Techneau, 06.
June 2007

Conceptual design modelling framework

Techneau, 06.
June 2007

TECHEAU

Conceptual design modelling framework

TECHNEAU is an Integrated Project Funded by the European Commission under the Sixth Framework Programme, Sustainable Development, Global Change and Ecosystems Thematic Priority Area (contractnumber 018320).

Colofon

Title: CONCEPTUAL DESIGN MODELLING FRAMEWORK

Author(s)
Luuk Rietveld, Petra Ross (TUD);
Glenn Dillon, Jeremy Dudley (WRc)

Quality Assurance
By Luuk Rietveld (TUD), Glenn Dillon (WRc)

Deliverable number
D 5.4.3.

This report is: PU = Public

Contents

1 Introduction

The objective of TECHNEAU Work Package 5.4 "Development of a water treatment plant simulator" is to produce a European platform for modelling of drinking water treatment processes. This document - the third deliverable of the project (D5.4.3) - describes the conceptual design of the modelling platform.

The state-of-the-art of existing water treatment simulators was reviewed in the first deliverable (D5.4.1) of this project. The review concluded that OTTER, developed by WRc, Stimela, developed by TU-Delft/DHV and Metrex, developed by the University of Duisburg/IWW were the most appropriate existing platforms to act as a foundation for future development and integration. The second deliverable (D5.4.2) described and discussed the methodology for integration. It was concluded that a prototype of the software would be developed, with WRc taking the lead in the development and technical descriptions of the software, and TU-Delft working on the technical descriptions of models to be incorporated in the prototype.

During a Work Package meeting in November 2006, agreement was reached with regard to the water quality parameters (determinands) and treatment processes - including an inventory of the development state of the models - to be incorporated into the prototype. At the same meeting, the boundary conditions and starting points of the modelling framework were agreed.

The present report gives a description of the conceptual design of the new modelling platform and an overview of the selected treatment processes and determinands.

2 Approach to the development of the modelling platform

2.1 Phasing of development of the modelling platform

The approach to the development of the modelling platform was to first produce a 0-version prototype. This prototype will include an impression of the framework including a rudimentary visualisation, well-developed models transferred to the framework and a reduced list of determinands that can be modelled with the prototype.

In the following versions of the software, the number of models and the list of determinands will be extended. This will be dependent on the needs of the different TECHNEAU partners and the level of knowledge and development of the processes models. In addition, the framework will be adapted based on the experiences of the selected users of the prototype. This will include visualisation and web-access.

2.2 Availability of the prototype of the new modelling platform

The prototype will be available in two forms. The first form will be meant for other TECHNEAU users who will use the model in an executable form. These users will evaluate the platform and provide suggestions for improvements and extensions. The second form will be a desktop version for model developer. With this version it is possible to upload data with a XML format, exporting data for MPC applications and Command line running of the models for optimisation.

2.3 Documentation and reporting

Three types of documents (in addition to the TECHNEAU progress reports) will be produced supporting the development of the modelling platform:

- The first document is a description of the software/framework (WRc-JD).
- The second document is a compilation of descriptions of the model algorithms (TUD-LR/PR).
- The third document is a compilation of modelling reports executed before and in the context of the TECHNEAU programme (TUD-LR/PR).

These documents are 'living' reports and will be extended whenever new developments take place.

3 SIMULATOR DESCRIPTION

3.1 Requirements

The program will be built upon the following requirements. First, the program will be free to use through access via the web, either directly with a web browser or by download. Free-to-use also implies that no third-party commercial packages will be involved at the end-user level. Second, it is important that water quality measurements, such as laboratory data, and simulated results are easy to import and export, respectively. This will be achieved by the construction of a user-friendly GUI and the provision of user documentation and help facilities. Third, the program will be able to communicate with other platforms (e.g. SCADA systems, distribution models). This is achieved by defining an application programming interface (API) for the simulator, which will result in the run-time version being accessible and controllable by other software. Fourth, the program must be easily extendable with new determinands and process models, to accommodate the discovery of existing and emerging substances at ever smaller concentrations and the subsequent development of existing or new treatment processes. And finally, it must be possible for run-time users to implement their own models in the simulator. Therefore an import mechanism for user-written models is provided.

As a result of these requirements, it has been decided to keep the simulator run-time and the user interface separate. The process flowsheet is created in the user interface, where one process icon relates to one process model. The chosen process model defines the model parameters and therefore the user interface for parameter entry must be "dynamic". In this way, model builders can add models without compiling. The simulator run-time uses standard entry points in the process model run-time file, which is (computer) language independent. Finally, all the results are written to the results data files for post processing.

3.2 Raw water

OTTER differentiates between water quality and water flow rate (water demand), unlike Stimela and Metrex. This division reflects the nature of water sources – the seasonal variation in a river abstraction point, for example, should not require the creation of new data files merely to increase or reduce the abstraction flow. It is possible, for example with an aquifer, for the abstraction flow to affect the water quality, but usually this information is not available for pure waterworks modelling.

It is recommended that the OTTER approach of raw water quality as one process model, and raw water flow as a second, be followed. The OTTER approach of keeping all data in a single large file simplifies data handling, but can make working with the raw data file difficult. We suggest that the influent dialogue editor contain an initial row, providing check boxes to

enable/disable the profile, and a higher-level check box to display disabled determinands.

3.3 Process model definition

To achieve the required flexibility for the process models, each process model is defined using two files. The files have the same unique name, but different extensions:

- An XML-file (xml): with all model parameter descriptions. The file contains all the information that is necessary for the model to be used in the simulator and user interface (the description determines what the user will see, e.g. name of the model, parameter of the model, number of stages, static data, process parameters, control inputs/outputs and the initial state).
- A dynamic linked library file (dll): the run-time model, to be integrated in the simulation engine, where the actual model calculations are programmed.

The model description also holds the list of determinands addressed in the model. The list is split into two groups: the determinands necessary to run the model and optional determinands that, if present, can be calculated.

3.4 Determinands/Water quality parameters

Drinking water must be free from colour, taste, odour and turbidity. In addition, disinfection must be sufficient, water must be chemically stable (e.g. as indicated by saturation index, SI), biologically stable (e.g. as indicated by assimilable organic carbon, AOC) and free from disinfection by-products (e.g. bromate) and organic micro-pollutants (e.g. pesticides) (Rietveld *et al*., 2006).

In the prototype of the modelling platform, not all water quality parameters and treatment processes are addressed. A selection has been made based on the availability of models and the occurrence of processes in the UK and the Netherlands. There is a long list of pesticides, micro-organisms and heavy metals defined in the EU legislation. It is intended to have some representatives of each of the different water quality groups.

A data dictionary for the determinands is used, where the short name is fixed and the long name can be translated into whatever is appropriate. The determinands are divided into four classes:

- Health-related parameters;
- Parameters listed in the Drinking Water Directive;
- Parameters for process operation;
- Aesthetic parameters (e.g. turbidity, colour).

Two lists are created, one comprising all the determinands that have been defined by the various process models available in the modelling library, and the second for those determinands that are required by the chosen unit processes. When the raw water profile is being created the user dialogue will only display those determinands that are required for the flowsheet. Previously, the user has been either presented a display of all potential

determinands, whether required or not (as in OTTER), and left to decide which are actually needed for the current flowsheet, or required to define those determinands that are expected to be used, ahead of creating the flowsheet, as in Stimela. In both cases this has imposed an additional layer of expert knowledge on the user.

It is possible that a given process model is defined to handle only a small number of determinands, and a complete flowsheet may result in there being determinands that are unknown to a given process model. Because the simulator will allow user-written models, addressing local problems, this issue of unexpected determinands will potentially affect the standard library models as well as user models. At the current stage of the software development this is recognised as an issue, which will be addressed first at the superficial level of requiring that all determinands define if they are soluble or particulate, so that as a minimum any model can query the nature of these unknown determinands. During the course of the TECHNEAU project, it is anticipated that this mechanism will be extended further, to requiring that determinands also provide data such as Henry's law coefficients. The default for each process will be that the determinands pass through unchanged.

3.5 Raw water data entry

Water quality is a specialised model that does not use the standard dialogue editor.

The water quality parameters to be entered depend on the models in the flowsheet. Data can be entered at irregular times and saved as a file for subsequent re-use. The data points can use either linear interpolation or step-interpolation (zero-order hold). The data can be imported from text files, to use external data sources.

The date/time is relative, where '0' is the beginning of the simulation and is required for all determinands.

3.6 Results data

To be able to post-process the results, a binary file is used to store all the results for any given process or stream. A binary file was felt to be the simplest solution around the issues of decimal separator (period in the UK, comma in the Netherlands) and record separator (usually a comma in the UK but a semicolon or space in the Netherlands). Other post-processing programs, if required, can access the results file. All result files have the same format:

- Header: header length (integer[1]), number of columns (integer);
- For each column: length of column header string (integer), column heading (string);
- The rest of the data are floating point numbers (IEEE double precision);

[1] Integers are 4-byte signed representations; floating point 8-byte double precision using the IEEE format; and strings a byte stream using ANSI encoding.

- Time is in days and is always Column 1.

In the user interface it will be possible to plot simulation data in combination with real plant data, to compare simulation and reality.

4 LIST OF PROCESSES AND DETERMINANDS

In the first version of the water treatment plant simulator, the focus will be on conventional treatment. A listing of the processes included, together with an indication of the parameters that are addressed, is given in Table 1.

Table 1. Water treatment processes with a selection of parameters

Process	Parameters
Aeration	O_2, CO_2, CH_4, H_2S, $Fe^{2+,3+,T}$
Rapid sand filtration	SS, $Fe^{2+,3+,T}$, Mn, NH_4 (T/DOC)
Ozonation	T/DOC, AOC, UV254, Pathogens (rotavirus, *Giardia*, E. coli), organic micropollutants (pesticides), BrO_3^-
Softening	CO_2, HCO_3^-, pH, EGV, temperature, Ca, Mg
GAC/PAC	Organic micropollutants, T/DOC, AOC, O_2
Coagulation/flocculation	SS, T/DOC, pH, heavy metals, UV_{254}, turbidity
Sedimentation/ flotation	SS, T/DOC, pH, heavy metals, UV_{254}, turbidity
Chlorine	Residual Cl_2 (free/combined), THMs
Conditioning	pH

In future versions, the water treatment simulator may be extended, for example with advanced oxidation processes and membrane filtration.

5 MODELLING APPROACH

A treatment process consists of the following mechanisms (see Figure 1) which can be described by basic differential equations (Rietveld, 2005):

- Flow of water containing compounds through the reactor.
- Equilibrium between water and gas or solid phase.
- Transfer of compounds to gas or solid phase.
- Decay in the water and/or solid phase.
- Mass balance between water, gas and solid phase (continuity law).

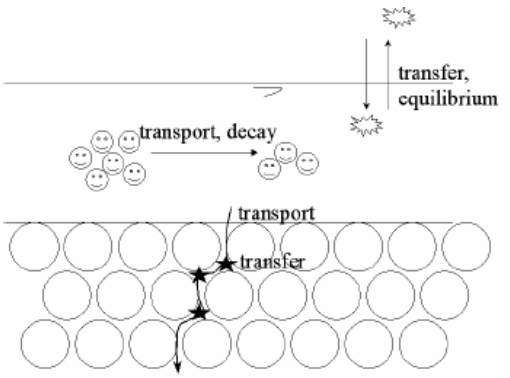

Figure 1. Mechanisms in treatment processes (Rietveld, 2005)

For the removal of solids from the water phase in a filter, e.g. neglecting decay and advection in the solid and water phase, and considering the water velocity in the pores, the overall equations for transport of solids:

$$\frac{\partial c}{\partial t} - D_x \frac{\partial^2 c}{\partial x^2} + u \frac{\partial c}{\partial x} + f_1(c) + f_2(c, c_s) = 0$$

$$\frac{\partial c_s}{\partial t} - D_{x,s} \frac{\partial^2 c_s}{\partial x^2} + u_s \frac{\partial c_s}{\partial x} + f_{1,s}(c_s) + f_{2,g}(c, c_s) = 0$$

are reduced to:

$$\frac{\partial c}{\partial t} - D_x \frac{\partial^2 c}{\partial x^2} + \frac{u}{\varepsilon} \frac{\partial c}{\partial x} + f_2(c, c_s) = 0$$

$$\frac{\partial c_s}{\partial t} - D_{x,s} \frac{\partial^2 c_s}{\partial x^2} + f_{2,s}(c, c_s) = 0$$

Plug flow can be described as follows:

$$\frac{\partial c_1}{dt} = -u \frac{\partial c_1}{dx} = -u\left(\frac{c_1 - c_0}{\Delta x} + \frac{\Delta x}{2} \frac{\partial^2 c_1}{\partial x^2}\right)$$

The last term is the discretisation error, also called numerical dispersion. Comparing the equation with the equation for advection and dispersion, and when $u\Delta x/2=Dx$, numerical dispersion is equal to physical dispersion and

can thus be used to describe real flow conditions. The equation for transport is thus written as a tanks-in-series model, considering the unit element as a continuous stirred tank reactor (CSTR). The more unit elements, the better plug flow is approached (Levenspiel, 1998).

The basic discrete equations of a treatment process, neglecting degradation in the water phase, are then given in the following ordinary differential equations (ODEs):

$$\frac{dc_i}{dt} = -u\frac{c_i - c_{i-1}}{\Delta x} - k_2(c_{ei} - c_i)$$

$$\frac{dc_{s_i}}{dt} = -p \cdot u\frac{c_i - c_{i-1}}{\Delta x}$$

Where Dx = dispersion coefficient in water (m^2/s)
Δx = height of unit element (m)
c_{i-1} = influent concentration (g/m^3)
c_i = effluent concentration (g/m^3)
c_e = equilibrium concentration (g/m^3)
u = flow velocity through reactor (m/s)
k_2 = transfer rate (s^{-1})

By integrating the equation numerically, the processes are dynamically modelled. To be able to solve these types of problems, the processes are subdivided into unit CSTRs, as shown in Figure 2.

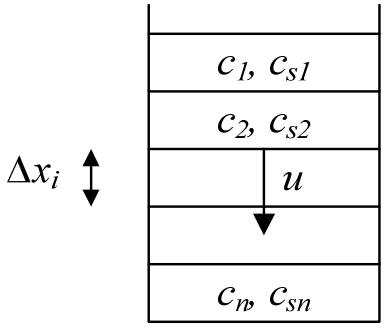

Figure 2. Schematic representation of unit CSTRs

For a combination of three unit elements in series, the system of ODEs becomes:

$$\begin{pmatrix} \frac{dc_1}{dt} \\ \frac{dc_2}{dt} \\ \frac{dc_3}{dt} \end{pmatrix} = \frac{u}{\Delta x} \begin{pmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} - k_2 . \begin{pmatrix} c_{e1} - c_1 \\ c_{e2} - c_2 \\ c_{e3} - c_3 \end{pmatrix}$$

$$\begin{pmatrix} \frac{dc_{s1}}{dt} \\ \frac{dc_{s2}}{dt} \\ \frac{dc_{s3}}{dt} \end{pmatrix} = \frac{u}{\Delta x} \begin{pmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

The models containing these equations are programmed in the desired language. A snapshot of how the models are programmed in C is given in Figure 3.

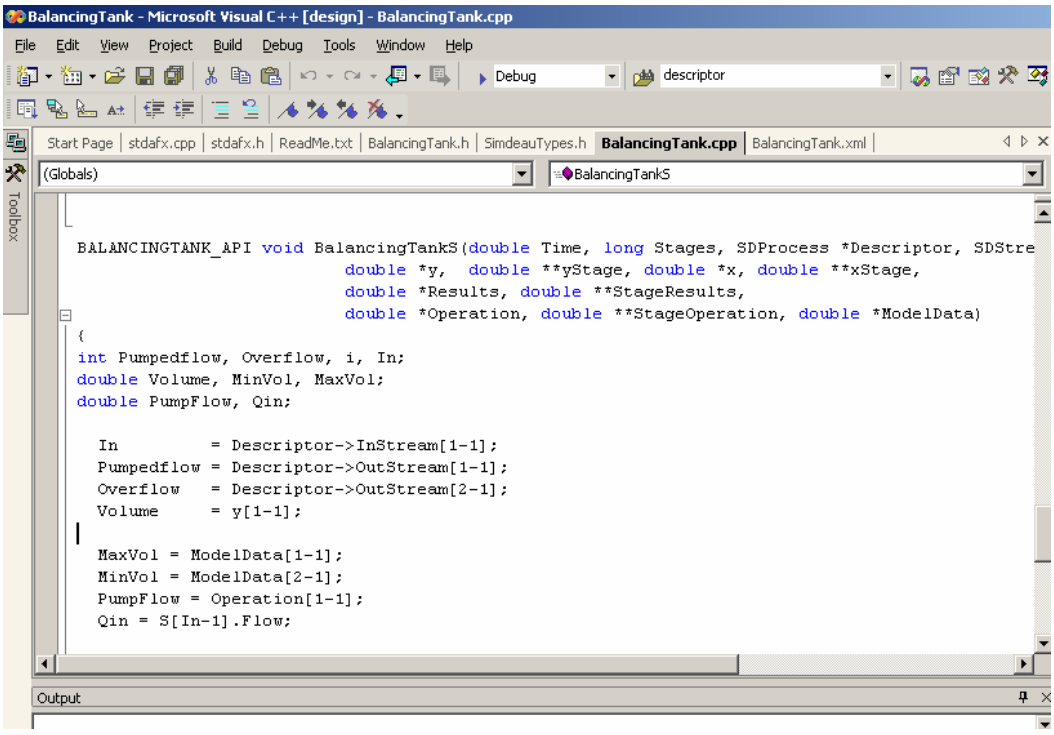

Figure 3. Layout of a model programmed in C

The total number of determinands is available to each model, and should be checked, because the model may only be set up to explicitly represent a subset of the determinands. Determinands may be accessed by a short dictionary name (e.g., "NTU" for turbidity) or by an integer in the range *1 .. number of determinands*. Access by name is intended to get and set values for those determinands that are explicit in a model, and access by integer for those remaining determinands that were not explicitly represented by the model.

6 LAYOUT OF THE WATER TREATMENT PLANT SIMULATOR

6.1 Layout creation

User interfaces for flowsheeting programs, whether in the chemical, wastewater or clean water industries, have generally converged to the approach where unit process models can be selected from a 'toolbox' and dragged to a 'drawing board' where the models are subsequently connected (see Figure 4). This keeps all the layout data in one location, rather than requiring the user to switch between different windows holding different components of the user interface.

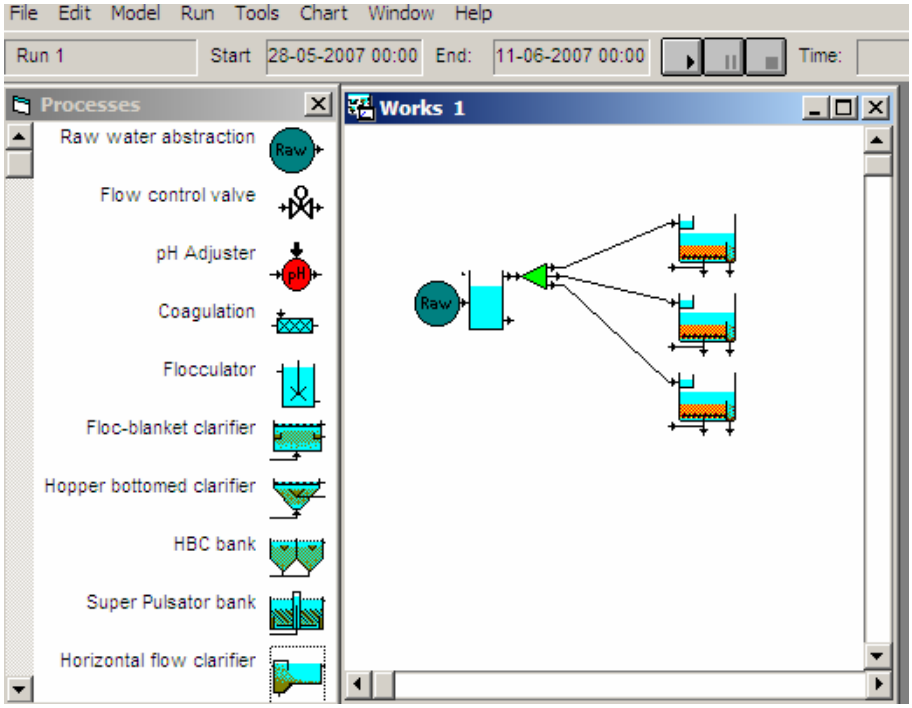

Figure 4. User interface layout

6.2 Model set-up

With the layout and the process units specified, the required model data is then entered, e.g. surface areas and volumes, operational characteristics and calibration data (Figure 5).

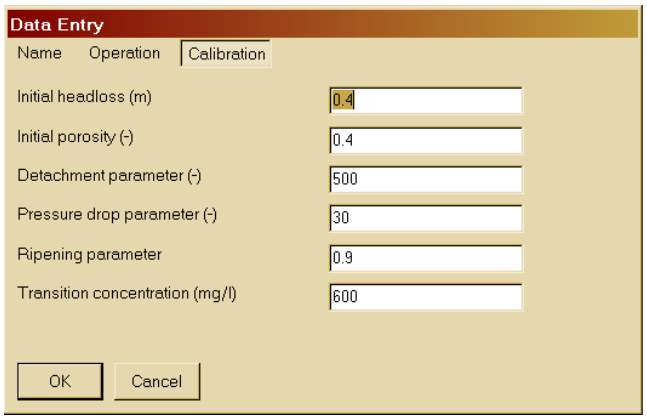

Figure 5. Example of a user dialogue

All this dialogue data is read in from the model data file, stored in an XML format, which allows users to translate dialogue prompts, if required (Figure 6). This file contains the name and location of the model DLL file, and the names of the entry points. Library models start out as user-defined models, and the only differentiation between them is that library models become part of a standard DLL rather than existing as a collection of separate DLLs.

Figure 6. Example of model data file

6.3 Simulation

Each process model is compiled and stored in a separate Windows DLL. The DLLs are loaded at the simulation run. Stream data can be displayed during the course of the simulation. At present, only one integration technique is supported, a Runge-Kutta code chosen for its suitability for moderately stiff problems (Abdulle and Medovikov, 2001). DAEs are thus not supported, but the structure has been chosen to make addition of a DAE code (e.g., DASPK, Brown et al., 1994) easy if required in future.

Because each model is compiled to a separate DLL, it should be easy to extend the simulator. Models are stored as at least two files in a standard location. The first file is the compiled DLL; the second file contains information about the model, including the DLL filename and the two entry points to the DLL. One entry point is used to define and calculate the differential equations, and the second is used to assign stream output variables.

At the start of the simulation, a process ordering algorithm is called, with the aim of minimising the number of recycle loops that would need iterative solution. Because the program does dynamic simulation, the recycles usually vary little from one output time to another, so that a simple iterative scheme is adequate to resolve the recycles. For this reason, the process ordering algorithm is also simple, rather than the more complex approaches found for steady-state simulators (Westerberg *et al.*, 1979).

The core modelling engine is written in Fortran, to simplify re-use of the availability of a large body of integration (and optimisation) routines readily available at Netlib. The use of dynamically-linked model files, rather than the more traditional approach of all models contained within one program, has required the use of one Fortran extension, Cray pointers, to map the DLL routines to the Fortran code. Cray pointers provide, for Fortran, the ability to carry out the same kind of indirection as provided by pointers in the C language, and are supported by the majority of Fortran compilers . This dynamic linking makes it easy for user-written models to be added to the overall program.

6.4 Results

Results are available for both processes and the connecting streams, displayed graphically or in tabular format (Figure 7).

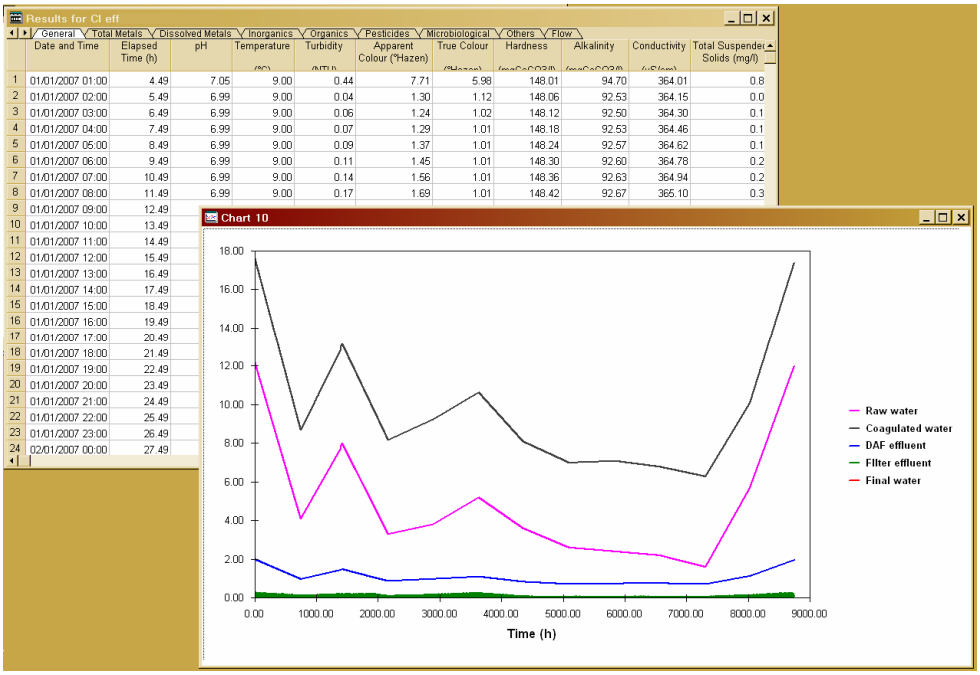

Figure 7. Tabular and graphical results views

7 CONCLUSIONS

A first step towards a European Water Treatment Simulator has been made by building upon experiences of existing modelling platforms. The key features are that the treatment simulator will be free to use, easy to handle and that the simulator can be easily extended with new processes and determinands. For individual processes, models will be developed including the relevant water quality parameters. By calibrating and validating the models for different raw water qualities and water treatment plants (with data provided by, e.g., Waternet located in the Netherlands and Riga located in Latvia), the simulator should be robust and widely applicable. In addition, within the TECHNEAU programme new models will be developed by Sintef, WRc, Delft University of Technology and the University of Riga, tackling the issues of NOM removal, biological treatment and membrane filtration.

The European Water Treatment Simulator enables users to construct a virtual water treatment plant for drinking water production and to simulate its performance based on measured raw water quality. Its free availability should result in a wider uptake of the use of modelling as an adjunct of more traditional methods for water treatment works design, operation, trouble-shooting and operator training.

8 REFERENCES

Abdulle, A and Medovikov, A, 2001. "Second order Cheby-shev methods based on orthogonal polynomials", Nu-merische Mathematik, 90(1), 1-18.

Brown, PN, Hindmarsh, AC, and Petzold, LR, 1994, "Using Krylov Methods in the Solution of Large-Scale Differen-tial-Algebraic Systems", SIAM J. Sci. Comp., 15, pp. 1467-1488

Levenspiel, O, 1998, Chemical Reaction Engineering, Wiley

Rietveld, L C, 2005. Improving operation of drinking water through modelling. PhD thesis, Faculty of Civil Engineer-ing, Technical University Delft, Delft, The Netherlands.

Rietveld, L C, Helm, A W C van der, Schagen, K S van, Aa, L T J van der and Dijk, J C van, 2006. "Integral deterministic modelling of drinking water treatment", submitted for Journal of Water Supply: Research and Technology – Aqua.

Westerberg, AW, Hutchinson, P, Motard, RL and Winter, P, 1979, Process Flowsheeting, Cambridge University Press.

I APPENDIX: PROTOTYPE SOFTWARE LISTING

This appendix provides the basic user interface specifications for the DLL files. The specification language is Fortran 95.

Definition of types PROCESS and STREAM

```
module ModelTypes
USE StreamMatch
  integer, parameter:: iLen   = 255
  integer, parameter:: iLenp1 = iLen + 1
  type ProcessModel
    integer:: ModelID
    integer:: ModelPointerDiff
    integer:: ModelPointerAlloc
  end type

  integer, parameter:: MAX_INLETS = 3
  integer, parameter:: MAX_OUTLETS = MAX_INLETS
  type Process
    integer:: ModelID
    integer:: ModelIndex
    integer:: InStream(MAX_INLETS)
    integer:: OutStream(MAX_OUTLETS)
    integer:: Stages
    integer:: y
    integer:: yStage
    integer:: x
    integer:: xStage
    integer:: Results
    integer:: StageResults
    integer:: Operation
    integer:: StageOperation
    integer:: ModelData
  end type

  type Stream
    double precision:: flow
    double precision:: t        ! Temperature
    double precision:: pH
    double precision:: Value(MAX_DETERMINANDS)
  end type

 integer, parameter:: LIBRARY_MODELS = 1
end module
```

Interfaces for Differential (locSubDY) and Stream assignment (locSubAssign) subroutines

```
  interface
    subroutine locDLLSubDY(t, Stages, Descriptor, Streams,      &
                           y,  yStage, x, xStage,               &
                           dy, dyStage, Results, StageResults,  &
                           Operation, StageOperation, ModelData)
    use CoreTypes
    implicit none
    double precision:: t
    integer         :: Stages
```

```
  type(Process)   :: Descriptor
  type(Stream)    :: Streams(*)
  double precision:: y(*),         yStage(Stages, *)
  double precision:: x(*),         xStage(Stages, *)
  double precision:: dy(*),        dyStage(Stages, *)
  double precision:: Results(*),   StageResults(Stages, *)
  double precision:: Operation(*), StageOperation(Stages, *)
  double precision:: ModelData(*)
  end subroutine locDLLSubDY

  subroutine locDLLSubAssign(t, Stages, Descriptor, Streams,      &
                             y,  yStage, x, xStage,               &
                             Results, StageResults,               &
                             Operation, StageOperation, ModelData)
  use CoreTypes
  implicit none
  double precision:: t
  integer         :: Stages
  type(Process)   :: Descriptor
  type(Stream)    :: Streams(*)
  double precision:: y(*),         yStage(Stages, *)
  double precision:: x(*),         xStage(Stages, *)
  double precision:: Results(*),   StageResults(Stages, *)
  double precision:: Operation(*), StageOperation(Stages, *)
  double precision:: ModelData(*)
  end subroutine locDLLSubAssign
end interface
```

II APPENDIX: EXAMPLE CODE FOR A BALANCING TANK MODEL

```
!  Balancing Tank.f90
!
!  FUNCTIONS/SUBROUTINES exported from Balancing Tank.dll:
!  Balancing Tank      - subroutine
!
! pH model
!   Work with excess of anion/cation
!   Given pH -> [H+], [OH-]
!   Then assign [Cat+], [An-] to close electroneutrality
!  Then we have
!   d [Cat] = ... Standard equation
!   d [An ] = ... Standard equation
!
!  Electroneutrality
!     [Cat+] + [H+] = [An-] + [OH-]
!  Equilibrium
!     [H+] [OH-] = Kw
!     [Cat+][H+] + [H+]**2 = [H+][An-] + Kw
!
! Solution options:
!     [1] Solve for Cat', An'
!           Solve for [H+] explicitly
!           This is an ODE system with an explicit algebraic
!           embedded equation.
!     [2] Solve for Cat', An', [H+]
!           This is a DAE system, of index-1
!           Usually easy to solve, providing
!           consistent initial conditions are used
!     [3] Solve for Cat', An', H' -- see below
!           This has turned the DAE system into an
!           ODE system. The DAE system was index-1 because
!           only one level of differentiation of [H+] was needed
!           to remove the algebraic constraints.
!
!   Taking time derivative
!     H Cat' + Cat H' + 2 H H' = H An' + An H'
!     H' (2 H + Cat - An) = H (An' - Cat')
!     H' (2 H + Kw/H - H) = H (An' - Cat')
!     H'  = (An' - Cat') H / (H + Kw/H)
!         = (An' - Cat') H**2 / (H**2 + Kw)
!
```

Definition for differential equation calculations

```
subroutine BalancingTank(Time, Stages, Descriptor, S,         &
                         y,  yStage, x, xStage,               &
                         dy, dyStage, Results, StageResults,  &
                         Operation, StageOperation, ModelData)
use CoreTypes
implicit none
double precision:: Time
integer         :: Stages
type(PROCESS)   :: Descriptor
type(STREAM)    :: S(*)
double precision:: y(*),         yStage(Stages, *)
double precision:: x(*),         xStage(Stages, *)
double precision:: dy(*),        dyStage(Stages, *)
double precision:: Results(*),   StageResults(Stages, *)
double precision:: Operation(*), StageOperation(Stages, *)
double precision:: ModelData(*)
```

```
  ! Expose subroutine Balancing Tank to users of this DLL
  !
  ! Export routine from DLL and ensure that it has
  ! the name 'BalancingTank'
  !DEC$ ATTRIBUTES DLLEXPORT::Balancing Tank
  !DEC$ ATTRIBUTES ALIAS: 'BalancingTank':: BalancingTank

  ! ***
  ! *** MUST LINK AGAINST WATSIMCORE.LIB
  ! *** AND MUST DECLARE WHAT FUNCTIONS ARE BEING IMPORTED IN
  ! *** FROM THAT LIBRARY.
  ! ***
```

Compiler-specific procedure for exporting subroutines from a DLL

```
  !DEC$ ATTRIBUTES DLLIMPORT:: NumberOfDeterminands
  !DEC$ ATTRIBUTES ALIAS: 'NumberOfDeterminands'::
NumberOfDeterminands
  ! Variables
  double precision:: MinVol, MaxVol, PumpFlow
  double precision:: Volume, Qoverflow, Qpump, Qin
  double precision:: H, OH, An, Cat, T, K
  integer         :: i, in
  integer         :: n
  integer, external:: NumberOfDeterminands

  ! Body of Balancing Tank
  MaxVol = ModelData(1)
  MinVol = ModelData(2)
  PumpFlow = Operation(1)

  in = Descriptor%InStream(1)
  Qin = s(in)%Flow

  n = NumberOfDeterminands() + 6
  Volume = y(1)
  Qoverflow = 0d0
  Qpump = PumpFlow
  if (Volume .le. MinVol) then
    Qpump = 0d0
  elseif (Volume .ge. MaxVol) then
    Qoverflow = max(0d0, Qin - Qpump)
  end if

 ! Y( 1) = Volume
 !    2  = Hydraulic age
 !    3 = Temperature
 !    4 = Cations
 !    5 = Anions
 !    6 = [H+]
 !    7 .. NumberOfComponents = Water quality parameters
  dy(1) = Qin - Qpump - Qoverflow
  if (Volume .gt. 0d0) then
    dy(2) = 1d0 - Qin * y(2) / Volume
    dy(3) = Qin * (s(in)%T - y(3)) / Volume
    T   = y(3)
    K   = Kw(T)
    H   = 10d0 ** (-s(in).pH)
    OH  = K / H
!
! Anions and cations are calculated from pH, rather than being
specified
! directly. However, if they were needed then they would be acquired
by the following:
!
!  An  = GetStreamValue(s(in), "ANION")
!  Cat = GetStreamValue(s(in), "CATION")
!
```

```
    An  = max(0d0, H - OH)
    Cat = max(0d0, OH - H)
! Typical concentrations are of the order 1E-6 - 1E-8
! Multiply by 1E10 to attain some scaling
    dy(4) = Qin * (1d10* Cat - y(4)) / Volume
    dy(5) = Qin * (1d10* An  - y(5)) / Volume
    H     = y(6)
    dy(6) = (dy(5) - dy(6)) * H**2 / (H**2 + K*1d20)
    if (n .gt. 6) then
      dy(7:n) = Qin * (s(in)%Value(:) - y(7:n)) / Volume
    end if
  else
    dy(2)  = 0d0
    dy(3:n) = 0d0
  end if

contains
  function Kw(T) result(K)
  double precision:: T, K
    K = 1d-14
  end function
end subroutine BalancingTank

!DEC$ ATTRIBUTES DLLEXPORT::Balancing TankS
!DEC$ ATTRIBUTES ALIAS: 'BalancingTankS':: BalancingTankS
```

Definition for stream assignment subroutine

```
subroutine BalancingTankS(t, Stages, Descriptor, S,           &
                          y,  yStage, x, xStage,              &
                          Results, StageResults,              &
                          Operation, StageOperation, ModelData)
!DEC$ ATTRIBUTES DLLIMPORT:: NumberOfDeterminands
!DEC$ ATTRIBUTES ALIAS: 'NumberOfDeterminands':: NumberOfDeterminands
use CoreTypes
implicit none
double precision:: t
integer         :: Stages
type(PROCESS)   :: Descriptor
type(STREAM)    :: S(*)
double precision:: y(*),         yStage(Stages, *)
double precision:: x(*),         xStage(Stages, *)
double precision:: Results(*),   StageResults(Stages, *)
double precision:: Operation(*), StageOperation(Stages, *)
double precision:: ModelData(*)
!
! Local variables
!
integer:: PumpedFlow, Overflow, n, In
double precision:: Volume, MinVol, MaxVol
double precision:: PumpFlow, Qin
integer, external:: NumberOfDeterminands

  in         = Descriptor%InStream(1)
  pumpedflow = Descriptor%OutStream(1)
  overflow   = Descriptor%OutStream(2)
  volume     = y(1)

  MaxVol = ModelData(1)
  MinVol = ModelData(2)
  PumpFlow = Operation(1)
  Qin = s(in)%Flow
  n = NumberOfDeterminands() + 6

  if (volume .le. MinVol) then
    s(pumpedflow) = 0d0
    s(overflow) = 0d0
  else
```

```
    s(pumpedflow)%flow = PumpFlow
    s(pumpedflow)%t    = y(3)
    s(pumpedflow)%ph   = -log10(y(6))
    if (n .gt. 6) then
      s(pumpedflow)%Value(:) = y(7:n)
    end if
    if (volume .lt. MaxVol) then
      s(overflow) = 0d0
    else
      s(overflow) = s(pumpedflow)
      s(overflow)%flow = Qin - PumpFlow
    end if
  endif
end subroutine BalancingTankS
```