxiliary loop to synchronize with the controller. Inside this loop the application is asking for accessing to the database until the controller answers that it is possible, then the loop is bracken and the application goes on with the steps of the process (read the data, simulate the process and write the data if there is no double synchronization).
In the case chosen, the double synchronization, there are two synchronizations, one before reading the database and the other before writing the results in the data base, so for each time the controller needs data it has to synchronize with the application twice. The improvement of the double synchronization case is discussed in the point "3.4.2 Program code", because this improvement is related with the implementation.

- ***Read from database***: Once the synchronization has taken part, the application accesses to the database and gets the values stored in it within variables in order to be able to manage them in Matlab background. The values needed to the simulation are the "name", the "numerical value" and the "quality value" of each variable, so the result after reading the database will be three vectors, one with all the names stored (vector "name"), one with the values stored (vector "value") and the last one with the quality of each value stored (vector "quality"). This file stores the values so that the component 'i' of each vector belongs to the variable 'i', it means:

$$\left.\begin{aligned} name &= [name_1 \ldots \quad name_i \ldots \quad name_n] \\ value &= [value_1 \ldots \quad value_i \ldots \quad value_n] \\ quality &= [quality_1 \ldots \quad quality_i \quad quality_n] \end{aligned}\right\} \text{var}\,iable_i = \begin{bmatrix} name_i \\ value_i \\ quality_i \end{bmatrix} \quad where \quad i = 0,\ldots n$$

, and 'n' is the number of variables acquired.

Matlab Algorithm

The "Matlab Algorithm", the most important step of the "main loop". The other steps are implemented to get the variables to the simulation of the process in Matlab and to return the results of Matlab simulation to the database, it means, the other functions are programmed to communicate the simulation process with the controller.

The Matlab algorithm is divided in three parts: processing the data before the simulation and processing the data after the simulation. Bellow is shown the diagram of the Matlab algorithm:

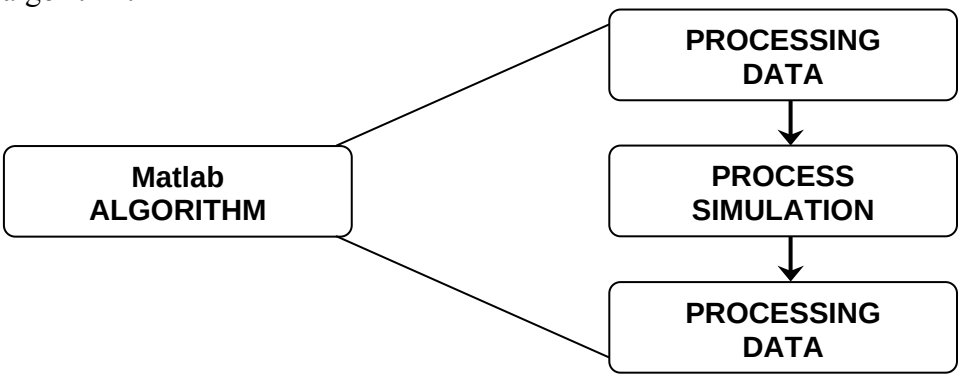


figure 18: Matlab Algorithm

- ***Processing data before the simulation***: As it has been told before, the function to read the data from the database stores them in three vectors. But these three vectors contain information about the variables about the name, value and quality. To use them it is necessary to store the information in another format. This first step of the "Matlab algorithm" is the responsible of making the translation of the information received from the database to make possible its use in the simulation process.

- ***Process simulation***: After all the steps to acquired the data and translate them, it is possible to make the simulation. In this step the application executes the necessary commands to simulate the simulink file which includes the model of the wastewater treatment plant. This model included can be whatever the company wants, from only a simple model of the wastewater treatment plant to a complex model of the wastewater treatment plant and the sensor models and with a model of the controller. The possibilities are limited only by the simulation time needed and the capacity of the computer used.

- ***Processing data after the simulation***: This last step is the data treatment before sending them to the database. The simulation returns the data in a specific format, that format is not the appropriate to send them to the controller and to store them in the database the most of times. Here all data are processed and expressed in the format used by the function which writes them in the database. This format is the one used when the application reads from the database, where all data are stored in three vectors, one with the names of the variables, another with the values of the variables and the last one with the quality of the variables. The output of this step of the "Matlab algorithm" is the three vectors shown below:

$$\left.\begin{aligned} name &= [name_1 \ldots \quad name_i \ldots \quad name_n] \\ value &= [value_1 \ldots \quad value_i \ldots \quad value_n] \\ quality &= [quality_1 \ldots \quad quality_i \quad quality_n] \end{aligned}\right\} where \quad i = 0,\ldots n$$

, and 'n' is the number of variables acquired.

Write in database

As it was shown before there are two steps to connect to the database, first the synchronization and after the access to the database to read the data. The two steps are explained below:

Synchronization: Here the synchronization if exactly the same function described in the point before "read from database". There is one auxiliary loop asking for access to the database. When the access is allowed the loop is bracken and the application goes on to the next step: write the results in the database.

- ***Write to database***: Once the synchronization has taken part, the application accesses to the database and writes the results the application has got from the simulation. The application has to write these three vectors in the database:

$$\left.\begin{aligned} name &= [name_1 \ldots \quad name_i \ldots \quad name_n] \\ value &= [value_1 \ldots \quad value_i \ldots \quad value_n] \\ quality &= [quality_1 \ldots \quad quality_i \quad quality_n] \end{aligned}\right\} where \quad i = 0,\ldots n$$

, and 'n' is the number of variables acquired.

These three vectors have the same meaning and the same structure as the three vectors read from the database.

3.4 APPLICATION IMPLEMENTED

Until now, in the points before it has told how to implement the automatic application, which simulates the wastewater treatment plant process and which is controlled by the STAR controller. There have been shown the structure of the program and the necessary blocks to get the purpose. In these last points of the chapter is presented the code programmed and the graphic interface of the application, so it will be easy to modify and to improve the code, it means, to update the code for future versions. Right now, the version of the application developed is a prototype and it is able to do all requirements which the company was asking for.

In the point below it is show the application, how it works and the code. It is divided in two points; the first to expose the application and the last to explain the code.

3.4.1 Application Interface

As all application software, it is needed a user interface to let the user manages the program. The user interface has been developed with GUIDE. A brief introduction about GUIDE is made in "*ANNEX II*", and to know more about the code implemented to program the application is recommended to consult "*ANNEX II*". The purpose of this point is to present the user interface developed.

Due to the simple actions that the user has to realise to run this application, the user interface is very simple. Bellow is shown in *figure 19* the user interface developed:

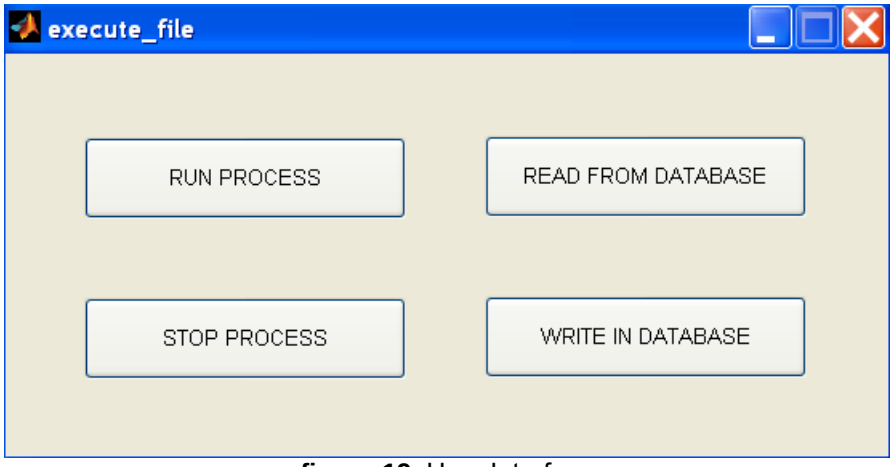


figure 19: User Interface

The user interface has four buttons, two main buttons to run and to stop the program and another two to check that the synchronization is working right. The use of each button is:

- **"RUN PROCESS":** Push this button to start the application. This button executes the file which contains all the code with the "initialization block" and the "main loop", so once the application is opened and the user press the button "RUN PROCESS" it is not necessary to do any action in the program, it will run until the user presses "STOP PROCESS", the user closes the application or the STAR controller makes the application to go out of the "main loop" using the control variables (*aux_sim* and *aux_loop*).
- **"STOP PROCESS":** The process can be stopped from the controller or from the user interface. To stop it from the user interface the user just has to push the button "STOP PROCESS" and the program will go out of the "main loop".

The other two buttons are programmed to emulate the STAR controller only with the synchronization of the simulation. As the application has been developed to have **double synchronization**, there are two buttons, one to check the synchronization before reading from the database and another to check the synchronization before writing data in the database.

- **"READ FROM DATABASE":** When the application is running it is waiting until the controller let it to access to the database before doing the simulation of the plant model. This button changes the value of the variables for the synchronization as it was the STAR controller, so if the application goes on with the simulation it means it is reading right the control variables. It is a good way to check if it is failing the Ethernet connection or the application if there is no synchronization.
- **"WRITE IN DATABASE":** This button has the same function as the one explained before, but for the second synchronization. If we implement the program with single synchronization it is possible to use the same user interface, but this button doesn't have to be used, or it has to be removed.

3.4.2 Program code

In the following lines it is detailed all the code implemented to program the application described along this chapter. The lines in red are comments introduced within the program code, so the compiler doesn't take care about these lines to execute the application code.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   MAIN PROGRAM TO SIMULATE THE WASTE WATER TREATMENT   %%%
%%%                        PLANT                            %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The first part that the program executes is the initialization functions. These functions are used once each time that the button "START PROCESS" (see point 3.4.1 "application interface") is pushed to start the application. The functions used in this part are shown bellow.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% INITIALIZATION %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Here the program calls the Matlab file "INIT.m". Inside this file all the variables and constants needed to the simulation of the process are defined. After, the program simulates the plant model to get the initial conditions of the state variables with the command "sim('name of the file of the plant model')". The last line is to define the vector 'x' to store the values of the initial values of the state variables. This part can be called **"model parameters initialization** and **model state variables initialization"** (see point "3.3.1 Initialization" described before).

```
%Initialize parameters-----------------------------------------------------------------------------------
INIT; %file .m to initialize all parameters of the model

% simulate plant to steady state
sim('plantmodel');
x=xInitial;
```

The following part is to initialize the control parameters. These parameters control the simulation and the main loop. If the variable "aux_loop" is equal to the value one the condition to go in the loop is complained and the application will run. The other variable is "aux_sim", and if it is equal to the value one the main loop is running but there is no simulation of the wastewater treatment process model. Thus, the application has the possibility of stopping the simulation if the controller doesn't want to do it.
The code initializes the value of both variables to one, so the application always starts executing the "main loop" and the simulation of the process. After the first time that the "main loop" is executed, the program read again the value of the "control variables" and it decides if the simulation or the main loop has to stop. In the point where it has been explained before this part was called **"control parameters initialization"**.

```
%Initialize control parameters
control_value = [1 1];
```

```
controlloop(control_value);

%read control parameters
[aux_loop aux_sim] = control_check; %we are reading control.xml
```

NOTE: check the "APPENDIX II" to read more about the functions "*controlloop*" and "*control_check*". About how the application is connecting with the database, read the next chapter "*chapter 4*".

To control the synchronization with the STAR controller have been defined two variables: 'readV' and 'writeV'. The controller will change the value of these variables to choose if it needs the application read data from the database or the applications writes in the database. In the initialization the values chosen are ***readV****=0* and ***writeV****=1*, so in the first synchronization at the beginning of the "main loop", the application will wait until the controller chose the "read" option, it means, ***readV****=1* and ***writeV****≠1*. This part is called **"synchronization parameters initialization"**.

```
%variables to synchronize with the controller
readV = 0; %when it has to read the XML file
writeV = 1; %when it has to write the XML file
```

This last part of the initialization is used to initialize the value of one variable used to read the data from the database. It is explained in detail in the following chapter.

```
% initialization variable to read from database
datatag=['data'];
```

Now, all the variables of the model and the variables of the program (synchronization variables and control variables) are initialized and the program executes the main loop to simulate the plant process each time STAR controller requires it.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% MAIN LOOP %%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

At the beginning of the "main loop", the first thing the code is doing is to check the value of the control variable "aux_loop" (it has been initialized to the one value before). If the value is the one expected the "main loop" starts running.
After there is a conditional instruction to check the other control variable: "aux_sim". If the value is not the expected the "main loop" executes itself but there is no simulation. As "aux_sim" has been initialized to unity value this first time that the "main loop" is executed the simulation will run.

```
while aux_loop == 1
  %variable used in the application to stop the simulation
  if aux_sim ~= 1
    disp('simulation stopped')
    pause(5);
  end
```

```
%process execution. I can stop the simulation if aux_sim not equal 1
if aux_sim = = 1
```

The following step once the control variables are checked is the synchronization. For the synchronization there are two possibilities: **single synchronization** and **double synchronization**. As it is shown in the code bellow, the implementation is the same in both cases, but there is one difference, in "single synchronization" there is one more line of code (code inside keys). As well as in "double synchronization" is the controller the one which changes the value of the synchronization variables always (there is one variable each time the controller synchronizes, when reading and when writing), in "single synchronization" it is only used the one variable ("readV"), because there is only one synchronization, when the application is going to read from the database. So, when the controller changes the value to the variable "readV" to indicate it is possible to read data from the database, the application has to reset this variable after going out of the "synchronization loop" in order to avoid that the application starts to simulate the process out of control because it read "readV" always as set value.
The problem of the "single synchronization" is that the controller can't reset the synchronization variable "readV". With the "double synchronization" this problem disappears. There are two synchronization variables: "readV" and "writeV". So, the first condition to synchronize is to have ***readV***=*1* and ***writeV***≠*1*. After, the application will wait until the second condition: ***readV***≠*1* and ***writeV***=*1*. It is evident is impossible to lose the control of the "main loop". The controller is the one which changes the value of the synchronization variables, and like there are two stops along the "main loop" to synchronize the application will always simulate the plant model once each period. Below is shown the code for both cases: "single synchronization" and "double synchronization".

CASE SINGLE SYNCHRONIZATION

```
%Synchronize with the server--------------------------------------------------------------------------

while readV ~= 1
  %we wait for the signal of the server, so we can synchronize each 6 minutes
  [readF writeF]=synchronizer;
  if (readF==1) & (writeF~=1)
     readV=1;
     writeV=0;
     break
  else
     readV=0;
     disp('we can not read now')
  end
  pause(15); %we wait 15 seconds until read again the file
end

{ %the variable is rise to 0 to stop the loop when it has to read data again
readV=0; }
```

CASE DOUBLE SYNCHRONIZATION

```
%Synchronize with the server--------------------------------------------------------------------------

while readV ~= 1
```

```
    %we wait for the signal of the server, so we can synchronize each 6 minutes
    [readF writeF]=synchronizer;
    if (readF==1) & (writeF~=1)
       readV=1;
       writeV=0;
       break
    else
       readV=0;
       disp('we can not read now')
    end
    pause(15); %we wait 15 seconds until read again the file
end
```

NOTE: the function used to synchronize with the controller is explained the next chapter "*chapter 4*".

Once the synchronization has taken part the program accesses to the database to read the data. First are defined the three vectors where the values of the variables are going to be stored: "nameR" to store the names of the variables, "valueR" to store the values and "qualityR" to store the quality of the value read. The function "readXML" used to access to the database is discussed in the following chapter *"chapter 4"*. The code implemented in the main code is presented below:

```
%Read from database----------------------------------------------------------------------
nameR=[''];
valueR=[ ];
qualityR=[ ];
[nameR,valueR,qualityR]=readXML(datatag);  %file .m where we must read the data
                                           from database
```

The next step after reading the data from the database is to execute the "Matlab Algorithm", which is divided in three functions shown below:

MATLAB ALGORITHM

The first thing the Algorithm has to do is to translate the data read into a format useful for the model is going to be simulated in Simulink. The function "read_data" is responsible to do this task. An example of how to implement this function is detailed in "APPENDIX II".

```
%Initialize parameters with the new data-------------------------------------------------
read_data; %file .m to initialize with the new data
```

Following the translation of the data read is executed the simulation of the process. It is done in two steps; first the options for the simulation are set with the function ***simset***, and finally the model is simulated with the function ***sim***. Both functions are standard functions from Matlab, for more information about them it is recommended to use the help of the program.

```
%Simulate the model---------------------------------------------------------------------
options=simset('InitialState',x(end,:),'Solver','ode45');
[t,x]=sim('plantmodel',[t(end) t(end)+6/60/24],options);)
```

The last step of the Algorithm is to translate the results of the simulation into the three vectors needed to send the data to the controller. To get this task is used the function "wirte_data". As before with the function "read_data", there is an example of how implementing this function in the "APPENDIX II".

```
%Write the new data in vectors form work space-----------------------------------------
write_data; %function to write the data in three vectors
```

CASE DOUBLE SYNCHRONIZATION

This code is only present in the program when it is implemented the double synchronization. If it is used the single synchronization it is not necessary.

```
%Synchronize with the server-----------------------------------------------------------------------
 while writeV ~= 1
    %we wait for the signal of the server, so we can synchronize each 6 minutes
    [readF writeF]=synchronizer;
    if (writeF==1) & (readF~=1)
       writeV=1;
       readV=0;
       break
    else
       writeV=0;
       disp('we can not write now')
    end
    pause(15); %we wait 15 seconds until read again the file
 end
```

The outputs of the "Matlab Algorithm" are three vectors: "name", "value" and "quality". These three vectors have the same structure than the vectors read from the database, but they have stored the values of the results of the simulation. The function "writeXML" has been implemented to send the data to the database. The function is explained in "*chapter 4"*.

```
%Send data to the database-------------------------------------------------------------------
n=length(value);
writeXML(name,value,quality,n); %we write all data i an XML file
end
```

The last step of the program is to check the value of the control variables in order to know if the "main loop" has to go on running or the controller has ordered to stop with the simulation.

```
%it tests if the simulation or the main control loop has to stop or not
%read variable "aux_loop" and variable "aux_sim"
[aux_loop aux_sim] = control_check; %we are reading control.xml

end
```

> **NOTE**: check the "APPENDIX II" to read more about the function "*control_check*".

CHAPTER 4

Data Exchange

The main application implemented has been shown in the chapter before, what is its purpose, the implemented code, the user interface, all the features. And it was said that one of the most important things is the communication with the STAR controller to: synchronize the application and the controller, to read data from the database, write the data and stop the application from the controller.
Now, how the application interacts with the controller is detailed in this chapter, it means, the code of all the functions used to transmit the data between the controller and the application or in the other way around, and the files where the variables are stored.

4.1 Introduction

Talking again about the purpose of this thesis, in the figure below (*figure 20*) is shown the diagram of the project developed taking care only about the communication between all of the devices involved:

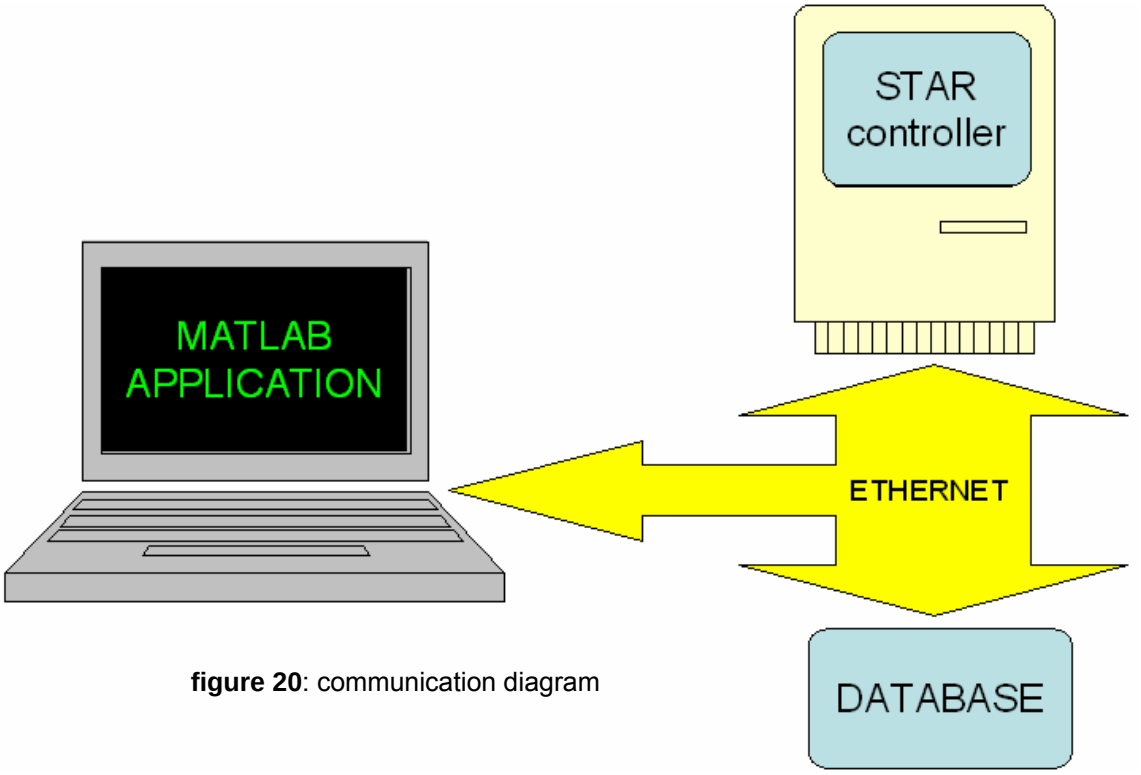


figure 20: communication diagram

It is possible to see that there are three main devices:

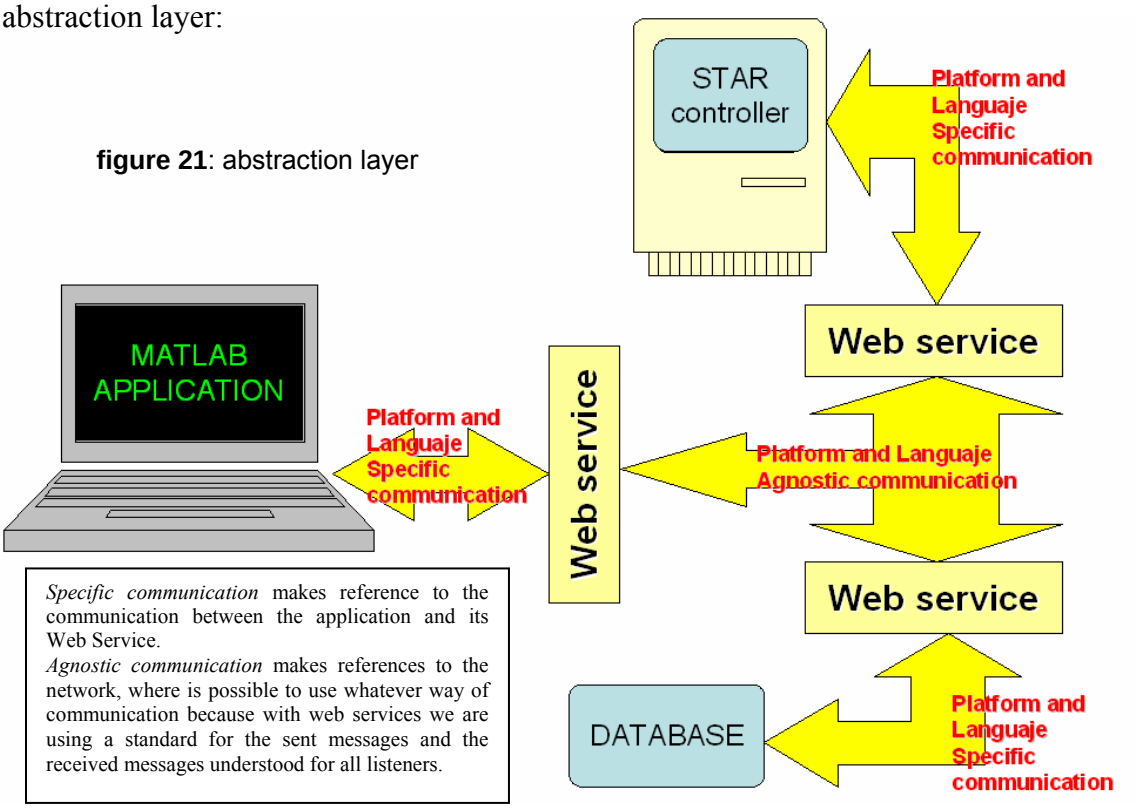
- ***STAR controller:*** which is connected directly with the wastewater treatment plant and it receives the measurements of the sensors that there are installed.
- ***Database:*** which stores the different values that the controller gets from: the sensors, the control action applied, the time, the data, at last, all data that needs to be analysed.
- ***Matlab application:*** Application developed in this thesis which simulates the wastewater treatment plant and gives the results to the controller, which compares the real results with the expected results got in the simulation process.

And, as it was said before, these three devices are connected using the Ethernet protocol. So, the functions implemented in the main code shown in *chapter 3* (point *3.4.2 program code*) must be able to connect with the controller and the database using an Ethernet protocol.

Nowadays, the different platforms connected through Ethernet are using different application code: the Matlab application is running under Windows Operative System, the database is running in the same interface or another and the STAR controller is a specific device able to use Ethernet network but not running under Windows Operative System. The best way to connect them is using **Web Services** (see "*APPENDIX II*" to better understanding). **Web Services** provide an abstraction layer between the client

application and the code application, it means that there is a layer programmed between the code application and the network, so it doesn't mind if the code application of the different platforms is not the same, because after the abstraction layer it will be possible communicate all of them. This abstraction layer translates the message into a standard that can be read by the other client application. In the figure below is represented the abstraction layer:

figure 21: abstraction layer



Specific communication makes reference to the communication between the application and its Web Service.
Agnostic communication makes references to the network, where is possible to use whatever way of communication because with web services we are using a standard for the sent messages and the received messages understood for all listeners.

To program the *Web Services* is possible to use an amount of codes to program the web services, which makes difficult the choice: java, Visual C, Basic, C++, Perl, etc. By the other side there are a lot of possible standards that can be chosen as XML protocol, XKMS, SAML, XML-Dsig, XML-Enc, XSD, P3P, WSFL, Jabber, ebXML, SOAP, etc. But the one used by the company to communicate the *database* and the *STAR controller* is **SOAP protocol**, which is going to be introduced in the next point.

4.1.1 What SOAP is

"*Simple Object Access Protocol*", that is SOAP. Let define in a few lines SOAP as a standardized packaging protocol used to transmit and receive messages by applications through Ethernet.

SOAP is an application of the XML specification. It means that the basis of SOAP is formed by XML messaging (applications exchange information using XML documents).

The best thing of SOAP protocol is that is based in XML, and XML is not tied to any application, operative system, programming language or whatever other technical implementation. With XML is possible to share information using only a message encoded understood by the applications, which are going to transmit and receive message.

This step of the project design has been made by the company, so this introduction to SOAP protocol is not going further (to know more check the "*APPENDIX III*" and the bibliography). The company has developed an intermediated application which receives and transmits the messages into SOAP protocol through the network. Now the diagram of the communication is the following:

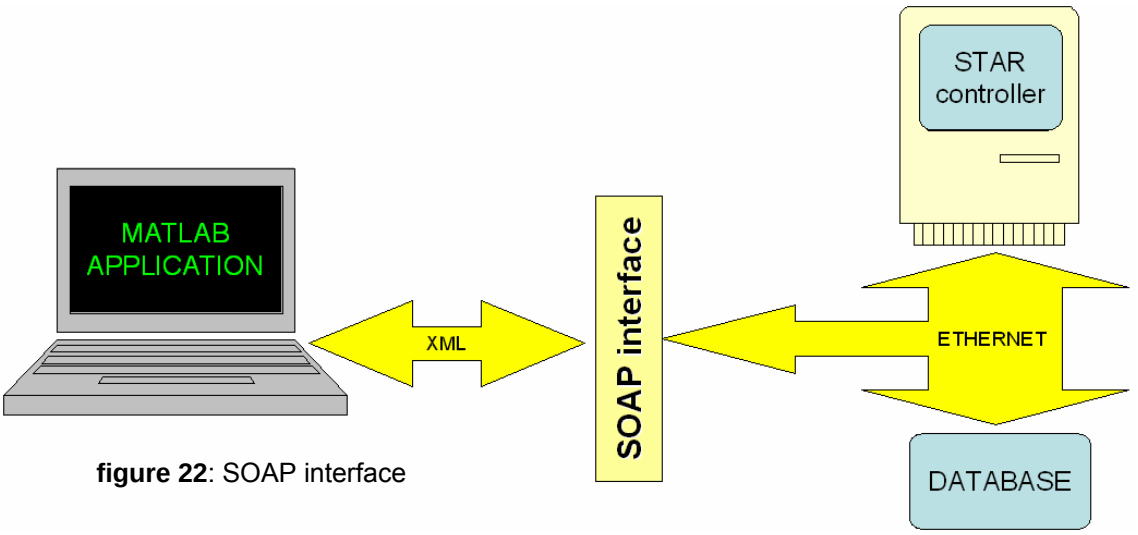


figure 22: SOAP interface

Beginning from here, the code implemented has to be able to read and translate the information that is going to arrive as XML data file. Thus, the communication between the application developed with Matlab and the other devices is reduced to read and write XML files.

4.1.2 XML communication files

Definitely, the Ethernet communication between the application developed in Matlab and the STAR controller and the database has been reduced to the task of writing and reading XML files. Hence, to write an XML file will be equivalent than to send a message to the controller or to the database, and to read an XML file will be equal than to receive a message.

It takes us to think about another question, if to write or to read an XML file is like writing or reading a message, it will be necessary to use one XML file for each kind of message, it means, one file for control variables, another for synchronization variables, another to read data from the database and at last one for writing the results from the simulation. The four XML files that store and are used to transmit and receive the information are the following files:

- ***"database.XML":*** the dates from the database, that the application needs to use in the simulation, are stored within this XML file. After reading the file, these dates are stores in three vectors with the values of the variables. The structure of the file is:

```
<?xml version="1.0" encoding="utf-8"?>
<dataexchange>

   <data>
        <name>name(1)</name>
        <value>value(1)</value>
        <quality>quality(1)</quality>
   </data>
.
.
.
   <data>
        <name>name(n)</name>
        <value>value(n)</value>
        <quality>quality(n)</quality>
   </data>
</dataexchange><!--this is a comment-->
```

- ***"results.XML":*** the Matlab application writes the results of the simulation in this file. The controller will read the values stored here. The structure of the file is:

```
<?xml version="1.0" encoding="utf-8"?>
<dataexchange>

   <data>
        <name>name(1)</name>
        <value>value(1)</value>
        <quality>quality(1)</quality>
   </data>
.
.
.
   <data>
        <name>name(k)</name>
        <value>value(k)</value>
        <quality>quality(k)</quality>
   </data>
</dataexchange><!--this is a comment-->
```

- ***"synchro.XML":*** the value of the variables used for the synchronization is stored within "*synchro.XML*". The controller writes the values and the Matlab application reads the content stored inside the file. The structure of the file is:

```
<?xml version="1.0" encoding="utf-8"?>
<synchronize>
   <sincro>
      <read>"readF value"</read>
      <write>"writeF value"</write>
   </sincro>
</synchronize>
```

- ***"controlFILE.XML":*** This XML file has the same function as "*synchro.XML*", but this file stores the value of the control variables, variables that are modified by the controller and read by the Matlab application. The structure of the file is:

```
<?xml version="1.0" encoding="utf-8"?>
<control>
   <data>
      <loop>"aux_loop value"</loop>
      <simulation>"aux_sim value"</simulation>
   </data>
</control>
```

NOTE: to know more about the variables that are stored in each XML file and for what is using the application each variable, it is recommended to check the point "*3.4.2 Program code*" in "*chapter 4"*. The function of each variable is not the purpose of this chapter.

The functions implemented to modify the files shown before within the application code are:

"**control_check**": to read "*controlFILE.XML*". This function is described in the point "*II.2.2 "control_check" code*" in "*APPENDIX II*".

"**controlloop**": to write "*controlFILE.XML*". This function is described in the point "*II.2.1 "controlloop" code*" in "*APPENDIX II*".

"**synchronizer**": to read "*synchro.XML*"

"**writeXML**_synchro": to write "*synchro.XML*". This function is described in the point "*II.1.1 "writeXML_synchro" code*" in "*APPENDIX II*".

"**readXML**": to read "*database.XML*"

"**writeXML**": to write "*results.XML*"

4.2 Data-flux in the communication

Now we have arrived to this point is evident that the problem of the communication of the application has been reduced to implement two functions: the function which reads the XML file and the function which writes the XML file. Because of there are four XML files, there are necessary more than one function to read and one to write, it means, it would be needed two functions for each XML file. Remembering the file "*database.XML*" only needs to be read by the application but not to be written, and "*results.XML*" to be written by the application but not to be read. The number of necessary functions is six; three functions to read and three functions to write (the six functions are shown up in point "*4.1.2 XML communication files*").

It seems better to try to implement a general function to read and another to write, and use both of them for reading and writing the four XML files. The problem is that it supposes to define the functions with a lot parameters as a variables, then, each time is going to be read or write an XML file these parameters have to be introduced into these functions. Thus, as the functions are not large (the largest one near fifty code lines) and the number of variables in the Matlab work-space can be reduced, at the moment the

solution with six functions is preferred. It has to be considerer that the code of the functions to read and to write would be increased considerably if we try to write or read four different structure message with the same functions.

Despite to have six functions, all of them follow the same data flux shown in the *figure 23* ("figure 23.A" represents the data flux when writing the XML file and "figure 23.B" represents the data flux when reading the XML file):

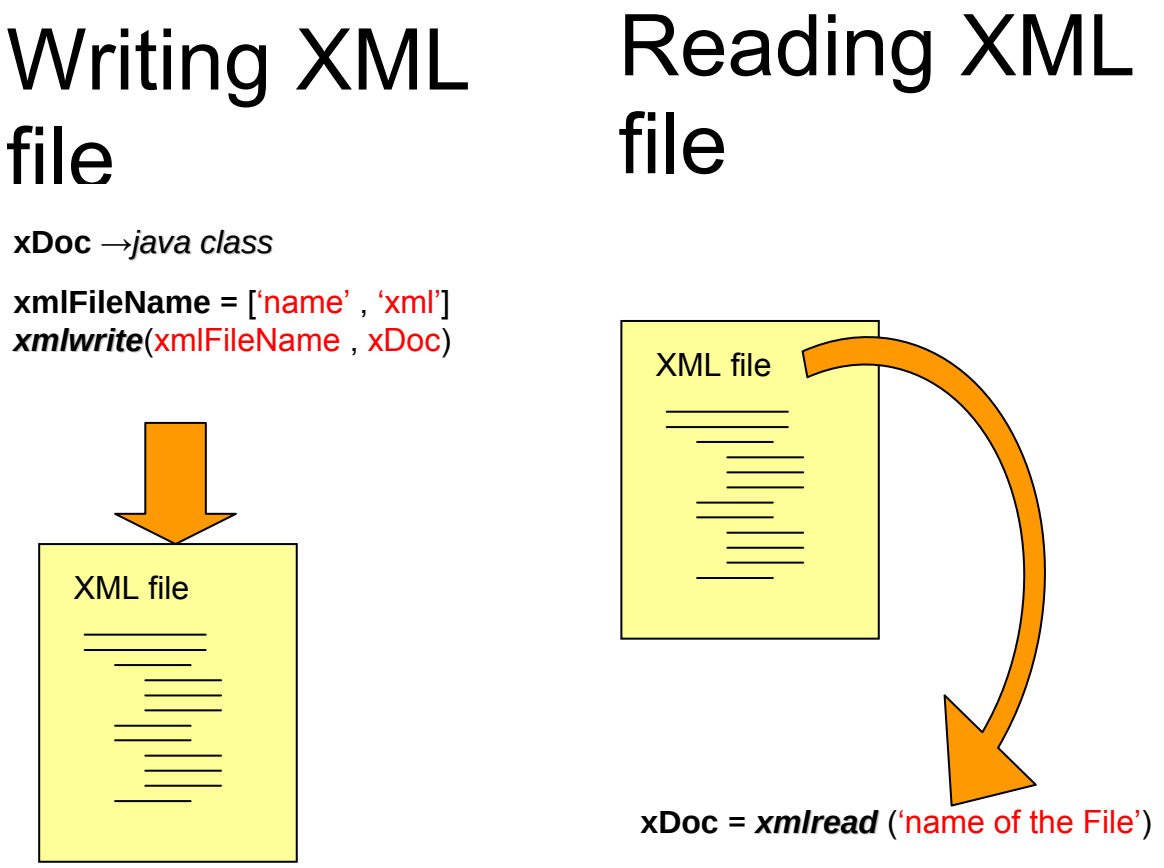


figure 23: data-exchange. A) write data into "*file.XML*", B) read data from "*file.XML*".

The diagrams shown in *figure 23* represent the procedures used to read and write XML files. There is one common step in both diagrams (*figure 23.A* and *figure 23.B*), this step is the definition of the variable "xDoc", variable used to store the values contented in the messages expressed as a XML files. The variable is the type of "***org.apache.xerces.dom.DocumentImpl***" (check "*APPENDIX III*" to know more about it), which is a class needed by the command "**xmlwrite()**" to create the XML file, and is the class that the command "**xmlread()**" returns when it is reading the XML file.

The steps followed to write an XML file are:

1. To create the variable "xDoc" of type "***org.apache.xerces.dom.DocumentImpl***".
2. To create the XML file with the variable **xmlFileName.**
3. To use the command "**xmlwrite(***xmlFileName, xDoc***)**" to generate the XML file.

The steps for reading the XML file are:

1. To use the command "**xmlread(***name of the XML file***)**" to store the XML file inside the variable called "xDoc" of type "***org.apache.xerces.dom.DocumentImpl***".
2. To scan the variable "xDoc" and extract the information stored.

4.3 Data-exchange with the database

In the points above it has been explained the kind of communication used between the Matlab application and the others devices connect to the network (STAR controller and database), the protocol used and the data-flux in the functions implemented to make possible this communication. Here the functions used to get data from the database and to write data inside the database are exposed explaining all the code implemented step by step.

4.3.1 Write the data in "results.XML"

The file "*results.XML*" is the XML file that the SOAP interface developed by the company sends as a message to the database with the results of the simulation of the Wastewater Treatment process. In agreement with the company the structure of the XML file has to be the one shown in the point "*4.1.2 Communication files*" and below these lines:

```
<?xml version="1.0" encoding="utf-8"?>
<dataexchange>

   <data>
        <name>name(1)</name>
        <value>value(1)</value>
        <quality>quality(1)</quality>
   </data>
.
.
.
   <data>
        <name>name(k)</name>
        <value>value(k)</value>
        <quality>quality(k)</quality>
   </data>
</dataexchange><!--this is a comment-->
```

As it can be appreciated, there is a number of *'n'* elements (children) called *"data"* that have three sub-elements: *"name"*, *"value"* and *"quality"*. Each element represents one variable with three properties, the name of the variable to know which variable is, the value of the measurement of this variable in the wastewater treatment plant stored in the database and the quality of this measurement.

The file *"database.XML"* that has to be read by the application has to follow the same structure than the XML file *"results.XML"*.

In the following lines is explained the code implemented to write the XML file as it was shown in the lines above in this point.

"writeXML" code

```
function archivoXML=writeXML(name,value,quality,n)
```

This first part of the code is the one which makes reference to the step one mentioned in "*4.2 Data-flux in the communication*", to write an XML file: to create the variable "xDoc" of type "***org.apache.xerces.dom.DocumentImpl***".
The variable created is *"docNode"*.

```
% Create a sample XML document.
%program
docNode = com.mathworks.xml.XMLUtils.createDocument('dataexchange');
docRootNode = docNode.getDocumentElement;
```

Inside the same step, as the function does not know how many *"data"* children has to create, it goes in a loop that will do as much children as long was the vector *"name"* (remember that the vectors "name", "value" and "quality" have the same size).

```
for i=1:n
   dataElement = docNode.createElement(sprintf('data'));
   nameElement = docNode.createElement(sprintf('name'));
   valueElement = docNode.createElement(sprintf('value'));
   qualityElement = docNode.createElement(sprintf('quality'));

   nameElement.setTextContent(sprintf('%s',name(i,:))); %name of the
                                                        variable
   valueElement.setTextContent(sprintf('%2.2f',value(i))); %value of the
                                                        variable
   qualityElement.setTextContent(sprintf('%2.2f',quality(i))); %quality of
                                                        the variable
```

The first part of the loop is to create the child with the three sub-elements (the lines above), and the last lines of the loop are to close each child once it has been created.

```
   dataElement.appendChild(nameElement);
   dataElement.appendChild(valueElement);
   dataElement.appendChild(qualityElement);
   docRootNode.appendChild(dataElement);

end
```

The second step is: to create the XML file with the variable **xmlFileName.** To do it is defined a structure with two elements: the route where is going to be created the file and the extension of the file (note that if it is only defined the name and not the route the function "*xmlwrite*" will create the XML file in the current directory).

```
% Save the sample XML document.
xmlFileName = ['results','.xml']; %we have the file in our current
                                    directory
```

Finally it is executed the function "***xmlwrite***" to create the XML file and finish with the third step: to use the command "**xmlwrite(***xmlFileName, xDoc***)**" to generate the XML file.

```
xmlwrite(xmlFileName,docNode);
```

> **NOTE**: All the functions implemented to write an XML file follow the same structure that has been exposed above, so all the comment in this point is not going to be told anymore along this thesis, except if it is necessary.

4.3.2 Read the data from "database.XML"

The function "***readXML***" is the one which will read the file "*database.XML*", where are stored the data that the database sends to the application. The structure of the XML file that is going to be read has to be the same exposed before for "*results.XML*".

"readXML" code

As it has been said in the point "*4.2 Data-flux in the communication*", the function read the XML file in two steps:

1. To use the command "**xmlread(***name of the XML file***)**" to store the XML file inside the variable called "xDoc" of type "***org.apache.xerces.dom.DocumentImpl***".
2. To scan the variable "xDoc" and extract the information stored.

```
function [a,b,c]=readXML(data)
```

Definition of the auxiliary variables:

```
dataname=[''];
datavalue=[];
dataquality=[];

infoLabel = 'Plot Tools';
infoCbk = '';
itemFound = false;
```

The first step is made here (to use the command "**xmlread(***name of the XML file***)**"), when the information stored in the XML file passes to the variable *"xDoc"*. The function used to define the variable is ***"xmlread"***, which needs one parameter; the name of the file with the extension. Thus, the function looks for the XML file in the current directory, if the file is in another directory the parameter included will be the name of the file with the extension and the route: C:/program files/application/files/database.xml, for example.

```
xDoc = xmlread('database.xml');
```

Here starts the second step. Previous to go in the loop used to scan the class where is stored the XML message, it is define the variable "*allListItems*". This variable stores the number of children "*data*" in the message that is the same as the number of variables stored in the message.

```
% Find a deep list of all list item elements.
allListItems = xDoc.getElementsByTagName(sprintf('%s',data(1,:)));
```

Once it is known the number of children "*data*" (check the structure of the message shown before) the loop scans the class storing all variables in a vectors to work in the application.

```
% Note that the item list index is zero-based.
for i = 1:allListItems.getLength          %it's looking for all items with
                                             <data(i,:)>
    thisListItem = allListItems.item(i-1);
    childNode = thisListItem.getFirstChild;
```

With this "***while***" loop we ensure the function scans the child until it has read all the sub-elements inside each child.

```
    while ~isempty(childNode)
        %Filter out text, comments, and processing instructions.
        if childNode.getNodeType == childNode.ELEMENT_NODE
            % Assume that each element has a single
            % org.w3c.dom.Text child.
            childText = char(childNode.getFirstChild.getData);%get the
                                                               data text
```

Inside each child there are three sub-elements, the "***switch***" is used to store the value of each sub-element inside the vector that it has to be stored in.

```
            switch char(childNode.getTagName)
                case 'name';
                    %itemFound = strcmp(childText, infoLabel);
                    dataname(i,:)=sprintf('%s',childText);%to get the data
                                                           name
                case 'value' ;
                    %infoCbk = childText;
                    datavalue(i)=sscanf(childText,'%f');%to get the data
                                                        value
                case 'quality' ;
                    dataquality(i)=sscanf(childText,'%f');%to get de data
                                                          quality
            end
        end  % End IF
        childNode = childNode.getNextSibling;
    end  % End WHILE

    if itemFound
        break;
    else
        infoCbk = '';
    end
end  % End FOR
```

Finally the three vectors got from the scanning of the XML file are stored inside the vectors '*a*', '*b*' and '*c*' that the function is going to return to the Matlab work-space.

```
%read value
 a=dataname;
 b=datavalue;
 c=dataquality;
```

4.4 Synchronization

As it has said before, the synchronization is made through the communication between the application and the STAR controller. The controller writes the value of the variables used for the synchronization in the file "*synchro.XML*", and the application reads the value of these variables from the XML file.

> **NOTE**: there is a function implemented to write the "*synchro.XML*" file, but it is used only in the initialization part of the application's main code (see "*CHAPTER 3: Application design*" point "*3.4.2 Program code*"). This function is used almost in the *emulator to check the synchronization code* of the application. To read more about this function check the point "*II.1.1 "writeXML synchro" code*" in "*APPENDIX II*".

4.4.1 Read the data from "synchro.XML"

The function implemented to read the synchronization XML file ("*synchro.XML*") is "***synchronizer***". The code is the same used to implemented "***readXML***" but with the particularities needed to decode the message written by the controller inside "*synchro.XML*".

"synchronizer" code

As it has been said in the point "*4.2 Data-flux in the communication*" and mentioned before, the function read the XML file in two steps:

1. To use the command "**xmlread(***name of the XML file***)**" to store the XML file inside the variable called "xDoc" of type "***org.apache.xerces.dom.DocumentImpl***".
2. To scan the variable "xDoc" and extract the information stored.

```
function [readF writeF]=syncrhronizer

readdata=[];
writedata=[];

infoLabel = 'Plot Tools';
infoCbk = '';
itemFound = false;
```

FIRST STEP:

```
xDoc = xmlread('synchro.xml');
```

SECOND STEP:

```
% Find a deep list of all listitem elements.
allListItems = xDoc.getElementsByTagName('sincro');

% Note that the item list index is zero-based.
for i = 1:allListItems.getLength          %it's looking for all items with
                                            <data(i,:)>
    thisListItem = allListItems.item(i-1);
    childNode = thisListItem.getFirstChild;

    while ~isempty(childNode)
        %Filter out text, comments, and processing instructions.
        if childNode.getNodeType == childNode.ELEMENT_NODE
            % Assume that each element has a single
            % org.w3c.dom.Text child.
            childText = char(childNode.getFirstChild.getData);%get the
                                                               data text

            switch char(childNode.getTagName)
                case 'read';
                    readdata(i)=sscanf(childText,'%f');%to get the
                                                       readdata value
                case 'write' ;
                    %infoCbk = childText;
                    writedata(i)=sscanf(childText,'%f');%to get the
                                                        writedata value
            end
        end  % End IF
        childNode = childNode.getNextSibling;
    end  % End WHILE

    if itemFound
        break;
    else
        infoCbk = '';
    end
end  % End FOR
```

Finally the function stores the value of the synchronization variables into two variables that will return to the Matlab work-space:

```
%read value
 readF=readdata;
 writeF=writedata;
```

CHAPTER 5

Modelling Batch Processes with Matlab

Along the past chapters the design and implementation of an automatic application in Matlab work-space has been discussed. The aim of this application is to simulate a model of the wastewater treatment plant that is being controlled by the STAR controller. The tool package used by Matlab to simulate processes and all kind of systems is Simulink. Therefore is needed a model implement within Simulink.
In this chapter is exposed one possible way to implement bach processes in Matlab, starting from one biological model implemented by the department of "Industrial Electric Engineering and Automation (IEA)".

5.1 Introduction

In "*CHAPTER 1: Introduction*" was made a presentation about the aims of the thesis. The main aim was to implement an application to simulate a model of the wastewater treatment plant controlled by the STAR controller. So that, the main goal is to simulate a model, it means, a model of the wastewater treatment plant is needed.

The application design in the chapters before has to be able to simulate the model implemented. The easiest way to get it is to implement the "wastewater treatment plant model" in Simulink, a tool of Matlab that shares the same work-space as Matlab, so there will be no problems to the data-exchange between the model implemented and the application.

Otherwise, to decide the application to implement the model is not the only problem. There are a lot of solutions when a model of a wastewater treatment plant has to be implemented; different biological processes have to be modelled, a model for settling and clarification vessel, modelling hydraulics, and a long etc.

Here in this chapter the ***ASM1 model*** developed by a task group work in collaboration with the International Water Association (IWA, formerly IAWQ and IAWPRC) in 1983 and published in 1987 has been chosen to model biological processes. For modelling settling and clarification is used the ***traditional one-dimensional layer settler model***. Both of them have been explained in "*chapter 2*", so this chapter is not going further in this point, to read more about it check the information in "*chapter 2*" or the bibliography.

Finally, the reader has to know that the model introduced in this chapter is not enough to simulate a complete wastewater treatment plant. The purpose of the chapter is to get a model to implement a standard block in the future, so it can be made a "wastewater model library" including more standard blocks ready to be used by other researchers. The block tries to include the necessary code to simulate the batch processes within a vessel (check the *chapter 2* or the bibliography to know more about the batch processes).

5.2 Model developed

About the introduction before, it id deduced that an extended model is needed in order to include "settling and clarification" and "biological processes" within the same model. It means that, if ***ASM1 model*** and ***traditional one-dimensional layer settler model*** are used, the new model developed must included both of them, with the particularity of using it as "*batch process model*", hence, this model developed will not be used in the typical configurations designed to model wastewater treatment plants as a continuous model with constant effluent.

The starting point is to use the "*traditional one-dimensional layer model*" explained in "*chapter 2*". The vessel is going to be divided in ten layers.

The second step is to include the equations of the "*ASM1 model*" in each layer to model the biological processes within them. So there are the general equations to model the process of settling and clarification and there are the particulate equations of the biological processes to model the biological processes of each layer. It means that it is assumed that the concentration of the suspended solids and the soluble particles inside one layer is constant and well mixed in that layer, so there are no elements with different concentrations within the same layer.

As it was said in "*chapter 2*", the ASM1 model is going to be extended to fourteenth state variables because the TSS (total suspended solids) is included.

All the things said until now are not new. The goal is only to join two models exposed before. But there are two particularities if we try to model batch process:

- The height now is variable, so is not a constant, it has to be included as a state variable, the fifteenth state variable in the model of each layer.
- There is a new input to model the flux generated by the turbines in the "*mixing phase*" (one phase that the batch process model has to have included, check "*chapter 2*"), this flux is called J_{down_up} and it has its maximum value in the "*mixing phase*" and it is equal to zero when the process is in another phase.

In the lines before it has been mentioned the "*mixing phase*". In "*chapter 2*", when we were talking about the batch process, the main characteristic was that this kind of model was not a continuous model with constant effluent, it was a discrete model with different phases. It means that the batch process model goes through several phases, to finish all the wastewater treatment in the same vessel, there are no needed several tanks and vessels, some of them for removing carbon or nitrogen and the others for settling and clarification, all the biological processes are taking part in the same vessel, but each one in a different phase.

The simplest batch process was presented in "*chapter 2*" and there are only four phases: "*filling phase*", "*mixing phase*", "*settling phase*" and "*draw phase*". The batch model described in this chapter has six phases: "*filling phase*", "*mixing phase*", "***reaction phase***", "*settling phase*", "*draw phase*" and "***waiting phase***".

The model for all the phases is the same, but the different is the value of the inputs, so that is one of the most important factors in the model, to decide which ones are the model inputs. For better understanding how a batch process plant works and which ones can be the inputs, the following figures shows the secondary treatment of one wastewater treatment plant with three vessels, working all of them as batch processes:

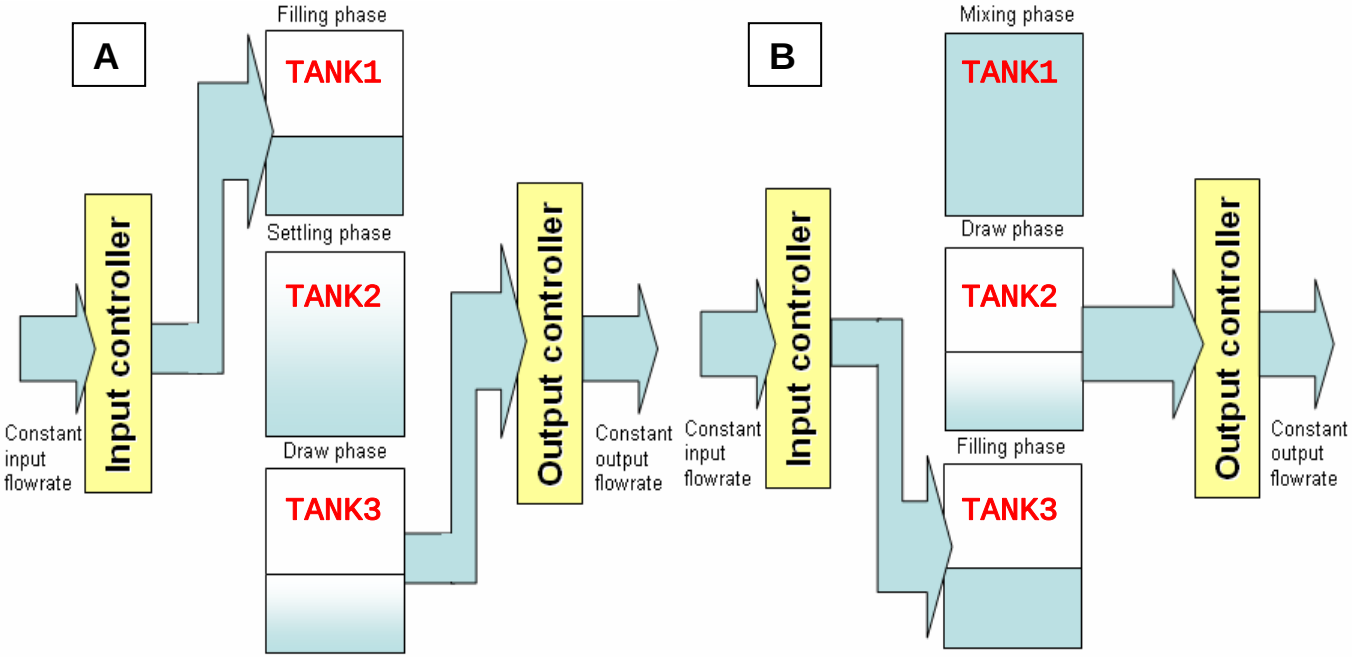


figure 24: Wastewater Treatment Plant with three vessels. The diagram **a)** (on the left) represents the model plant, and the diagram **b)** (on the right) represents the same model plant in another phase.

The model developed would represent the biological progress of one of the tanks of the wastewater treatment plant shown in "*figure 24*". As it is evident, in one wastewater treatment plant the input flowrate of wastewater and the output flowrate of treated water is constant. It means that are necessary more than one tank to model the plant, and the tanks must be synchronized to pass all of them through all the phases keeping the total effluent of the plant constant.

Looking the diagram of the "*figure 24*" can be deduced some of the inputs of the model. One input is the ***input flowrate***, another input is the ***output flowrate*** (the controller decides when the tank can be drawed), and the others are not evident. One is the ***extra flux*** needed to get the tank was well mixed in the filling phase and in the mixing phase. This flux represents the flux generated by the input flowrate in the filling phase and the one generated by the turbines in the mixing and reaction phase. The extra flux will be applied in both senses, up and down, so the concentration of suspended solids that is going down in the tank is going up too and the vessel is well mixed, characteristic common in these three first phases.
There are two inputs more in the model: the ***waste sludge flowrate*** and the ***constant of the dissolved oxygen concentration's reaction***. The first one is always constant, and the second one represents the increasing of dissolved oxygen in the biological phases (mixing phase and reaction phase). The inputs of the model are:

- Q_i: input flowrate
- Q_o: output flowrate
- J_{down_up}: extra flux in filling phase and the biological phases
- Q_u: waste sludge flowrate
- $K_l a$: constant of the dissolved oxygen concentration's reaction

5.2.1 Model diagram

The batch process model implemented has six different phases: "*filling phase*", "*mixing phase*", "*reaction phase*", "*settling phase*", "*draw phase*" and "*waiting phase*". Each phase represents one state of the process and it is characterised by the different values of the inputs defined in the point before: Q_i, Q_o, J_{down_up}, Q_u, and K_la. In the figure bellow (*figure 25*) is represented the main diagram of the batch process' phases and the order of progressing.

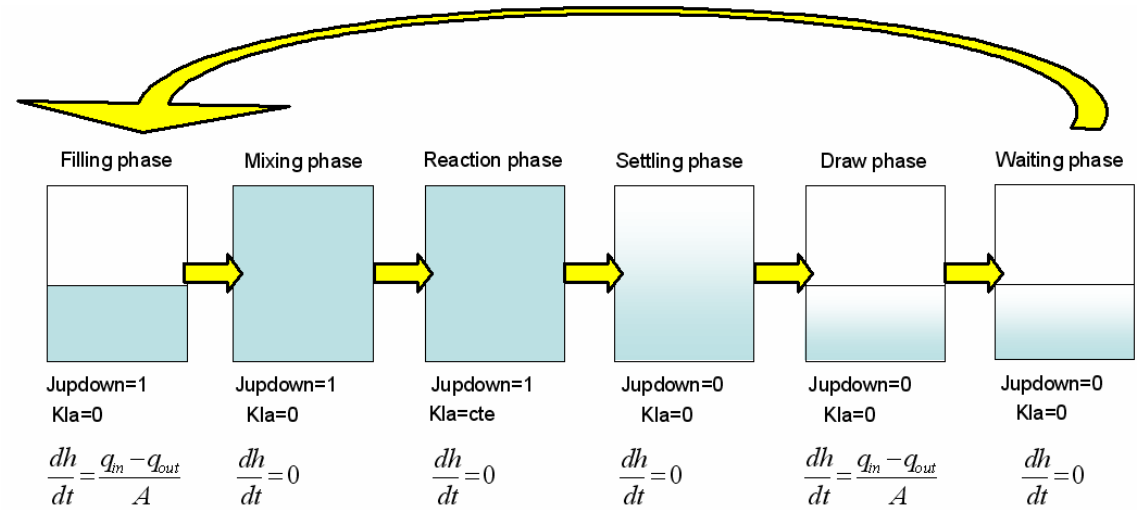


figure 25: Batch process model diagram

The input Q_u is not represented, but this input will be constant along all the phases, so its value is the same always. By the other side, in *figure 25* is shown the evolution of the new state variable *height* that depends of Q_i and Q_o.

It has to be said the particularity of the *waiting, phase*, it is not representing a reaction step in the evolution of the process, as can be the "settling and clarification", the "mixing" of wastewater. This phase is an auxiliary phase programmed to make easier the control of the different tanks in the wastewater treatment plant. It is known that the effluent of wastewater in the treatment plant is constant (see *figure 24* in the page before), so if there is one tank asking for input flowrate, and after another one asks for it, this second tank will have to wait until the other one has finished the *filling phase*.

With the output flowrate does not occurs the same because the *settling phase* can work as a waiting phase too. If there is one tank in the *draw phase* the next one can wait in the *settling phase* until the first one has finished.

Otherwise, the control has to find the best synchronization between all the tanks of the wastewater treatment plant to try to minimize the time spent in the *waiting phase*.

Now is going to be explained the particularities of each phase and shown the diagram of each one.

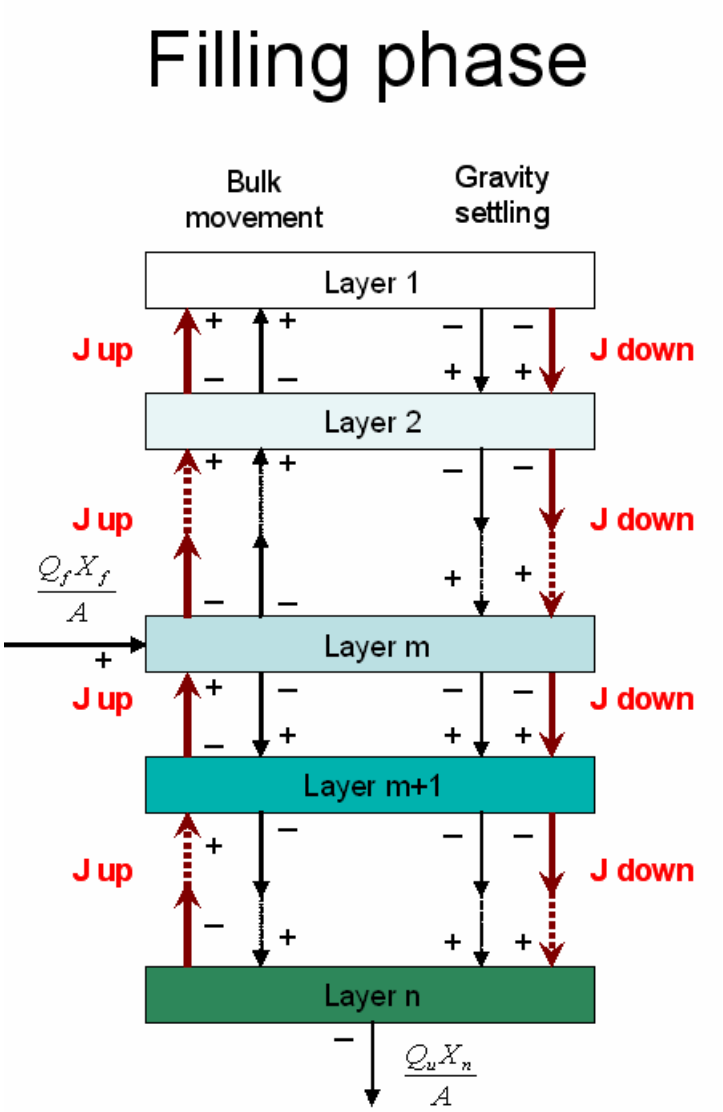


figure 26: Filling Phase

The *filling phase* is the first phase of the model, although the programmer of the model can chose the phase in initial conditions. It is assumed that the height of the layers have the minimum value (the level of the tank is the minimum level of wastewater) at the beginning of this phase.
The characteristics of the *filling phase* are:

- *Tank well mixed*. The value of the flux generated by the input flowrate (J_{up} = -J_{uown}) is bigger than the settling flux due to the gravity force and the bulk flux due to the waste sludge flowrate, both of them are despicable in front of the value of J_{up}.
- J_{up} = -J_{down} > 0.
- q_{out} = 0.
- q_{in} > 0.
- q_u > 0.
- $K_L a$ = 0, there is no reaction.

- $\frac{dh}{dt} = \frac{q_{in} - q_{out}}{A}$, where A is the area of the tank. The derivative of the height is positive which means that the height of the level is increasing. This is the main characteristic of the *filling phase*.

The two main characteristics of the *filling phase* are that the tank is well mixed and that the value of the derivative of the height is not zero.

This phase is active until the height reaches its maximum value, then it starts the following phase and the *filling phase* is finished.

The *mixing phase* is the next phase after the *filling phase*. There is no more input flowrate and the level of the wastewater is the maximum (maximum height). The principal characteristics are:

- *Tank well mixed*. The value of the flux generated by the input flowrate (J_{up} = -J_{uown}) is bigger than the settling flux due to the gravity force and the bulk flux due to the waste sludge flowrate, both of them are despicable in front of the value of J_{up}.
- $J_{up} = -J_{down} > 0$.
- $q_{out} = 0$.
- $q_{in} = 0$.
- $q_u > 0$.
- $K_L a = 0$, there is no reaction.
- $\frac{dh}{dt} = 0$, the derivative of the height is null because there is no input and output flowrate, and it is right because the height has a constant value.

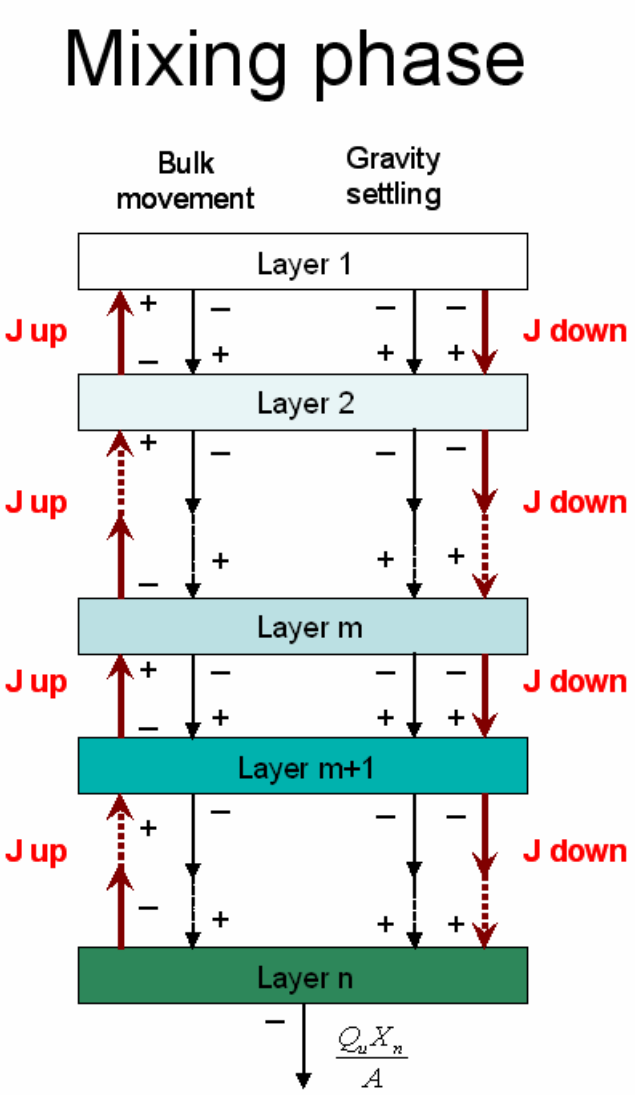


figure 27: Mixing Phase

The main characteristic of the *mixing phase* is that the tank is well mixed, as the name of the phase stands up. There is an external flux that makes practically despicable the settling flux and the flux due to the waste sludge flowrate.

When the tank is in this phase is equivalent to an anoxic tank where the biological process called nitrification is taking part by the autotrophs microorganisms (see *chapter 2: Wastewater Treatment Processes*, point *"2.1 Process description"*).

The next phase after the *mixing phase* is the *reaction phase*, explained in the following page.

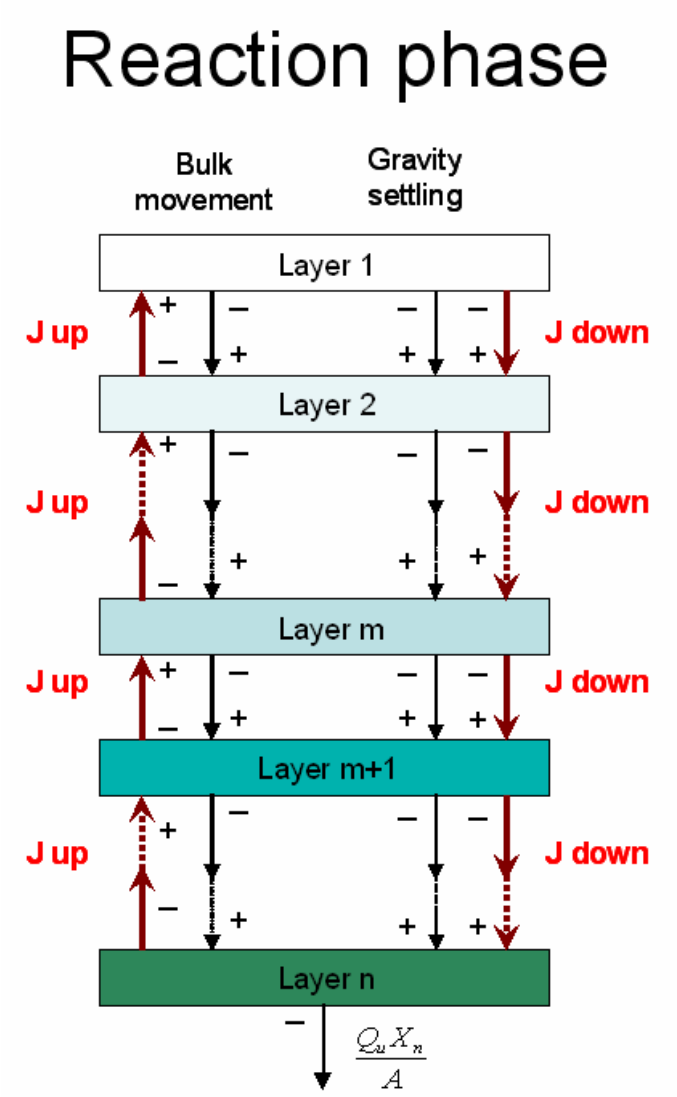


figure 28: Reaction Phase

The main characteristics in the *reaction phase* are:

- *Tank well mixed*. The value of the flux generated by the input flowrate (J_{up} = -J_{uown}) is bigger than the settling flux due to the gravity force and the bulk flux due to the waste sludge flowrate, both of them are despicable in front of the value of J_{up}.
- $J_{up} = -J_{down} > 0$.
- $q_{out} = 0$.
- $q_{in} = 0$.
- $q_u > 0$.
- $K_L a > 0$, there is reaction.
- $\frac{dh}{dt} = 0$, the derivative of the height is null because there is no input and output flowrate, and it is right because the height has a constant value.

The value of the $K_L a$ is the only difference between the *mixing phase* and the *reaction phase*.

The *reaction phase* is equivalent to the aerobic tank in modelling continuous wastewater treatments (see *chapter 2: Wastewater Treatment Processes*, point *"2.1 Process description"*).

The main characteristics of this phase are that the tank is well mixed and that there is an increment of the concentration of dissolved oxygen because of the tank is now aerated. In this phase is taking part the denitrification by the heterotrophs microorganisms principally.

Once the *reaction phase* has taken part, the process goes in the *settling phase*. This is the first phase where the vessel is not well mixed. In this phase takes part the settling of solid particles in the down layer of the settler, because now the tank is working as a settler. Although there are taking part biological processes, the principal process is settling and clarification.

The main characteristics of this phase are:

- *The tank is not well mixed*. The value of the flux generated by the input flowrate (J_{up} = J_{uown}= 0) is zero and there are only the settling flux and the flux due to the waste sludge flowrate.
- $J_{up} = J_{down} = 0$.
- $q_{out} = 0$.
- $q_{in} = 0$.
- $q_u > 0$.
- $K_L a = 0$, there is no reaction.
- $\frac{dh}{dt} = 0$, the derivative of the height is null because there is no input and output flowrate, and it is right because the height has a constant value.

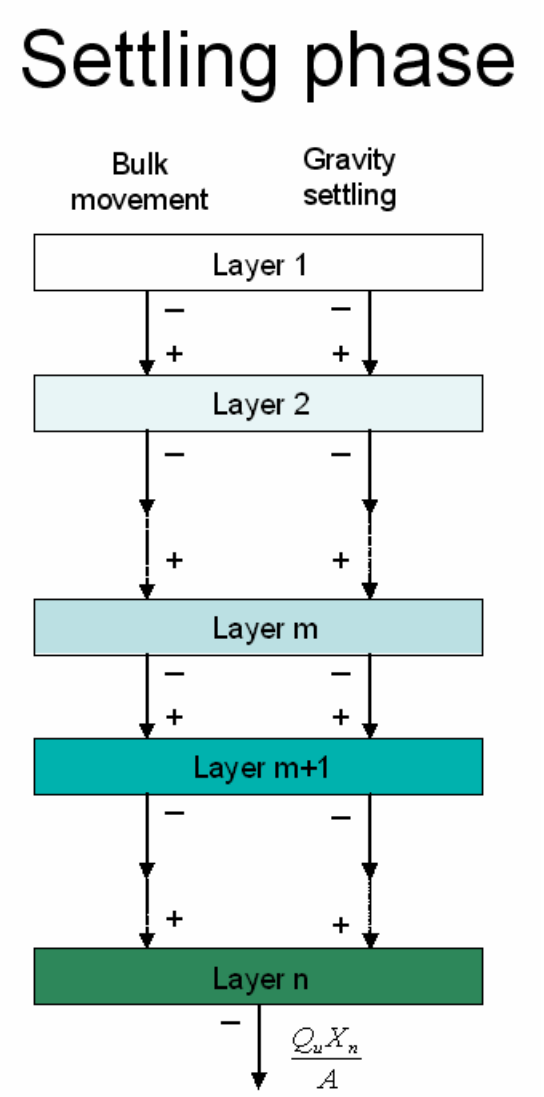


figure 29: Settling Phase

As it has been said, the main characteristic of this phase is the settling and clarification. Now the flux present in the three first phases is not present, so the settling flux and the flux due to the waste settling flowrate are the responsible of the settling of the solid particles in the down layer on the button of the vessel.

This phase is the last treatment of the wastewater in the "secondary treatment" (see *chapter 2: Wastewater Treatment Processes*, point *"2.1 Process description"*). The wastewater will go out from the vessel to trespass the tertiary treatment.

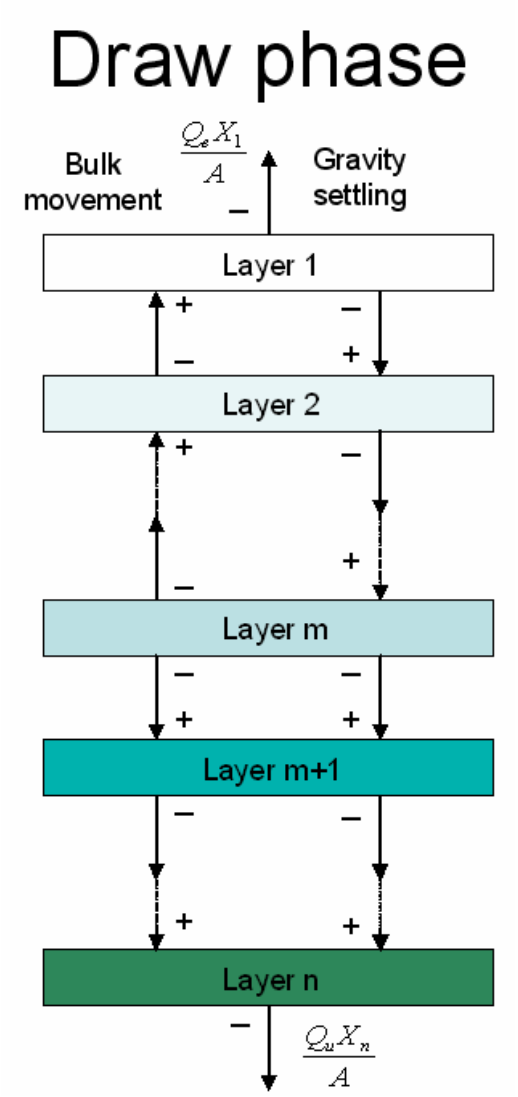


figure 30: Draw Phase

This phase is certainly the real last phase of the process. The clarified wastewater goes out of the vessel, the level of the wastewater in the vessel decreases and almost all the solid bulks rest on the button layer of the vessel. After the past phases there are no solid particles in the up layers of the vessel.

The main characteristics of the *draw phase* are:

- *The tank is not well mixed*. The value of the flux generated by the input flowrate (J_{up} = J_{uown}= 0) is zero and there are only the settling flux and the flux due to the waste sludge flowrate.
- $J_{up} = J_{down} = 0$.
- $q_{out} > 0$, is the main characteristic, now there is output flow rate.
- $q_{in} = 0$.
- $q_u > 0$.
- $K_L a = 0$, there is no reaction.
- $\frac{dh}{dt} = \frac{q_{in} - q_{out}}{A}$, where A is the area of the tank. The derivative of the height is negative which means that the height of the level is decreasing. This is the main characteristic of the *draw phase*.

This phase is practically like the *settling phase*, the only particularity is the variation of the height in the level of the wastewater.

The *draw phase* could be the last one, but there is one problem that has been exposed before. When the *draw phase* has reached the end (the height has the minimum value it can have) the process has to start again from the first phase, the *filling phase*. It means that it is necessary input flowrate. The input flowrate in one wastewater treatment plant is assumed is constant, so if there is another tank in the *filling phase*, another one can not goes from the *draw phase* to the *filling phase* because there is no input flowrate for both tanks. But the process can not be in the *draw phase* until there is input flowrate, because there is a minimum value for the height *'h'* of the level in the vessel. So, it is need a last phase to avoid this problematic situation. This phase is called ***waiting phase***.

The *waiting phase* is the last phase of the process modelled. This phase is a transition phase from the *draw phase* to the *filling phase*. The main characteristics of the phase are the same as the ones in the settling phase, but here the height of the wastewater level has the minimum value. The characteristics are:

- *The tank is not well mixed*. The value of the flux generated by the input flowrate (J_{up} = J_{uown}= 0) is zero and there are only the settling flux and the flux due to the waste sludge flowrate.
- $J_{up} = J_{down} = 0$.
- $q_{out} = 0$.
- $q_{in} = 0$.
- $q_u > 0$.
- $K_L a = 0$, there is no reaction.
- $\frac{dh}{dt} = 0$, the derivative of the height is null because there is no input and output flowrate, and it is right because the height has a constant value.

The inputs have exactly the same value as in the *settling phase*.

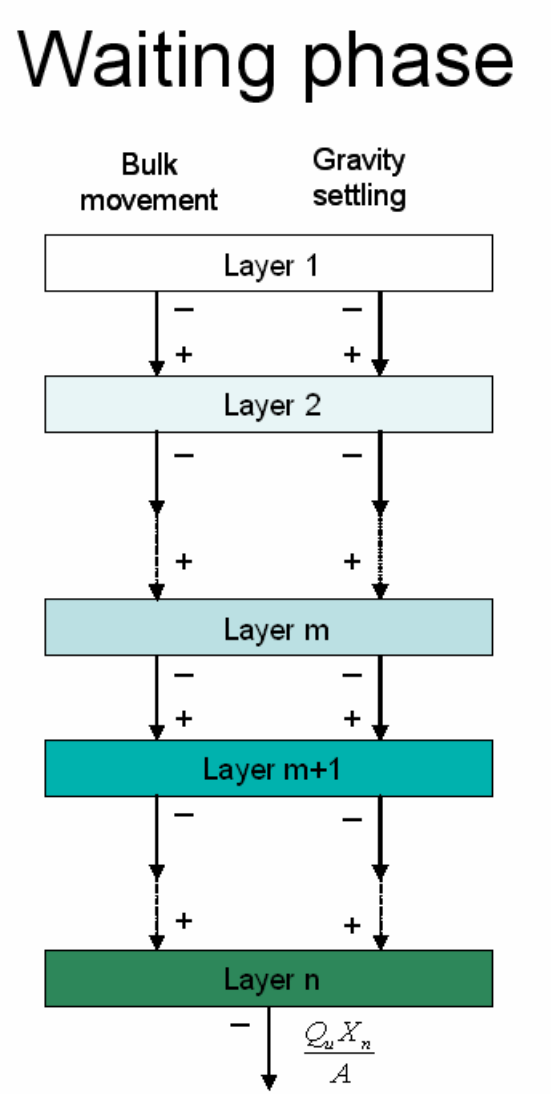


figure 31: Waiting Phase

After analysing all the phases, there are some things that must be stood up. The waste sludge flowrate (q_u) is always active in all phases and thereby the flux generated by it. The gravity force is always present too; it is which generate the settling flux. By the other side, there are three phases where the vessel is well mixed, the three first phases: *filling phase*, *mixing phase* and *reaction phase*. And in the last three phases the vessel is not well mixed, there is more concentration of solid particles in the button of the vessel.

Rather there is one data is not clear when the phases are explained. How takes part the transition between two phases. It is clear that the transition between the *filling phase* and the *mixing phase* is the value of the height, and between the *draw phase* and the *waiting phase* is the height too. And between the waiting phase and the filling phase is clear too, when the controller can give to the vessel input flowrate. But the other three transitions are not clear. To do these transitions they can be suitable more than one condition. For example it is possible to measure in the real plants where it is apply the batch process the time of each phase. Another condition is to choose a limit value of the critical concentration of solid particles in each phase (ammonium in the *mixing phase*,

nitrate in the *reaction phase*, "total suspended solids" in the *settling phase*, for example).

5.2.2 Model equations

The characteristics and differences of the phases have been explained in the point before, but, how to express it as logic equations, this is the step that has to be resolved once it is clear the model is going to be applied.

The starting point is the ***traditional one-dimensional layer settler model,*** with the ***ASM1 model*** included in each layer to model the biological reactions and the settling and clarification process in the same mathematical model. This model was implemented by Ulf Jeppsson (Industrial Electrical Engineering and Automation –IEA-, LTH, Sweden) and Krist V. Gernaey (CAPEC, Dept. of Chemical Engineering, DTU, Denmark) previously, so here is not going to be designed the model from the beginning. The existing model has been used to get the batch process equations. There have been introduced some modifications in the equations of the current model.

In the lines bellow are shown the modified equations in the model, but is not presented the whole model that was developed before by Ulf Jeppsson and Krist V. Gernaey. Otherwise (the equations are the ones explained in *chapter 2*, point *2.3 Modelling of wastewater treatment processes*), the reader can see the whole model if he checks the code implementation in the point after (*5.3 Implementation of the model*). The equations modified are:

STATE VARIABLE height:

$$\frac{dh}{dt} = \frac{q_{in} - q_{out}}{area}$$

VELOCITY OF THE BULK MOVEMENT:

$$v_{up} = \begin{cases} -v_{down} & if \quad q_{in} = 0 \quad and \quad q_{out} = 0 \\ \dfrac{q_{in} - q_u}{area} & if \quad q_{in} > 0 \quad and \quad q_{out} = 0 \\ \dfrac{q_{out} - q_u}{area} & if \quad q_{out} > 0 \quad and \quad q_{in} = 0 \end{cases}, \text{ where } v_{down} = \frac{q_u}{area}$$

DISSOLVED OXYGEN CONCENTRATION:

Equation for the layers up to the feed layer:

$$\frac{dS_{O_i}}{dt} = \frac{-v_{up}a + v_{up}b}{h} + \left[\begin{array}{l} \dfrac{1 - Y_H}{Y_h} \left(\hat{\mu}_H \dfrac{S_{S_i}}{K_S + S_{S_i}} \dfrac{S_{O_i}}{K_{O,H} + S_{O_i}} X_{B,H_i} \right) - \\ - \dfrac{4{,}57 - Y_A}{Y_A} \left(\hat{\mu}_A \dfrac{S_{NH_i}}{K_{NH} + S_{NH_i}} \dfrac{S_{O_i}}{K_{O,A} + S_{O_i}} \right) X_{B,A_i} \end{array} \right] + K_L a \left(S_{Osat} - S_{O_i} \right)$$

Equation for the layers down to the feed layer:

$$\frac{dS_{O_i}}{dt} = \frac{v_{down} S_{O_{i-1}} - v_{down} S_{O_i}}{h} + \left[\begin{array}{l} \frac{1-Y_H}{Y_h} \left(\hat{\mu}_H \frac{S_{S_i}}{K_S + S_{S_i}} \frac{S_{O_i}}{K_{O,H} + S_{O_i}} X_{B,H_i} \right) - \\ - \frac{4{,}57 - Y_A}{Y_A} \left(\hat{\mu}_A \frac{S_{NH_i}}{K_{NH} + S_{NH_i}} \frac{S_{O_i}}{K_{O,A} + S_{O_i}} \right) X_{B,A_i} \end{array} \right] + K_L a \left(S_{Osat} - S_{O_i} \right)$$

Equation for the feed layer:

$$\frac{dS_{O_i}}{dt} = \frac{v_{in} S_{Oin} - v_{up} a - v_{down} S_{O_i}}{h} + \left[\begin{array}{l} \frac{1-Y_H}{Y_h} \left(\hat{\mu}_H \frac{S_{S_i}}{K_S + S_{S_i}} \frac{S_{O_i}}{K_{O,H} + S_{O_i}} X_{B,H_i} \right) - \\ - \frac{4{,}57 - Y_A}{Y_A} \left(\hat{\mu}_A \frac{S_{NH_i}}{K_{NH} + S_{NH_i}} \frac{S_{O_i}}{K_{O,A} + S_{O_i}} \right) X_{B,A_i} \end{array} \right] + K_L a \left(S_{Osat} - S_{O_i} \right)$$

$$where \quad a = \begin{cases} S_{O_i} & if \quad others \\ S_{O_{i-1}} & if \quad v_{up} = -v_{down} \end{cases}$$

$$and \quad b = \begin{cases} S_{O_{i+1}} & if \quad others \\ S_{O_i} & if \quad v_{up} = -v_{down} \end{cases}$$

, and '*i*' indicates the layer.

SUSPENDED SOLID CONCENTRATION:

This equation is the same for all suspended solids.

Equation for the layers up to the feed layer:

$$\frac{dX_i}{dt} = \frac{X_i}{TSS_i} \left(- J_{flow_i} - J_{settler_i} - J_{down\_up} \right) + \frac{X_{i-1}}{TSS_i} J_{settler_i} + \frac{X_{i+1}}{TSS_{i+1}} \frac{J_{flow_i}}{h} + \frac{X_{i+1}}{TSS_{i+1}} J_{down\_up}$$

Equation for the layers down to the feed layer:

$$\frac{dX_i}{dt} = \frac{X_i}{TSS_i} \left(- J_{flow_i} - J_{settler_i} - J_{down\_up} \right) + \frac{X_{i+1}}{TSS_{i+1}} \frac{J_{flow_i} + J_{settler_i}}{h} + \frac{X_{i-1}}{TSS_{i-1}} J_{down\_up}$$

Equation for the feed layer:

$$\frac{dX_i}{dt} = \frac{X_i}{TSS_i} \left(- J_{flow_i} - J_{flow_{i+1}} - J_{settler_i} - J_{down\_up} \right) + \frac{X_{i-1}}{TSS_{i-1}} J_{settler_i} + v_{in} \frac{X_{Iin}}{h} + \frac{X_{i-1}}{TSS_{i-1}} J_{down\_up}$$

, where *X* represents the solid particles concentration of X_I, X_S, $X_{B,H}$, $X_{B,A}$, X_P or X_{ND}, and '*i*' represents the layer. The fluxes are:

- J_{down_up} = constant
- $J_{settler} = v_{sedimentation_i} TSS_i$
- $J_{flow_i} = \begin{cases} v_{up} TSS_i & if \quad i < m \quad and \quad (q_{in} > 0 \quad or \quad q_{out} > 0) \\ v_{down} TSS_{i-1} & if \quad others \end{cases}$, where '*m*' is the feed layer.

To know more about the meaning of each parameter in the equations check it in *chapter 2*, point *2.3 Modelling of wastewater treatment processes,* or see the bibliography (recommended Ulf Jeppsson's book [2] in the bibliography, in pages 372 and 373 is written the ASM1 model with all the parameters). About the fluxes and the settling equations check Ulf Jeppsson's book [2] too.

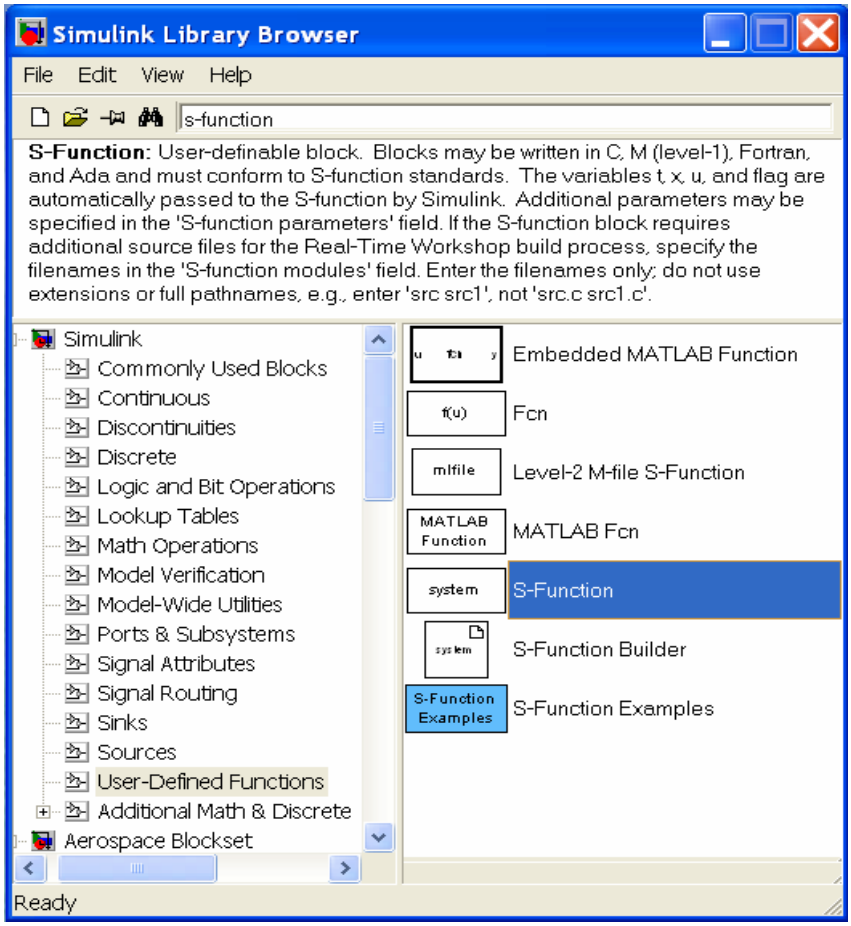
5.3 Implementation of the model

In *chapter 3* was described the application designed and the background of the simulation. For the simulation of the model is run Simulink, the simulation tool of Matlab.

To program the equations of the model is used the object S-function. Is a Simulink object which let to program with C code or with Matlab code (m-function) a model and include it as a block in the Simulink workspace, so it is possible to use the tools of Simulink to define the value of the inputs and the representation of the outputs, the feedback of the model process.

The S-function chosen is the one which let to program the model as a C file and to include it in Simulink workspace.

ABOUT THE "S-function":



We can find the S-function in the "Simulink Library Brower", as the *figure 32* shows.

We select the object S-function and drag it to the workspace of the Simulink file where is going to be implemented the model.

figure 32: S-function in Simulink Library Browser.

Once the block of the S-function is in the Simulink workspace, the only thing it is necessary to do is to press click twice selecting the block S-Function, and the window to define the *S-function Block Parameters* will open. See *figure 33* bellow.

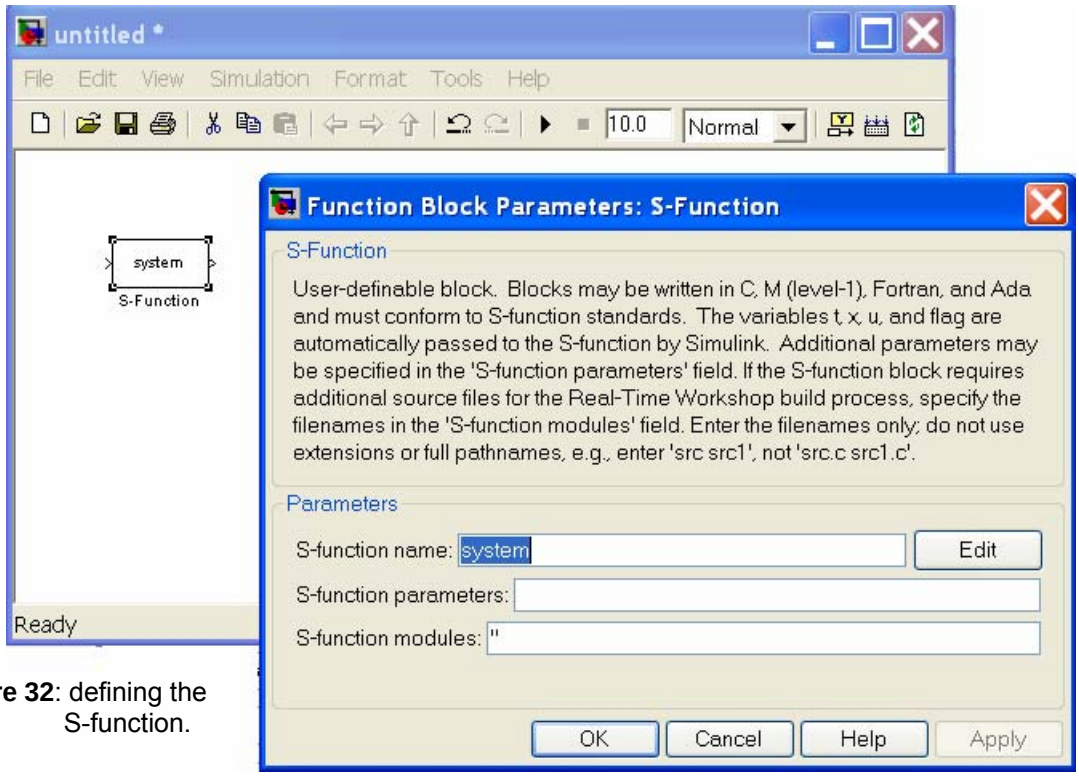


figure 32: defining the S-function.

There are three fields to fill up: "*S-function name*", "*S-function parameters*" and "*S-function modules*". The only two necessary fields are the "*S-function name*" and the "*S-function parameters*". In the first one it has to be written the name of the c-file where it is implemented the code of the S-function. The program locks for a file with the extension .C, so it is not necessary to indicate the extension of the file. In the second field they are written the input parameters used to define the value of the coefficients in the equations. These parameters have to be define within the c-code before the functions definition, in the field of the include libraries.

In the following lines is presented the functions that Matlab define to implement the code of the S-function. It is recommended to use only these defined functions, because Simulink can have problems with the simulation if we try to define another functions different to the usual functions that the application can recognise.

The first part is the include libraries, where it is indicated the name of the c-file, the include libraries and the constants define in Matlab workspace that the S-function is going to use to define variables:

```
#define S_FUNCTION_NAME name_of_the_c-file

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

#define NAME_OF_THE_PARAMETER   ssGetArg(S,0)
```

The second part is the function where it is defined the size of the input vector, output vector, the number of state variables continuous and discontinuous and another important parameters. If we don't write anything the value is zero:

```
/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(S, 0);    /*13 number of continuous states         */
    ssSetNumDiscStates(S, 0);    /* number of discrete states             */
    ssSetNumInputs(S, 0);        /* number of inputs                      */
    ssSetNumOutputs(S, 0);       /* number of outputs                     */
    ssSetDirectFeedThrough(S, 0);/* direct feedthrough flag               */
    ssSetNumSampleTimes(S, 0);   /* number of sample times                */
    ssSetNumSFcnParams(S, 0);    /* number of input arguments             */
    ssSetNumRWork(S, 0);         /* number of real work vector elements   */
    ssSetNumIWork(S, 0);         /* number of integer work vector elements*/
    ssSetNumPWork(S, 0);         /* number of pointer work vector elements*/
}
```

The following function is used to define the sample time needed to simulate the model defined. Here is defined if the function is continuous or the period of simulation if the model is a discontinuous model.

```
/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
```

In this function are defined the value of the initial conditions of the state variables of the model:

```
/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}
```

In the ***mdlOutputs*** function are defined the value of the outputs of the model:

```
/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S, int tid)
{
}
```

This is the last function, where the model and the derivative functions are implemented. Here are defined all the auxiliary variables and the equations of the model:

```
/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct *S, int
                           tid)
{
}
```

This part of the chapter is an introduction of the S-functions, they are not explained all the functions that can be implemented in the S-function and all their features. If the reader is interested in going further with the S-function definitions and uses he can check the bibliography.

5.3.1 S-function programmed

To implement the S-function it has been used the code programmed by Ulf Jeppsson and Krist V. Gernaey, following the same structure and utilising the equations implemented by them. Hence, in this point it is not going to be explained all the structure and parameters of the S-function programmed, it is only stood up the modifications made to the original file. Thereby it is possible to see the all code of the function checking the "*APPENDIX IV*".

In the following lines is explained the code implemented to write the modified equations described in "*5.2.2 Model equations*" in the current chapter.

The first thing is to define the parameters that are needed by the equations. These parameters are declared as a constants join to the declaration of the libraries used by the program. How to declare the parameters is shown bellow:

```
#define XINIT   ssGetArg(S,0)      /* initial values                    */
#define SEDPAR   ssGetArg(S,1)     /* parameters sedimentation model    */
#define DIM     ssGetArg(S,2)      /* dimensions clarifier              */
#define LAYER   ssGetArg(S,3)      /* Number of layers                  */
#define ASMPAR   ssGetArg(S,4)     /* parameters activated sludge model */
#define SOSAT   ssGetArg(S,5)
```

After declaring all the libraries and parameters, the next step is to implement the functions. The first function in the S-function is the ***mdlInitializeSizes*** function. This function is only to indicate to the compiler the number of inputs, outputs, state variables, parameters, etc. The ***mdlInitializeSizes*** function implemented is:

```
static void mdlInitializeSizes(SimStruct *S)
{
  ssSetNumContStates(  S, 141); /*   number   of   continuous   states,   ASM1
              components + TSS for each settler layer + total height for the
              vessel*/
  ssSetNumDiscStates(    S, 0); /* number of discrete states         */
  ssSetNumInputs(       S, 19);  /* number of inputs, 13 ASM1 components + TSS
                                 + influent flow rate + Kla + J_down_up +
                                 Q_output+ Q_u waste sludge flowrate */
  ssSetNumOutputs(     S, 144);  /* number of outputs: 141 states 13 ASM1
                                 components + height + TSS for each layer)+
                                 Q_e + Q_out + Q_u*/
```

```
  ssSetDirectFeedThrough(S, 1);  /* direct feedthrough flag           */
  ssSetNumSampleTimes(   S, 1);  /* number of sample times            */
  ssSetNumSFcnParams(    S, 6);  /* number of input arguments         */
  ssSetNumRWork(         S, 0);  /* number of real work vector elements */
  ssSetNumIWork(         S, 0);  /* number of integer work vector elements*/
  ssSetNumPWork(         S, 0);  /* number of pointer work vector elements*/
}
```

The functions between ***mdlInitializeSizes*** function and ***mdlOutputs*** function are not modified, so the next function implemented is ***mdlOutputs*** function. Within this function is defined the value of all the outputs. The 141 state variables are defined as an outputs to can measure and represent the evolution of the state variables of the model. The other outputs are the effluent flowrate (Q_e), the waste sludge flowrate (Q_u) and the output flowrate (Q_o).

```
static  void  mdlOutputs(double  *y,  double  *x,  double  *u,
                         SimStruct *S, int tid)
{
int i;
int j;
    /* Clarifier state variables */
    for (i=0;i<10;i++){
        for (j=0;j<14;j++){
        y[(i*14)+j]=x[i+j*10];    //we have written 14 because now we have 15
                                     states variables per each layer
        }                          //the order of outputs is from 1 to 14 layer
                                     1, from 15 to 28 layer 2, etc
    }                             //the inputs is, from 1 to 10 the SI of all
                                     layers, from 11 to 20 the Ss of all
                                     layers, etc
   y[140] = x[140];

   /* Flow rates out of the vessel */
   if ((u[14]>0)&&(u[17]==0)){
       y[141]=u[14]-u[18];       /* Q_e  */
   }
   if ((u[14]==0)&&(u[17]>0)){
       y[141]=u[17]-u[18];       /* Q_e  */
   }
   else y[141]=0;

   y[142]=u[18];                 /* Q_u   */
   y[143]=u[17];                 /* Q_out */
 }
```

The more complex function that has to be defined is the ***mdlDerivatives*** function. Within it is where is defined the model of the wastewater treatment plant. The implementation made is longer than the lines shown bellow, but here are shown the main changes made in the original S-function to program the batch process model (check the *appendix IV* to see the full code)

```
static  void  mdlDerivatives(double  *dx,  double  *x,  double  *u,
                             SimStruct *S, int tid)
```

The implementation of the equation of the velocity up v_{up}, which have the solid particles presents up to the feed layer, shown in the point *5.2.2 Model equations*, in the current chapter, is shown bellow:

```
/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    NEW    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */
// Equaction of effluent
   if ((u[14]>0)&&(u[17]==0)){
       Q_e = u[14]-Q_u;       /* Q_e  */
   }
   if ((u[14]==0)&&(u[17]>0)){
       Q_e = u[17]-Q_u;       /* Q_e  */
   }
   else Q_e = 0;

// Equation of the velocity up
   if ((u[14]>0)||(u[17]>0)){
       v_up = Q_e/area;
   }
   else v_up = -v_dn;
```

$$v_{up} = \begin{cases} -v_{down} & if \quad q_{in} = 0 \quad and \quad q_{out} = 0 \\ \dfrac{q_{in} - q_u}{area} & if \quad q_{in} > 0 \quad and \quad q_{out} = 0 \\ \dfrac{q_{out} - q_u}{area} & if \quad q_{out} > 0 \quad and \quad q_{in} = 0 \end{cases}$$

```
// The flux for filling, mixing and reaction phase,
// where the wastewater is well mixed
J_d_u = u[16];
/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */
```

The line "***J_d_u = u[16];***" is the declaration of the flux generated in *filling phase*, *mixing phase* and *reaction phase* to get that the vessel was well mixed.

The following representative change is the code programmed to implement the sludge flux due to the liquid flow:

```
/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    NEW    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */
/* calculation of the sludge flux due to the liquid flow (upflow or downflow,
depending on layer) */
for (i = 0; i < 11; i++) {
   //bulk movement up to the feed layer
   if (i < (feedlayer-eps)){
       if ((u[14]>0)||(u[17]>0)){
           Jflow[i] = v_up*xtemp[i+130];
       }
       else{
           Jflow[i] = v_dn*xtemp[i-1+130];
       }
   }
   //bulk movement down to the feed layer and in the feed layer
   else
      Jflow[i] = v_dn*xtemp[i-1+130];
}
/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */
```

$$J_{flow_i} = \begin{cases} v_{up} TSS_i & if \quad i < m \quad and \quad (q_{in} > 0 \quad or \quad q_{out} > 0) \\ v_{down} TSS_{i-1} & if \quad others \end{cases}$$

The first state variable defined is the state variable *height*, the main characteristic of this model. Now the height of the vessel is not a constant as in the models before. As it is assumed all the layers have the same height, first is define the state variable of the total height of the vessel, and after it is divided by the number of layers in the model:

```
/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    NEW    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */
/* height of the tank: H*/

    dx[140] = (u[14]-u[17])/area; /*the heigth of the vessel*/
    h = x[140]/n;                 /*the heigth of each layer*/

/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */
```

The other significant change in the state variables has been to include the flux needed to get the vessel was well mixed (***J_d_u***) within the equations of the particulate components: X_I, X_S, $X_{B,H}$, $X_{B,A}$, X_P and X_{ND}. The changes made are stood up in black letters:

```
/* particulate component X_I */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+20] = ((xtemp[i+20]/xtemp[i+130])*(-Jflow[i]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+20]/xtemp[i-
                 1+130])*Js[i]+(xtemp[i+1+20]/xtemp[i+1+130])*Jflow[i+1])/h +
                 (xtemp[i+1+20]/xtemp[i+1+130])*J_d_u + reac3[i];
   else if (i > (feedlayer-eps))
      dx[i+20] = ((xtemp[i+20]/xtemp[i+130])*(-Jflow[i+1]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+20]/xtemp[i-1+130])*(Jflow[i]+Js[i]))/h +
                 (xtemp[i-1+20]/xtemp[i-1+130])*J_d_u + reac3[i];
   else
      dx[i+20] = ((xtemp[i+20]/xtemp[i+130])*(-Jflow[i]-Jflow[i+1]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+20]/xtemp[i-1+130])*Js[i]+v_in*u[2])/h +
                 (xtemp[i-1+20]/xtemp[i-1+130])*J_d_u + reac3[i];
}
```

The last equation modified is the equation of the state variable dissolved oxygen "S_O". Has been included the reaction rate (*KLa*(SO_sat-xtemp[i+70])*) of the oxygen to model the *reaction phase*. The second change is to add the variables *a* and *b* within the equations. It is necessary because of the different values of the velocity up v_{up}, which depends of $q_{effluent}$ (Q_e).

```
/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    NEW    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */
/* soluble component S_O */

KLa = u[15];

for (i = 0; i < 10; i++) {
   if(v_up==-v_dn){
       a = xtemp[i-1+70];
       b = xtemp[i+70];
   }
   else{
       a = xtemp[i+70];
       b = xtemp[i+1+70];
   }

   if (i < (feedlayer-1-eps))
      dx[i+70]  = (-v_up*a +v_up*b)/h +reac8[i] + KLa*(SO_sat-xtemp[i+70]);
                            //we have added rSO (Kla!=0)
```

$$a = \begin{cases} S_{O_i} & if \quad others \\ S_{O_{i-1}} & if \quad v_{up} = -v_{down} \end{cases}$$

$$b = \begin{cases} S_{O_{i+1}} & if \quad others \\ S_{O_i} & if \quad v_{up} = -v_{down} \end{cases}$$

```
    else if (i > (feedlayer-eps))
       dx[i+70]  = (v_dn*xtemp[i-1+70] -v_dn*xtemp[i+70])/h +reac8[i] +
                    KLa*(SO_sat-xtemp[i+70]); //we have added rSO (Kla!=0)
    else
       dx[i+70]  = ((v_in*u[7] -v_up*a -v_dn*xtemp[i+70])/h +reac8[i]) +
                    KLa*(SO_sat-xtemp[i+70]); //we have added rSO (Kla!=0)
}
/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */
```

5.4 Comments

In the points before a batch process model has been implemented. The S-function has been programmed to model the six phases of the batch process model. The inputs that control if the model is in one or another phase are:

- Q_i: input flowrate
- Q_o: output flowrate
- J_{down_up}: extra flux in filling phase and the biological phases
- Q_u: waste sludge flowrate
- $K_l a$: constant of the dissolved oxygen concentration's reaction

Now, only rests to implement a control as it shows the *figure 33*:

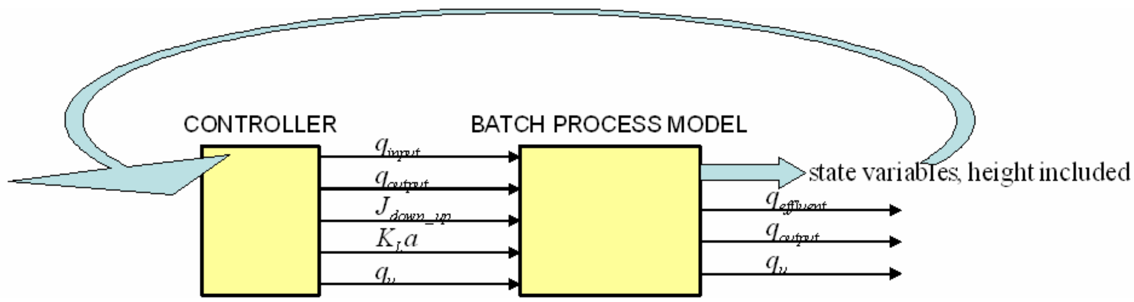


figure 33: batch process control

Matlab offers a lot of possibilities to implement the control in Simulink or even in Matlab workspace as an "m function".

The aim of this chapter was to model batch process model, the controller is not part of it, so the chapter is not going further in this topic.

PART III

TITLE: ***Appendixes***
(Included Papers)

APPENDIX I

Appendixes to CHAPTER 2 (Wastewater Treatment Processes)

In this chapter there is more information about Wastewater Treatment Processes, in order to extend the one given in “Chapter 2” before, for these ones who want to know more about it.
But if it is needed more knowledge and to go further from this introduction to wastewater treatment process, there is the possibility of checking the bibliography.

I.1 Appendixes for MODELLING WASTEWATER TREATMENT PROCESSES

The first thing it is pertinent to know is how to read the tabular form in which activates sludge stoichiometry and kinetics are conventionally presented.
The kind of the tabular form is presented below:

PROCESS	COMPONENTS				KINETICS
	State var. 1	State var. 2	…	State var. n	
Process 1	Value 1,1	Value 1,2	…	Value 1,n	Kinetic 1
Process 2	Value 2,1	Value 2,2	…	Value 2,n	Kinetic 2
…	…	…	…	…	…
Process k	Value k,1	Value k,2	…	Value k,n	Kinetic k

, where, for instance, the ***reaction rate*** of the "state variable 2" is:

$$reactionrate2 = value(1,2) * kinetic(1) + value(2,2) * kinetic(2) + ... + value(k,2) * kinetic(k)$$

I.1.1 Tabular form for "simple biological kinetics"

PROCESS	COMPONENTS		KINETICS
	Nutrient N	Biomass B	
Aerobic heterotrophic growth	$-\frac{1}{Y_B}$	1	$\hat{\mu}\left(\frac{S_N}{K_N + S_N}\right)X_B$

I.1.2 Tabular form for "simple biological kinetics" and "carbon removal kinetics"

PROCESS	COMPONENTS			KINETICS
	Nutrient	Oxygen	Biomass	
Aerobic heterotrophic growth	$-\frac{1}{Y_H}$	$\frac{Y_H - 1}{Y_H}$	1	$\hat{\mu}_H\left(\frac{S_S}{K_S + S_S}\right)\left(\frac{S_O}{K_{OH} + S_O}\right)X_H$
Decay of heterotrophs	$1 - f_P$		-1	$b_H X_H$

I.1.3 Tabular form for "simple biological kinetics", "carbon removal kinetics" and "nitrogen removal kinetics"

PROCESS	COMPONENTS					
	Carbon	Oxygen	Ammonium	Nitrate	Heterotrophs biomass	Autotrophs biomass
Aerobic heterotrophic growth	$-\frac{1}{Y_H}$	$\frac{Y_H-1}{Y_H}$	$-i_{XB}$		1	
Anoxic heterotrophic growth	$-\frac{1}{Y_H}$		$-i_{XB}$	$\frac{Y_H-1}{2{,}86Y_H}$	1	
Aerobic autotrophic growth		$\frac{Y_A-4{,}57}{Y_A}$	$-i_{XB}-\frac{1}{Y_A}$	$\frac{1}{Y_A}$		1
Decay of heterotrophs	$1-f_P$		$i_{XB}-f_P i_{XP}$		-1	
Decay of autotrophs	$1-f_P$		$i_{XB}-f_P i_{XP}$			-1

PROCESS	KINETICS
Aerobic heterotrophic growth	$\hat{\mu}_H\left(\frac{S_S}{K_S+S_S}\right)\left(\frac{S_O}{K_{OH}+S_O}\right)X_H$
Anoxic heterotrophic growth	$\hat{\mu}_H\left(\frac{S_S}{K_S+S_S}\right)\left(\frac{K_{OH}}{K_{OH}+S_O}\right)\left(\frac{S_{NO}}{K_{NO}+S_{NO}}\right)\eta_g X_H$
Aerobic autotrophic growth	$\hat{\mu}_A\left(\frac{S_{NH}}{K_{NH}+S_{NH}}\right)\left(\frac{S_O}{K_{OA}+S_O}\right)X_A$
Decay of heterotrophs	$b_H X_H$
Decay of autotrophs	$b_A X_A$

I.1.4 Tabular form for "phosphorus removal kinetics"

PROCESS	COMPONENTS						
	S_F	S_A	S_{O2}	S_{PO4}	X_{PHA}	X_{PP}	X_{PAO}
Fermentation	-1	1					
P Release		-1		Y_{PO4}	1	$-Y_{PO4}$	
P Uptake			$-Y_{PHA}$	-1	$-Y_{PHA}$	1	
PAO Growth			$1-\frac{1}{Y_H}$	$-i_{PBM}$	$-\frac{1}{Y_H}$		1
PHA Breakdown		1			-1		
PP Breakdown				1		-1	
PAO Breakdown				ν_P			-1

PROCESS	KINETICS
Fermentation	$q_{FE}\left(\frac{K_{O2}}{K_{O2}+S_{O2}}\right)\left(\frac{K_{NO3}}{K_{NO3}+S_{NO3}}\right)\left(\frac{S_F}{K_{FE}+S_F}\right)X_H$
P Release	$q_{PHA}\left(\frac{S_A}{K_A+S_{A2}}\right)\left(\frac{X_{PP}/X_{PAO}}{K_{PP}+X_{PP}/X_{PAO}}\right)X_{PAO}$
P Uptake	$q_{PP}\left(\frac{S_{O2}}{K_{O2}+S_{O2}}\right)\left(\frac{S_{PO4}}{K_P+S_{PO4}}\right)\left(\frac{X_{PHA}/X_{PAO}}{K_{PHA}+X_{PHA}/X_{PAO}}\right)\left(\frac{K_{MAX}-X_{PP}/X_{PAO}}{K_{IPP}+K_{MAX}-(X_{PP}/X_{PAO})}\right)X_{PAO}$
PAO Growth	$\hat{\mu}_{PAO}\left(\frac{S_{O2}}{K_{O2}+S_{O2}}\right)\left(\frac{S_{PO4}}{K_P+S_{PO4}}\right)\left(\frac{S_{NH}}{K_{NH}+S_{NH}}\right)\left(\frac{X_{PHA}/X_{PAO}}{K_{PHA}+(X_{PHA}/X_{PAO})}\right)X_{PAO}$
PHA Breakdown	$b_{PHA}X_{PHA}$
PP Breakdown	$b_{PP}X_{PP}$
PAO Breakdown	$b_{PAO}X_{PAO}$

APPENDIX II

Appendixes to CHAPTER 3 (Application Design)

In this chapter there is more information about the application designed with Matlab; the code and the *'.exe'* application.
There is nothing about the synchronization, data exchange and model design, these parts are described in chapters 4 and 5 and the appendixes of those chapters.

This appendix tries to complete the information given in *"chapter 3"* about the implementation of the application. But first it is necessary to do a brief explanation about the communication between the application and the controller, part which is explained in detail in *"chapter 3"*.

The way chosen to communicate the controller and the application through Ethernet is writing and reading "XML" files, so to store the value of all the variables that the controller has to update (control variables, synchronization variables, data read from the database and the results from the simulation) it is needed an "XML" file. It means that all the functions implemented to change or to read the values of the control variables and of the synchronization variables, they only have to be able to write or to read an "XML" file. About how to get this with Matlab is explained in *"chapter 4"*.

Another reason for storing the variables that they are changing during the application running in "XML" files is that Matlab is not able to change the value of the variables in the work-space directly when it is executing a loop. It only can change the values of the variables of the work-space inside the loop, or with a function which reads the value of these variables from a file, where the values are stored. So, when the application is executing the "main loop", to make possible that it reads the changes that the controller or the other buttons out of the "main loop" make to the control and synchronization variables, they have to change the values writing the file where they are stored, so when the reading functions inside the "main loop" reads the file the changes made have effect.

II.1 STAR emulator to check the synchronization code

The user interface designed for the application has two buttons to check the synchronization phases. An image about the user interface is shown below:

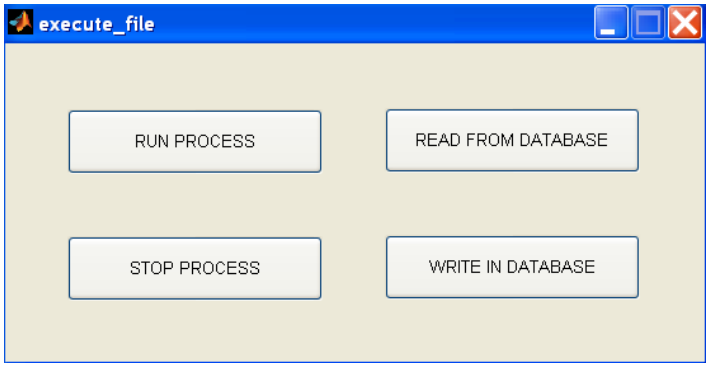


figure 34: User Interface

The buttons "READ FROM DATABASE" and "WRITE IN DATABASE" use the function implemented to emulate the controller only in the synchronization phases. These buttons execute the function "*writeXML_synchro (variables)*", where "variables" is a vector with two components. This function writes the value introduced (variables) in the file "XML" that stores the value of the synchronization variables.

II.1.1 "writeXML_synchro" code

The function "writeXML_synchro" uses the "**xmlwrite**" command to write an XML file where stores the value of the two synchronization variables. This function works like the function "writeXML" uses to write in the database the results of the simulation. The function makes the XML file in three steps:

- to create a "java object"
- to create an XML file
- to write the "java object" which contains the variables and the values in the XML file with the function "xmlwrite"

```
function archivoXML=writeXML_synchro(valueP)

    % Create a sample XML document.

    docNode = com.mathworks.xml.XMLUtils.createDocument('synchronize');
    docRootNode = docNode.getDocumentElement;
    %docRootNode.setAttribute('attribute','attribute_value');
       dataElement = docNode.createElement(sprintf('sincro'));
       nameElement = docNode.createElement(sprintf('read'));
       valueElement = docNode.createElement(sprintf('write'));

       nameElement.setTextContent(sprintf('%f',valueP(1)));   %name   of   the
                                                                 variable
       valueElement.setTextContent(sprintf('%f',valueP(2)));  %value  of  the
                                                                 variable
       dataElement.appendChild(nameElement);
       dataElement.appendChild(valueElement);
       docRootNode.appendChild(dataElement);

    % Save the sample XML document.
    xmlFileName  =  ['synchro','.xml'];  %we  have  the  file  in  our  current
                                            directory
    xmlwrite(xmlFileName,docNode);
```

How to create the "java object" to write an XML file or read an XML file is explained in "*chapter 4*", which is about the data-exchange and the synchronization files.

II.2 Control functions

In the implementation of the program there have been defined two control variables: "aux_loop" and "aux_sim". The value of these variables is stored in the file "*controlFILE.xml*" and the meaning of each variable is:

- ***aux_loop***: control variable for the "main loop", so if *"aux_loop≠1"* the condition to keep running the program is not complained and the program goes out of the "main loop".
- ***aux_sim***: control variable to stop the simulation of the plant process. The difference between "aux_sim" and "aux_loop" is that "aux_sim" doesn't stop the "main loop".

With these variables the controller can stop the simulation if it is not necessary in some period, but also it can stop the application if there is no needed more data. Nevertheless, don't forget once the application goes out of the main loop it is not possible to start it again from the controller.

II.2.1 "controlloop" code

As the function described in the point *7.1.1 "writeXML_synchro" code*, "controlloop" has the same structure, the only difference is that the two variables stored inside are the control variables and not the synchronization variables as before.
This function is used once in the initialization part (check *chapter 3* last point *3.4.2 programme code*) to write the initial values of the control variables. After that, only the controller will be the one which changes the values of the control variables. The code is presented below:

```
function archivoXML=controlloop(control)

    % Create a sample XML document.
    %program
    docNode = com.mathworks.xml.XMLUtils.createDocument('control');
    docRootNode = docNode.getDocumentElement;
    %docRootNode.setAttribute('attribute','attribute_value');
%    for i=1:n
      dataElement = docNode.createElement(sprintf('data'));
      loopElement = docNode.createElement(sprintf('loop'));
      simElement = docNode.createElement(sprintf('simulation'));

      loopElement.setTextContent(sprintf('%f',control(1))); %name of the
                                                             variable
      simElement.setTextContent(sprintf('%f',control(2))); %value of the
                                                             variable

      dataElement.appendChild(loopElement);
      dataElement.appendChild(simElement);
      docRootNode.appendChild(dataElement);

    % Save the sample XML document.
    xmlFileName = ['controlFILE','.xml']; %we have the file in our current
                                           directory
    xmlwrite(xmlFileName,docNode);
```

II.2.2 "control_check" code

As well as the function presented in the point *7.2.1 "controlloop" code* is a function which writes an XML file, "control_check" read the values of the control variables stored in the XML file "controlFILE". The Maltlab command used to read the XML file is "**xmlread**". This function is divided in three steps:

- First it reads the XML file with the function "*xmlread*", that stores the XML file in a java object.
- After it scans the java object to find the items with the name "data". The item data has two children; "loop" is the child which has the value of the control variable "aux_loop" and "simulation" is the child which has the value of the control variable "aux_sim".
- At last, the function stores the values in the variables that returns to the Matlab work-space.

```
function [loop sim]=control_check

loopdata=[];
simdata=[];

infoLabel = 'Plot Tools';
infoCbk = '';
itemFound = false;

xDoc = xmlread('controlFILE.xml');


    % Find a deep list of all listitem elements.
    allListItems = xDoc.getElementsByTagName('data');

    % Note that the item list index is zero-based.
    for i = 1:allListItems.getLength          %it's looking for all items with
                                               <data(i,:)>
        thisListItem = allListItems.item(i-1);
        childNode = thisListItem.getFirstChild;

        while ~isempty(childNode)
            %Filter out text, comments, and processing instructions.
            if childNode.getNodeType == childNode.ELEMENT_NODE
                % Assume that each element has a single
                % org.w3c.dom.Text child.
                childText   =   char(childNode.getFirstChild.getData);%get   the
                                                                       data text

                switch char(childNode.getTagName)
                    case 'loop';
                        %itemFound = strcmp(childText, infoLabel);
                        loopdata(i)=sscanf(childText,'%f');%to     get     the
                                                          readdata value
                    case 'simulation' ;
                        %infoCbk = childText;
                        simdata(i)=sscanf(childText,'%f');%to      get      the
                                                          writedata value
                end
            end  % End IF
            childNode = childNode.getNextSibling;
        end  % End WHILE

        if itemFound
            break;
        else
            infoCbk = '';
        end
    end  % End FOR

%read value
 loop=loopdata;
 sim=simdata;
```

This kind of function is explained with more details in *chapter 4*, chapter about the communication of the application and the controller.

II.3 Processing data within Matlab

It has been shown in *chapter 3* that the "Matlab algorithm" contains three parts: process the data read, simulate the plant model and process the results. To do the steps of "process the data read" and "process the results" two functions have been implemented. These two functions are examples, and they depend of the model used in the application. So these functions are presented as a guide to implement them in the future.

II.3.1 "read_data" function

This function is the one which processes the data read from the database, it means, it has to translate the three vectors stored into variables useful to the simulation:

$$\left.\begin{aligned} name &= [name_1 \ldots \quad name_i \ldots \quad name_n] \\ value &= [value_1 \ldots \quad value_i \ldots \quad value_n] \\ quality &= [quality_1 \ldots \quad quality_i \quad quality_n] \end{aligned}\right\} \text{var}\,iable_i = \begin{bmatrix} name_i \\ value_i \\ quality_i \end{bmatrix}$$

$$where \quad i = 0,\ldots n$$

, and 'n' is the number of variables acquired.

One simple way is to implement a loop with conditional statements that scans the vector of the names, then for the variable "*variableDO*" used in the model, when the loop finds the component *nameR(i) = 'DO'*, it stores the value of *valueR(i)* inside the variable "*variableDO*".

```
% file to write the data from the database to work space

endloop = length (valueR);

for i = 1:endloop
    if nameR(i,:) == 'DO'
        variableDO = valueR(i)
    end
    if nameR(i,:) == 'SS'
        variableSS = valueR(i)
    end
    if nameR(i,:) == 'Xs'
        variableXs = valueR(i)
    end
    if nameR(i,:) == 'Hh'
        variableHh = valueR(i)
    end
end
```

II.3.2 “write_data.m” function

To implement the function which builds the three vectors with the results of the simulation is even easier than the case before. The only task is to write the three vectors and all the components ordered. The code is shown below:

```
% file to write the data from the work space to database

name = ['SI';'SS';'XI']
value = [reac5(end,1) reac5(end,14) reac5(end,3)]
quality = [1 0 1]
```

II.4 GUIDE application

II.4.1 What’s GUIDE

To developed the graphical interface of the application has been used GUIDE. GUIDE is an application integrated in Matlab which makes possible in a few steps and with a little time developed graphical interfaces. By the other side, the main advantage of GUIDE is that it is a Matlab application, so all the programs developed with GUIDE can access directly to the Matlab work-space.

To start GUIDE it has to be typed the command “guide” in Matlab work-space and it appears the window shown in the following figure:

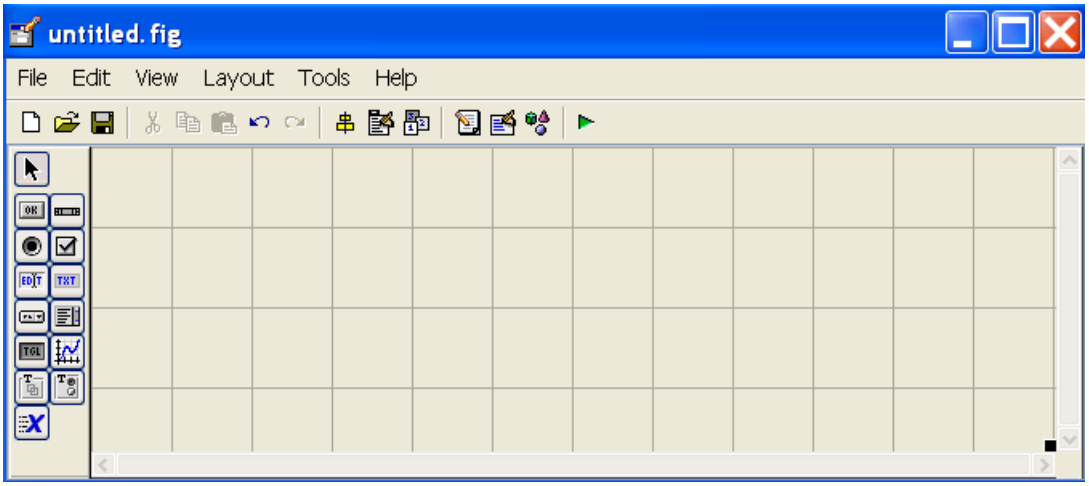


figure 35: GUIDE background

Once this window is opened, the only things we have to do is to set the graphical interface that we want and to save the application.

II.4.2 Application GUIDE code

When the application implemented in GUIDE is saved, it is created another file with the same name which contains all internal code. This internal code contains the functions of the objects that have been included in the window of the application. GUIDE defines

the functions when we add an object in the graphical work-space, and the only thing it is necessary to do is to include the code of the functions that the button has to execute.

In the following lines is shown the function of the four buttons of the user interface implemented in the application of this thesis.

RUN PROCESS

The function of the button that start the process:

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)

control_value = [1 1];
controlloop(control_value);
MAIN
disp('the process is RUNNING')
```

STOP PROCESS

The function of the button to stop the process completely:

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)

control_value = [1 0]
controlloop(control_value);
disp('the process is STOPED')
```

READ FROM DATABASE

Function of the button that simulates the STAR controller to check the synchronization before reading the data from the database:

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)

valueP = [1 0]
writeXML_synchro(valueP);
disp('the process is READING')
```

WRITE IN DATABASE

Function of the button that simulate the STAR controller to check the synchronization before writing the results of the simulation in the database:

```
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)

valueP = [0 1]
writeXML_synchro(valueP);
disp('the process is WRITING')
```

APPENDIX III

Appendixes to CHAPTER 4 (Data Exchange)

This appendix tries to clarify some concepts mentioned in "*CHAPTER 4: Data Exchange*".
The chapter is divided in three points: "*Web Services*", "*SOAP protocol*" and "*<org.apache.xerces.dom.DocumentImpl> class*". These three points give us a brief sight about *Web Services*, *SOAP protocol* and the class "*org.apache.xerces.dom.DocumentImpl*", so that, check the bibliography if you are really interested in going further in these topics.

III.1 Web Services

Web Services can be defined as network accessible interface to application functionality, built using standard Internet Technologies. It means that an application is a *Web Service* if it can be accessed using a standard protocols as HTTP, XML, SMTP or Jabber, or a combination of them, over the network.

A more technically definition can be:

- ***Web Service***: interface positioned between the application code and the user of that code. It acts as an abstraction layer, separating the platform and programming-language-specific details of how the application code is actually invoked. This standarized layer means that any language that supports the web service can access the application's functionality.

The abstraction layer provided by the Web Service means that it does not matter if the application services and the browser are written using different languages (Java, C++, Perl, .NET), or one is deployed on an Unix box and the other on Windows, the Web Service will be capable of sending and receiving messages, and always using a standard Internet protocol to program it.

A web service consists in two parts:

- ***Service Proxy***: that decodes the requests into calls into the application code.
- ***Service Listener***: that speaks the transport protocol (SOAP in the case of this thesis) and receives incoming requests.

The web services are divided in two parts, but to implement them it is made through the layering of five types of technologies, organized into layers that build upon one another. This division is called "Web Service Technology Stack" and the five layers that compound the Web Services are, starting from the down layer to the up layer: "*network layer*", "*transport layer*", "*packaging layer*", "*description layer*" and "*discovery layer*".

Each part of the web services stack addresses a separate business problem, so it is not necessary to rewrite significant chunks of the infrastructure when a new layer of the stack is needed, to support a new form of exchanging information.

To finish with the introduction to *Web Services*, some of the most important standarizations initiatives currently being pursued in programming *Web Services* are:

- XML protocol
- XKMS
- SAML (Security Assertions Markup Language): it is an XML grammar for expressing the occurrence of security events.
- XML-Dsig (XML digital signatures)
- XML-Enc (XML encryption)
- XSD

- P3P
- WSFL (Web Services Flow Language): it is an extension to WSDL.
- Jabber: asynchronous transport protocol used in peer-to-peer applications.
- ebXML: built to use SOAP.

III.2 SOAP Protocol

The point before talks about Web Services and what is this. Here there is an introduction about SOAP protocol, the protocol used by the company (Veolia Water System) to write Web Services.

A definition of SOAP can be: a standardized packaging for the messages shared by applications. The SOAP's specification defines a simple XML-based envelope for transferring information, and a set of rules for translating application and platform-specific data types into XML representations. So it can be deduced that SOAP is an application of the XML specification. It relies heavily on XML standards like XML Schema and XML Namespace for its definition and function (check the World Wide Web Consortium's web site at http://www.w3c.org to find more information about these specifications).

So, to understand how works SOAP, it is necessary to understand XML messaging. It is where applications exchange information using XML documents. It provides a flexible way for applications to communicate. The exchange of information through XML documents forms the basis of SOAP. The fundamental idea is that two applications may openly share information using message encoded in a way that both applications understand, not taking care about the operative system, the programming code or another technical limitation. SOAP provides a standard way to structure XML messages.

SOAP has two related applications: RPC and EDI.

- Remote Procedure Call (RPC) is the basis of distributed computing, the way for one program to make a procedure call on another, passing arguments and receiving return values.
- Electronic Document Interchange (EDI) is basis of automated business transactions, defining a standard format and interpretation of financial and commercial documents and messages.

But, it is not enough saying that the server and the client are using XML both of them. It is necessary to define too:

- The types of information we are exchanging.
- How that information is to be expressed as XML.
- How to actually go about sending that information.

SOAP provides these conventions that help to decode the information given.

The structure of a SOAP message consists of an envelope containing an optional header and a required body:

- The header contains blocks of information relevant to how the message is to be processed.
- The body contains the actual message to be delivered and processed.

There have been several versions of the SOAP specification. The last version, SOAP version 1.2, represents the first fruits of the World Wide Web Consortium's (W3C) effort to standardize an XML-based packaging protocol for web services. The W3C chose SOAP as the basis for that effort. The previous version of SOAP, Version 1.1, is still widely used.

III.3 "org.apache.xerces.dom.DocumentImpl" class

The package "*org.apache.xerces.dom*" uses **DocumentImpl**, this is because the class is called *org.apache.xerces.dom.DocumentImpl*. **DocumentImpl** can be used by other packages: "org.apache.html.dom", "org.apache.wml.dom" and "org.apache.xerces.parses".

The public class **DocumentImpl** is the Document interface that represents the entire HTML or XML document. Conceptually, it is the root of the document tree, and provides the primary access to the document's data.
Since elements, text nodes, comments, processing instructions, etc. cannot exist outside the context of a Document, the Document interface also contains the factory methods needed to create these objects. The Node objects created have an *ownerDocument* attribute which associates them with the Document within whose context they were created.
The **DocumentImpl** class also implements the DOM Level 2 DocumentTraversal interface. This interface is comprised of factory methods needed to create *NodeIterators* and *TreeWalkers*. The process of creating *NodeIterator* objects also adds these references to this document. After finishing with an iterator it is important to remove the object using the remove methods in this implementation. This allows the release of the references from the iterator objects to the DOM Nodes.

The uses of "*DocumentImpl"* in "*org.apache.xerces.dom*":

SUBCLASSES:

- ***DeferredDocumentImpl***: the document interface represents the entire HTML or XML document. Conceptually, it is the root of the document tree, and provides the primary access to the document's data. Since elements, text nodes, comments, processing instructions, etc. cannot exist outside the context of a Document, the Document interface also contains the factory methods needed to create these objects. The Node objects created have a ownerDocument attribute which associates them with the Document within whose context they were created.

CONSTRUCTORS in "*org.apache.xerces.dom*" **with parameters of type** *DocumentImp*:

- ***RangeImpl(****DocumentImpl document****):*** Clients must use *DocumentRange.createRange()*, because it registers the Range with the document, so it can be fixed-up.

- ***NodeIteratorImpl(****DocumentImpl document, Node root, int whatToShow, NodeFilter nodeFilter, boolean entityReferenceExpansion****)***: *DefaultNodeIterator* implements a *NodeIterator*, which iterates a DOM tree in the expected depth first way. The *whatToShow* and filter functionality is implemented as expected. This class also has method *removeNode* to enable iterator "fix-up" on DOM remove. It is expected that the DOM implementation call *removeNode* right before the actual DOM transformation. If not called by the DOM, the client could call it before doing the removal.

At last the methods of the class ***DocumentImpl*** are exposed bellow (All information about the methods of the *org.apache.xerces.dom.DocumentImpl* class expose here has been extracted from http://xerces.apache.org/xerces-j/apiDocs/org/apache/xerces/dom/DocumentImpl.html):

```
addEventListener(NodeImpl node,             java.lang.String type,
                 EventListener listener, boolean useCapture)
cloneNode(boolean deep)
createEvent(java.lang.String type)
createNodeIterator(Node root,  int whatToShow,  NodeFilter filter,
                 boolean entityReferenceExpansion)
createNodeIterator(Node root, short whatToShow, NodeFilter filter)
createRange()
createTreeWalker(Node root,   int whatToShow,   NodeFilter filter,
                 boolean entityReferenceExpansion)
createTreeWalker(Node root, short whatToShow, NodeFilter filter)
dispatchAggregateEvents(NodeImpl node,     AttrImpl enclosingAttr,
                 java.lang.String oldvalue, short change)
dispatchAggregateEvents(NodeImpl node,
                 org.apache.xerces.dom.DocumentImpl.EnclosingAttr
                 ea)
dispatchEvent(NodeImpl node, Event event)
dispatchEventToSubtree(NodeImpl node, Node n, Event e)
getEventListeners(NodeImpl n)
getImplementation()
getUserData(NodeImpl n)
removeEventListener(NodeImpl node,          java.lang.String type,
                 EventListener listener, boolean useCapture)
saveEnclosingAttr(NodeImpl node)
setEventListeners(NodeImpl n, java.util.Vector listeners)
setUserData(NodeImpl n, java.lang.Object data)
```

APPENDIX IV

Appendixes to CHAPTER 5 (Modelling Batch Processes with Matlab)

This appendix contains the full code of the functions not shown in *chapter 5*. The main parts of the code were explained in *chapter 5*, so the reader does not find any other information related with the S-functions and the model implemented for modelling Batch Processes.
To go further with the researching it is recommended to check the bibliography.

IV.1 S-function “settlerB_batchM.c” code

In this point is exposed the code programmed in the S-function to model batch processes with Matlab:

```
/*
 * settler1d_reac is a C-file S-function for defining a reactive 10 layer
sedimentation tank model
 * Compatibility: ASM1 model
 * The model is developed based on the settler1dv4.c model of Ulf Jeppsson
(IEA, LTH, Sweden)
 *
 * 25 October 2003
 * Krist V. Gernaey, CAPEC, Dept. of Chemical Engineering, DTU, Denmark
 *
 * Modified on 29 August 2006
 * by Josep Carrasco Martínez, Dept. of Industrial Electrical Engineering and
Automation, LTH, Sweden. Model modified to use in "batch processes"
 */

#define S_FUNCTION_NAME settlerB_batchM

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

#define XINIT   ssGetArg(S,0)      /* initial values                    */
#define SEDPAR   ssGetArg(S,1)     /* parameters sedimentation model    */
#define DIM     ssGetArg(S,2)      /* dimensions clarifier              */
#define LAYER   ssGetArg(S,3)      /* Number of layers                  */
#define ASMPAR   ssGetArg(S,4)     /* parameters activated sludge model */
#define SOSAT   ssGetArg(S,5)

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
  ssSetNumContStates(  S, 141); /*   number   of   continuous   states,   ASM1
              components + TSS for each settler layer + total height for the
              vessel*/
  ssSetNumDiscStates(    S, 0); /* number of discrete states         */
  ssSetNumInputs(       S, 19);  /* number of inputs, 13 ASM1 components + TSS
                                 + influent flow rate + Kla + J_down_up +
                                 Q_output+ Q_u waste sludge flowrate */
  ssSetNumOutputs(     S, 144);  /* number of outputs: 141 states 13 ASM1
                                 components + TSS (for each layer)+ + height
                                 Q_e + Q_out + Q_u*/
  ssSetDirectFeedThrough(S, 1); /* direct feedthrough flag           */
  ssSetNumSampleTimes(   S, 1); /* number of sample times            */
  ssSetNumSFcnParams(    S, 6); /* number of input arguments         */
  ssSetNumRWork(         S, 0); /* number of real work vector elements */
  ssSetNumIWork(         S, 0); /* number of integer work vector elements*/
  ssSetNumPWork(         S, 0); /* number of pointer work vector elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
```

```
/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
int i;

for (i = 0; i < 141; i++) {
   x0[i] = mxGetPr(XINIT)[i];
}
}

/*
 * mdlOutputs - compute the outputs
 */

static  void  mdlOutputs(double  *y,  double  *x,  double  *u,
                         SimStruct *S, int tid)
{
int i;
int j;
    /* Clarifier state variables */
    for (i=0;i<10;i++){
        for (j=0;j<14;j++){
        y[(i*14)+j]=x[i+j*10];    //we have written 14 because now we have 15
                                     states variables per each layer
        }                          //the order of outputs is from 1 to 14 layer
                                     1, from 15 to 28 layer 2, etc
    }                             //the inputs is, from 1 to 10 the SI of all
                                     layers,  from  11  to  20  the  Ss  of  all
                                     layers, etc
   y[140] = x[140];

   /* Flow rates out of the vessel */
   if ((u[14]>0)&&(u[17]==0)){
       y[141]=u[14]-u[18];       /* Q_e  */
   }
   if ((u[14]==0)&&(u[17]>0)){
       y[141]=u[17]-u[18];       /* Q_e  */
   }
   else y[141]=0;

   y[142]=u[18];                 /* Q_u   */
   y[143]=u[17];                 /* Q_out */
 }

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int
                      tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static  void  mdlDerivatives(double  *dx,  double  *x,  double  *u,
                             SimStruct *S, int tid)
{

double v0_max, v0, r_h, r_p, f_ns, X_t, area, h, feedlayer, volume;
int n;  //number of layers
double Q_f, Q_e, Q_u, v_up, v_dn, v_in, eps;
```

```
int i;
double vs[10];
double Js[11];
double Jstemp[10];
double Jflow[11];
double mu_H, K_S, K_OH, K_NO, b_H, mu_A, K_NH, K_OA, b_A, ny_g, k_a, k_h, K_X,
       ny_h;
double  Y_H,  Y_A,  f_P,  i_XB,  i_XP,  X_I2TSS,  X_S2TSS,  X_BH2TSS,  X_BA2TSS,
       X_P2TSS ;
double  proc1[10],  proc2[10],  proc3[10],  proc4[10],  proc5[10],  proc6[10],
       proc7[10], proc8[10];
double  reac1[10],  reac2[10],  reac3[10],  reac4[10],  reac5[10],  reac6[10],
       reac7[10], reac8[10];
double reac9[10], reac10[10], reac11[10], reac12[10], reac13[10], reac14[10];
double xtemp[150];
double SO_sat; //to calculate rSO
double KLa, a, b, J_d_u;

v0_max = mxGetPr(SEDPAR)[0];
v0 = mxGetPr(SEDPAR)[1];
r_h = mxGetPr(SEDPAR)[2];
r_p = mxGetPr(SEDPAR)[3];
f_ns = mxGetPr(SEDPAR)[4];
X_t = mxGetPr(SEDPAR)[5];
area = mxGetPr(DIM)[0];
// h = mxGetPr(DIM)[1]/mxGetPr(LAYER)[1]; /* now h is introduced as initial
                                             conditions */

n = mxGetPr (LAYER)[1]; //number of layers
feedlayer = mxGetPr(LAYER)[0];
volume = area*mxGetPr(DIM)[1];

/* ASM1: Stoichiometric parameters */
mu_H = mxGetPr(ASMPAR)[0];
K_S = mxGetPr(ASMPAR)[1];
K_OH = mxGetPr(ASMPAR)[2];
K_NO = mxGetPr(ASMPAR)[3];
b_H = mxGetPr(ASMPAR)[4];
mu_A = mxGetPr(ASMPAR)[5];
K_NH = mxGetPr(ASMPAR)[6];
K_OA = mxGetPr(ASMPAR)[7];
b_A = mxGetPr(ASMPAR)[8];
ny_g = mxGetPr(ASMPAR)[9];
k_a = mxGetPr(ASMPAR)[10];
k_h = mxGetPr(ASMPAR)[11];
K_X = mxGetPr(ASMPAR)[12];
ny_h = mxGetPr(ASMPAR)[13];
Y_H = mxGetPr(ASMPAR)[14];
Y_A = mxGetPr(ASMPAR)[15];
f_P = mxGetPr(ASMPAR)[16];
i_XB = mxGetPr(ASMPAR)[17];
i_XP = mxGetPr(ASMPAR)[18];
X_I2TSS = mxGetPr(ASMPAR)[19];
X_S2TSS = mxGetPr(ASMPAR)[20];
X_BH2TSS = mxGetPr(ASMPAR)[21];
X_BA2TSS = mxGetPr(ASMPAR)[22];
X_P2TSS = mxGetPr(ASMPAR)[23];
SO_sat = mxGetPr(SOSAT)[0]; //to read SO_sat

eps = 0.01;
v_in = u[14]/area;
Q_f = u[14];
Q_u = u[18];
v_dn = Q_u/area;

/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    NEW    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */
// Equaction of effluent
   if ((u[14]>0)&&(u[17]==0)){
```

```
        Q_e = u[14]-Q_u;        /* Q_e  */
    }
    if ((u[14]==0)&&(u[17]>0)){
        Q_e = u[17]-Q_u;        /* Q_e  */
    }
    else Q_e = 0;

// Equation of the velocity up
    if ((u[14]>0)||(u[17]>0)){
        v_up = Q_e/area;
    }
    else v_up = -v_dn;

// The flux for filling, mixing and reaction phase,
// where the wastewater is well mixed
J_d_u = u[16];
/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%   */

for (i = 0; i < 141; i++) {
    if (x[i] < 0.0)
      xtemp[i] = 0.0;
    else
      xtemp[i] = x[i];
}

/* calculation of the sedimentation velocity for each of the layers */
for (i = 0; i < 10; i++) {
    vs[i] = v0*(exp(-r_h*(xtemp[i+130]-f_ns*u[13]))-exp(-r_p*(xtemp[i+130]-
              f_ns*u[13])));       /* u[13] = influent SS concentration */
    if (vs[i] > v0_max)
      vs[i] = v0_max;
    else if (vs[i] < 0.0)
      vs[i] = 0.0;
}

/* calculation of the sludge flux due to sedimentation for each layer (not
taking into account X limit) */
for (i = 0; i < 10; i++) {
    Jstemp[i] = vs[i]*xtemp[i+130];
}

/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%     NEW     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%   */
/* calculation of the sludge flux due to the liquid flow (upflow or downflow,
depending on layer) */
for (i = 0; i < 11; i++) {
    //bulk movement up to the feed layer
    if (i < (feedlayer-eps)){
        if ((u[14]>0)||(u[17]>0)){
            Jflow[i] = v_up*xtemp[i+130];
        }
        else{
            Jflow[i] = v_dn*xtemp[i-1+130];
        }
    }
    //bulk movement down to the feed layer and in the feed layer
    else
      Jflow[i] = v_dn*xtemp[i-1+130];
}
/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%   */

/* calculation of the sludge flux due to sedimentation for each layer */
Js[0] = 0.0;
Js[10] = 0.0;
for (i = 0; i < 10; i++) {
    if ((i < (feedlayer-1-eps)) && (xtemp[i+1+130] <= X_t))
      Js[i+1] = Jstemp[i];
    else if (Jstemp[i] < Jstemp[i+1])
      Js[i+1] = Jstemp[i];
```

```
   else
      Js[i+1] = Jstemp[i+1];
}

/* Reaction rates ASM1 model */

for (i = 0; i < 10; i++) {
proc1[i] = mu_H*(xtemp[i+10]/(K_S+xtemp[i+10]))*
          (xtemp[i+70]/(K_OH+xtemp[i+70]))*xtemp[i+40];
proc2[i] = mu_H*(xtemp[i+10]/(K_S+xtemp[i+10]))*(K_OH/(K_OH+xtemp[i+70]))*
          (xtemp[i+80]/(K_NO+xtemp[i+80]))*ny_g*xtemp[i+40];
proc3[i] = mu_A*(xtemp[i+90]/(K_NH+xtemp[i+90]))*
          (xtemp[i+70]/(K_OA+xtemp[i+70]))*xtemp[i+50];
proc4[i] = b_H*xtemp[i+40];
proc5[i] = b_A*xtemp[i+50];
proc6[i] = k_a*xtemp[i+100]*xtemp[i+40];
proc7[i] = k_h*((xtemp[i+30]/xtemp[i+40])/(K_X+(xtemp[i+30]/xtemp[i+40])))*
          ((xtemp[i+70]/(K_OH+xtemp[i+70]))+ny_h*(K_OH/(K_OH+xtemp[i+70]))*(xt
          emp[i+80]/(K_NO+xtemp[i+80])))*xtemp[i+40];
proc8[i] = proc7[i]*xtemp[i+110]/xtemp[i+30];

reac1[i] = 0.0;
reac2[i] = (-proc1[i]-proc2[i])/Y_H+proc7[i];
reac3[i] = 0.0;
reac4[i] = (1.0-f_P)*(proc4[i]+proc5[i])-proc7[i];
reac5[i] = proc1[i]+proc2[i]-proc4[i];
reac6[i] = proc3[i]-proc5[i];
reac7[i] = f_P*(proc4[i]+proc5[i]);
reac8[i] = -((1.0-Y_H)/Y_H)*proc1[i]-((4.57-Y_A)/Y_A)*proc3[i];
reac9[i] = -((1.0-Y_H)/(2.86*Y_H))*proc2[i]+proc3[i]/Y_A;
reac10[i] = -i_XB*(proc1[i]+proc2[i])-(i_XB+(1.0/Y_A))*proc3[i]+proc6[i];
reac11[i] = -proc6[i]+proc8[i];
reac12[i] = (i_XB-f_P*i_XP)*(proc4[i]+proc5[i])-proc8[i];
reac13[i] = -i_XB/14.0*proc1[i]+((1.0-Y_H)/(14.0*2.86*Y_H)-
           (i_XB/14.0))*proc2[i]-
           ((i_XB/14.0)+1.0/(7.0*Y_A))*proc3[i]+proc6[i]/14.0;
reac14[i] = X_I2TSS*reac3[i]+X_S2TSS*reac4[i]+X_BH2TSS*reac5[i]+
           X_BA2TSS*reac6[i]+X_P2TSS*reac7[i];

}

/* ASM1 model component balances over the layers */
/* ASM1: [Si Ss Xi Xs Xbh Xba Xp So Sno Snh Snd Xnd Salk TSS Q_in]   */
/* New state variable => dH/dt*/

/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%    NEW    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */
/* height of the tank: H*/

    dx[140] = (u[14]-u[17])/area; /*the heigth of the vessel*/
    h = x[140]/n;                 /*the heigth of each layer*/

/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */

/* soluble component S_I */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i]  = (-v_up*xtemp[i] +v_up*xtemp[i+1])/h+reac1[i];
   else if (i > (feedlayer-eps))
      dx[i]  = (v_dn*xtemp[i-1] -v_dn*xtemp[i])/h+reac1[i];
   else
      dx[i]  = (v_in*u[0] -v_up*xtemp[i] -v_dn*xtemp[i])/h+reac1[i];
}

/* soluble component S_S */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+10]  = (-v_up*xtemp[i+10] +v_up*xtemp[i+1+10])/h +reac2[i];
   else if (i > (feedlayer-eps))
```

```
      dx[i+10]  = (v_dn*xtemp[i-1+10] -v_dn*xtemp[i+10])/h +reac2[i];
   else
      dx[i+10]  = (v_in*u[1] -v_up*xtemp[i+10] -v_dn*xtemp[i+10])/h +reac2[i];
}

/* particulate component X_I */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+20] = ((xtemp[i+20]/xtemp[i+130])*(-Jflow[i]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+20]/xtemp[i-
                 1+130])*Js[i]+(xtemp[i+1+20]/xtemp[i+1+130])*Jflow[i+1])/h +
                 (xtemp[i+1+20]/xtemp[i+1+130])*J_d_u + reac3[i];
   else if (i > (feedlayer-eps))
      dx[i+20] = ((xtemp[i+20]/xtemp[i+130])*(-Jflow[i+1]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+20]/xtemp[i-1+130])*(Jflow[i]+Js[i]))/h +
                 (xtemp[i-1+20]/xtemp[i-1+130])*J_d_u + reac3[i];
   else
      dx[i+20] = ((xtemp[i+20]/xtemp[i+130])*(-Jflow[i]-Jflow[i+1]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+20]/xtemp[i-1+130])*Js[i]+v_in*u[2])/h +
                 (xtemp[i-1+20]/xtemp[i-1+130])*J_d_u + reac3[i];
}

/* particulate component X_S */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
       dx[i+30] = ((xtemp[i+30]/xtemp[i+130])*(-Jflow[i]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+30]/xtemp[i-
                 1+130])*Js[i]+(xtemp[i+1+30]/xtemp[i+1+130])*Jflow[i+1])/h +
                 (xtemp[i+1+30]/xtemp[i+1+130])*J_d_u + reac4[i];
   else if (i > (feedlayer-eps))
       dx[i+30] = ((xtemp[i+30]/xtemp[i+130])*(-Jflow[i+1]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+30]/xtemp[i-1+130])*(Jflow[i]+Js[i]))/h +
                 (xtemp[i-1+30]/xtemp[i-1+130])*J_d_u + reac4[i];
   else
       dx[i+30] = ((xtemp[i+30]/xtemp[i+130])*(-Jflow[i]-Jflow[i+1]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+30]/xtemp[i-1+130])*Js[i]+v_in*u[3])/h +
                 (xtemp[i-1+30]/xtemp[i-1+130])*J_d_u + reac4[i];
}

/* particulate component X_BH */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
       dx[i+40] = ((xtemp[i+40]/xtemp[i+130])*(-Jflow[i]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+40]/xtemp[i-
                 1+130])*Js[i]+(xtemp[i+1+40]/xtemp[i+1+130])*Jflow[i+1])/h +
                 (xtemp[i+1+40]/xtemp[i+1+130])*J_d_u + reac5[i];
   else if (i > (feedlayer-eps))
       dx[i+40] = ((xtemp[i+40]/xtemp[i+130])*(-Jflow[i+1]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+40]/xtemp[i-1+130])*(Jflow[i]+Js[i]))/h +
                 (xtemp[i-1+40]/xtemp[i-1+130])*J_d_u + reac5[i];
   else
       dx[i+40] = ((xtemp[i+40]/xtemp[i+130])*(-Jflow[i]-Jflow[i+1]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+40]/xtemp[i-1+130])*Js[i]+v_in*u[4])/h +
                 (xtemp[i-1+40]/xtemp[i-1+130])*J_d_u + reac5[i];
}

/* particulate component X_BA */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
       dx[i+50] = ((xtemp[i+50]/xtemp[i+130])*(-Jflow[i]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+50]/xtemp[i-
                 1+130])*Js[i]+(xtemp[i+1+50]/xtemp[i+1+130])*Jflow[i+1])/h +
                 (xtemp[i+1+50]/xtemp[i+1+130])*J_d_u + reac6[i];
   else if (i > (feedlayer-eps))
       dx[i+50] = ((xtemp[i+50]/xtemp[i+130])*(-Jflow[i+1]-Js[i+1]-
                 J_d_u)+(xtemp[i-1+50]/xtemp[i-1+130])*(Jflow[i]+Js[i]))/h +
                 (xtemp[i-1+50]/xtemp[i-1+130])*J_d_u + reac6[i];
   else
```

```
       dx[i+50] = ((xtemp[i+50]/xtemp[i+130])*(-Jflow[i]-Jflow[i+1]-Js[i+1]-
                  J_d_u)+(xtemp[i-1+50]/xtemp[i-1+130])*Js[i]+v_in*u[5])/h +
                  (xtemp[i-1+50]/xtemp[i-1+130])*J_d_u + reac6[i];
}

/* particulate component X_P */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
       dx[i+60] = ((xtemp[i+60]/xtemp[i+130])*(-Jflow[i]-Js[i+1]-
                  J_d_u)+(xtemp[i-1+60]/xtemp[i-
                  1+130])*Js[i]+(xtemp[i+1+60]/xtemp[i+1+130])*Jflow[i+1])/h +
                  (xtemp[i+1+60]/xtemp[i+1+130])*J_d_u + reac7[i];
   else if (i > (feedlayer-eps))
       dx[i+60] = ((xtemp[i+60]/xtemp[i+130])*(-Jflow[i+1]-Js[i+1]-
                  J_d_u)+(xtemp[i-1+60]/xtemp[i-1+130])*(Jflow[i]+Js[i]))/h +
                  (xtemp[i-1+60]/xtemp[i-1+130])*J_d_u + reac7[i];
   else
       dx[i+60] = ((xtemp[i+60]/xtemp[i+130])*(-Jflow[i]-Jflow[i+1]-Js[i+1]-
                  J_d_u)+(xtemp[i-1+60]/xtemp[i-1+130])*Js[i]+v_in*u[6])/h +
                  (xtemp[i-1+60]/xtemp[i-1+130])*J_d_u + reac7[i];
}

/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%    NEW    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */
/* soluble component S_O */

KLa = u[15];

for (i = 0; i < 10; i++) {
   if(v_up==-v_dn){
       a = xtemp[i-1+70];
       b = xtemp[i+70];
   }
   else{
       a = xtemp[i+70];
       b = xtemp[i+1+70];
   }

   if (i < (feedlayer-1-eps))
      dx[i+70]  = (-v_up*a +v_up*b)/h +reac8[i] + KLa*(SO_sat-xtemp[i+70]);
                                 //we have added rSO (Kla!=0)
   else if (i > (feedlayer-eps))
      dx[i+70]  = (v_dn*xtemp[i-1+70] -v_dn*xtemp[i+70])/h +reac8[i] +
                  KLa*(SO_sat-xtemp[i+70]);       //we have added rSO (Kla!=0)
   else
      dx[i+70]  = ((v_in*u[7] -v_up*a -v_dn*xtemp[i+70])/h +reac8[i]) +
                  KLa*(SO_sat-xtemp[i+70]); //we have added rSO (Kla!=0)
}
/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */

/* soluble component S_NO */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+80]  = (-v_up*xtemp[i+80] +v_up*xtemp[i+1+80])/h +reac9[i];
   else if (i > (feedlayer-eps))
      dx[i+80]  = (v_dn*xtemp[i-1+80] -v_dn*xtemp[i+80])/h +reac9[i];
   else
      dx[i+80]  = (v_in*u[8] -v_up*xtemp[i+80] -v_dn*xtemp[i+80])/h +reac9[i];
}

/* soluble component S_NH */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+90]  = (-v_up*xtemp[i+90] +v_up*xtemp[i+1+90])/h +reac10[i];
   else if (i > (feedlayer-eps))
      dx[i+90]  = (v_dn*xtemp[i-1+90] -v_dn*xtemp[i+90])/h +reac10[i];
   else
      dx[i+90]  = (v_in*u[9] -v_up*xtemp[i+90] -v_dn*xtemp[i+90])/h
                  +reac10[i];
```

```
}

/* soluble component S_ND */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+100]  = (-v_up*xtemp[i+100] +v_up*xtemp[i+1+100])/h +reac11[i];
   else if (i > (feedlayer-eps))
      dx[i+100]  = (v_dn*xtemp[i-1+100] -v_dn*xtemp[i+100])/h +reac11[i];
   else
      dx[i+100]  = (v_in*u[10] -v_up*xtemp[i+100] -v_dn*xtemp[i+100])/h
                    +reac11[i];
}

/* particulate component X_ND */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+110] = ((xtemp[i+110]/xtemp[i+130])*(-Jflow[i]-Js[i+1]-
                  J_d_u)+(xtemp[i-1+110]/xtemp[i-
                  1+130])*Js[i]+(xtemp[i+1+110]/xtemp[i+1+130])*Jflow[i+1])/h
                  + (xtemp[i+1+110]/xtemp[i+1+130])*J_d_u + reac12[i];
   else if (i > (feedlayer-eps))
      dx[i+110] = ((xtemp[i+110]/xtemp[i+130])*(-Jflow[i+1]-Js[i+1]-
                  J_d_u)+(xtemp[i-1+110]/xtemp[i-1+130])*(Jflow[i]+Js[i]))/h +
                  (xtemp[i-1+110]/xtemp[i-1+130])*J_d_u + reac12[i];
   else
      dx[i+110] = ((xtemp[i+110]/xtemp[i+130])*(-Jflow[i]-Jflow[i+1]-Js[i+1]-
                  J_d_u)+(xtemp[i-1+110]/xtemp[i-1+130])*Js[i]+v_in*u[11])/h +
                  (xtemp[i-1+110]/xtemp[i-1+130])*J_d_u + reac12[i];
}

/* soluble component S_ALK */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+120]  = (-v_up*xtemp[i+120] +v_up*xtemp[i+1+120])/h +reac13[i];
   else if (i > (feedlayer-eps))
      dx[i+120]  = (v_dn*xtemp[i-1+120] -v_dn*xtemp[i+120])/h +reac13[i];
   else
      dx[i+120]  = (v_in*u[12] -v_up*xtemp[i+120] -v_dn*xtemp[i+120])/h
                    +reac13[i];
}

/* particulate component X_TSS */
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+130] = ((-Jflow[i]-Js[i+1])+Js[i]+Jflow[i+1])/h +reac14[i];
   else if (i > (feedlayer-eps))
      dx[i+130] = ((-Jflow[i+1]-Js[i+1])+(Jflow[i]+Js[i]))/h +reac14[i];
   else
      dx[i+130] = ((-Jflow[i]-Jflow[i+1]-Js[i+1])+Js[i]+v_in*u[13])/h
                  +reac14[i];
}

}

/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

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

PART IV

TITLE: ***Bibliography***

BIBLIOGRAPHY

There is no bibliography referred to the "*APPENDIX I*", "*APPENDIX II*" and "*APPENDIX III*", but it does not mean that it is inexistent. The bibliography is organized so that the documents consulted (either books or web links) for one chapter have been used for the chapter, the appendixes of the chapter or both.

CHAPTER 2: Wastewater Treatment Processes

[1] *Gustaf Olsson & Bob Newel* (1999). **"Wastewater Treatment Systems. Modelling, Diagnosis and Control"**. IWA Publishing, chapters 2, 3 and 4.

[2] *Ulf Jeppsson* (1996). **"Modelling Aspects of Wastewater Treatment Processes"**. IEA. LTH, pp. 191-228

[3] *Mark J. Hummer* (1986). **"Water and Wastewater Treatment Technology"**. Prentice-Hall International Editions. (used in 2.1 in chapter 2 for process description)

CHAPTER 4: Dataexchange

[4] *James Snell, Doug Tidwell and Pavel Kulchenko* (2001). **"Programming web services with SOAP"**. O'reilly. (chapter 1, chapter 2 and chapter 3)

[5] Link about XML and SOAP protocol to program web services: www.w3.org/2000/xp/Group, from the main page www.w3.org,

[6] Java API for XML: (http://java.sun.com/webservices/jaxp/dist/1.1/docs/api/)

[7] Methods of the class "***org.apache.xerces.dom.DocumentImpl***" used by the functions "**xmlread**" and "**xmlwrite**": (http://xerces.apache.org/xerces-j/apiDocs/org/apache/xerces/dom/class-use/DocumentImpl.html

CHAPTER 5: Modelling batch processes with Matlab

[8] *The MATH WORKS Inc.* (2000). **"SIMULINK. Dynamic System Simulation for MATLAB®"**. Using Simulink Version 4.

[9] *The MATH WORKS Inc.* (2000). **"SIMULINK. Dynamic System Simulation for MATLAB®"**. Writing S-Functions Version 4.

[10] *A product of COST Action 624 and COST Action 682* (2002). **"The COST simulation benchmark. Description and simulator manual"**. EUROPEAN COMISION, Directorate-General for Research, European research area: structural aspects - COST, chapters 1, 2, 3, 8