WEST: Modelling biological wastewater treatment

Henk Vanhooren[*+], Jurgen Meirlaen[*], Youri Amerlinck[**], Filip Claeys[*], Hans Vangheluwe[*] and Peter A. Vanrolleghem[*]

[*] *BIOMATH, Ghent University, Coupure Links 653, B-9000 Gent, Belgium (biomath.rug.ac.be, peter.vanrolleghem@biomath.rug.ac.be)*

[**] *HEMMIS NV, Koning Leopold III-laan 2, B-8500 Kortrijk, Belgium (www.hemmis.com, ya@hemmis.com)*

[+] *current postal address: EPAS NV, Technologiepark 3, B-9052 Gent, Belgium, tel. + 32 9 241.56.18 fax + 32 9 221.82.18, www.epas.be)*

Abstract

Modelling is considered to be an inherent part of the design and operation of a wastewater treatment system. The models used in practice range from conceptual models and physical design models (lab-scale or pilot-scale reactors) to empirical or mechanistic mathematical models. These mathematical models can be used during the design, the operation and the optimisation of a wastewater treatment system. To do so, a good software tool is indispensable. WEST is a general modelling and simulation environment and can, together with a model base, be used for this task. The model base presented here is specific for biological wastewater treatment and is written in MSL-USER. In this high-level object-oriented language, the dynamics of systems can be represented along with symbolic information. In WEST's graphical modelling environment, the physical layout of the plant can be rebuilt, and each building block can be linked to a specific model from the model base. The graphical information is then combined with the information in the model base to produce MSL-EXEC-code, which can be compiled with a C++-compiler. In the experimentation environment, the user can design different experiments, such as simulations and optimisations of, for instance, designs, controllers and model fits to data (calibration).

Key Words

Biological Wastewater Treatment, Calibration, Knowledge Base, Model Specification, Optimisation, Simulation

Notation

b	*s*pecific decay rate [T^{-1}]
C_i	concentration of component *i* [$M\ L^{-3}$]
$K_{S,(i)}$	half-velocity constant (of component *i*) [$M\ L^{-3}$]
M_i	mass (of component *i*) [M]
r_j	process rate for the species *j* [$M\ L^{-3}T^{-1}$]
R_i	reaction rate of component *i* [$M\ L^{-3}T^{-1}$]
Q	flow rate [$L^3\ T^{-1}$]
S_i	concentration of soluble component *i* [$M\ L^{-3}$]
V	volume [L^3]
X_i	concentration of particulate component *i* or biomass [$M\ L^{-3}$]
$Y_{(i)}$	yield coefficient for growth (on substrate *i*) [$M\ M^{-1}$]

$\mu_{(j)}$ specific growth rate (of biomass species *j*) [T^{-1}]
ν_{ij} stoichiometric coefficient for the species *j* with respect to the substrate *i*
ρ_i density of component *i* [$M\ L^{-3}$]
Φ_{i_α} flux of component *i* in the flux at terminal α [$M\ T^{-1}$]

Subscripts

B Biomass
A Autotrophic bacteria
H Heterotrophic bacteria
O oxygen
S biodegradable substance

Modelling wastewater treatment: benefits and practical use

The problem of modelling and simulation of wastewater treatment plants (WWTP's) has been found important as a result of growing environmental awareness. Compared to the modelling of well-defined (such as electrical and mechanical) systems, modelling of ill-defined systems such as WWTP's is more complex. In particular, choosing the "right" model is a non-trivial task.

Modelling is an inherent part of the design of a wastewater treatment system. At the fundamental level, a design model may be merely conceptual. The engineer reduces the complex system he is dealing with to a conceptual image of how it functions. That image then determines the design approach. Often, however, the engineer recognises that the conceptual model alone does not provide sufficient information for design and thus he constructs a physical model, such as a lab-scale reactor or a pilot plant, on which various design ideas can be tested. Given sufficient time for testing, such an approach is entirely satisfactory. However, the engineer may find that limitations of time and money prevent exploration of all potentially feasible solutions. Consequently, he often turns to the use of mathematical models to further explore the feasible design space. He may devise empirical models, which incorporate a statistical approach to mimic the end results obtained by studies on the physical model or if his conceptual understanding expands sufficiently, he may attempt to formulate models based on mechanistic knowledge. These mechanistic models are the more powerful because they allow extrapolation of the design space to conditions beyond that experienced on the physical model. In this way, many potentially feasible solutions may be evaluated quickly and inexpensively, allowing only the most promising ones to be selected for actual testing in the physical model.

To be able to use mathematical – be it empirical or mechanistic – models, a good software tool to implement and simulate the models is indispensable. Several tools are available that can be applied to the modelling and simulation of wastewater treatment plants. Increasingly, the "system" modelled also transcends the WWTP and includes the "environment" (in the engineering sense). The WWTP model is then integrated in a conceptual model of the wastewater producing plant, the sewer system and the river (with its natural water purification properties or toxicity tolerance) in which the effluent is discharged (Meirlaen *et al.,* 2001).

Wastewater treatment practice has now progressed to the point where the removal of organic matter and nutrient removal by biological nitrification and denitrification and biological phosphorus removal, can be accomplished in a single system. The non-linear dynamics and properties of these biological processes are still not very well understood. As a consequence, a unique model cannot always be identified. This contrasts

to traditional mechanical and electrical systems where the model can be uniquely derived from physical laws. Also, the calibration of wastewater treatment models is particularly hard: many expensive experiments may be required to accurately determine model parameters. Yet, even with the limitations and difficulties stated above, modelling and simulation of wastewater treatment is considered useful (Henze *et al.*, 2000). Models are excellent tools to summarise and increase the understanding of complex interactions in biological systems. More quantitatively, they can be used to predict the dynamic response of the system to various disturbances.

Despite of their promising properties described above, the practical use of dynamic modelling of wastewater treatment is rather limited (Morgenroth *et al.*, 2000). Especially the labour and cost intensive calibration of WWTP models is considered hard to accomplish in practical situations. New methodologies are being developed to overcome this bottleneck (Petersen *et al.*, 2001). In some cases, *e.g.* when modelling biofilm wastewater treatment, the development of models that are able to describe the wastewater treatment system well enough to correctly predict system responses, but that are at the same time having a feasible complexity for simulation, is also problematic. However, the application field for good WWTP models is promising. They could be used to:

1. predict dynamic responses of the system to influent variations so as to develop strategies to optimise treatment plant operation. This can be done either off-line or with on-line 'real-time' simulations that are used for control and optimisation.
2. trouble shoot plant operation. Operators might be interested to use models in finding answers to practical questions.
3. integrate multiple processes. As mentioned above, the removal of organic matter, nitrogen and phosphorus, is accomplished in a single system nowadays. Models are a promising tool to help creating more understanding of the interactions between these processes.
4. design WWTP reactors. Models can be used to evaluate data from pilot-scale reactors and to predict performance of full-scale plants.

Introducing the WEST modelling and simulation environment

WEST (Wastewater Treatment Plant Engine for Simulation and Training) provides the modeller with a user-friendly platform to use existing models or to implement and test new models. Basically, WEST is a modelling and simulation environment for any kind of process that can be described as a structured collection of Differential Algebraic Equations (DAE's). Currently, however, WEST is mainly applied to the modelling and simulation of wastewater treatment plants (Vangheluwe *et al.*, 1998).

WEST is especially aimed at facilitating and optimising the implementation and re-use of knowledge in wastewater treatment models and to maximise the simulation speed and accuracy of simulations with the models used. However, the aims of modelling and simulation are in a sense contradictory. Hence, the WEST modelling and simulation environment makes a strict distinction between a *modelling environment* which aims to enable re-use of model knowledge and the *experimentation environment*, which aims to maximise accuracy and performance.

Next to these two user environments, the *model base* plays a central role in WEST. In this model base, models are described in MSL-USER (MSL stands for *model specification language*), a high level object-oriented declarative language specifically developed to incorporate models. The model base is aimed at

maximal re-use of existing knowledge and is therefore structured hierarchically. All re-usable knowledge – such as mass balances, physical units, default parameter values and applicable ranges – is thus defined centrally and can be re-used by an expert user to build new models. This indicates that WEST has an open structure in that the user is allowed to change existing models and define new ones as needed.

As depicted in Figure 1, the model base is loaded and all relevant information for the modeller is extracted from it when the modelling environment is started (step 1). Using the symbolic information in the model base, the 'atomic' models available in the model base are linked to a graphical representation. A hierarchical graphical editor (*HGE*) allows for the interactive composition of complex configurations from these basic graphical building blocks. Also the input-output structures (*terminals)* of the models are extracted from the model base so as to decide whether or not two models can be linked together in the HGE. For instance, a model for the activated sludge process can not directly be coupled to a river model, since the set of components used in these models to describe the substrates is not the same. In case such coupling need to be done, a explicit component convertor needs to be used. Next, the parameter set of the different models is loaded so that parameters of different models can be linked. For instance, the same yield coefficient can be used for all activated sludge tanks in a WWTP configuration. Finally, when a configuration has been built, the HGE starts from the information extracted from the model base and creates and outputs a *coupled model* in MSL-USER (step 2), which is automatically added to the model base for further use in new model exercises (step 3).

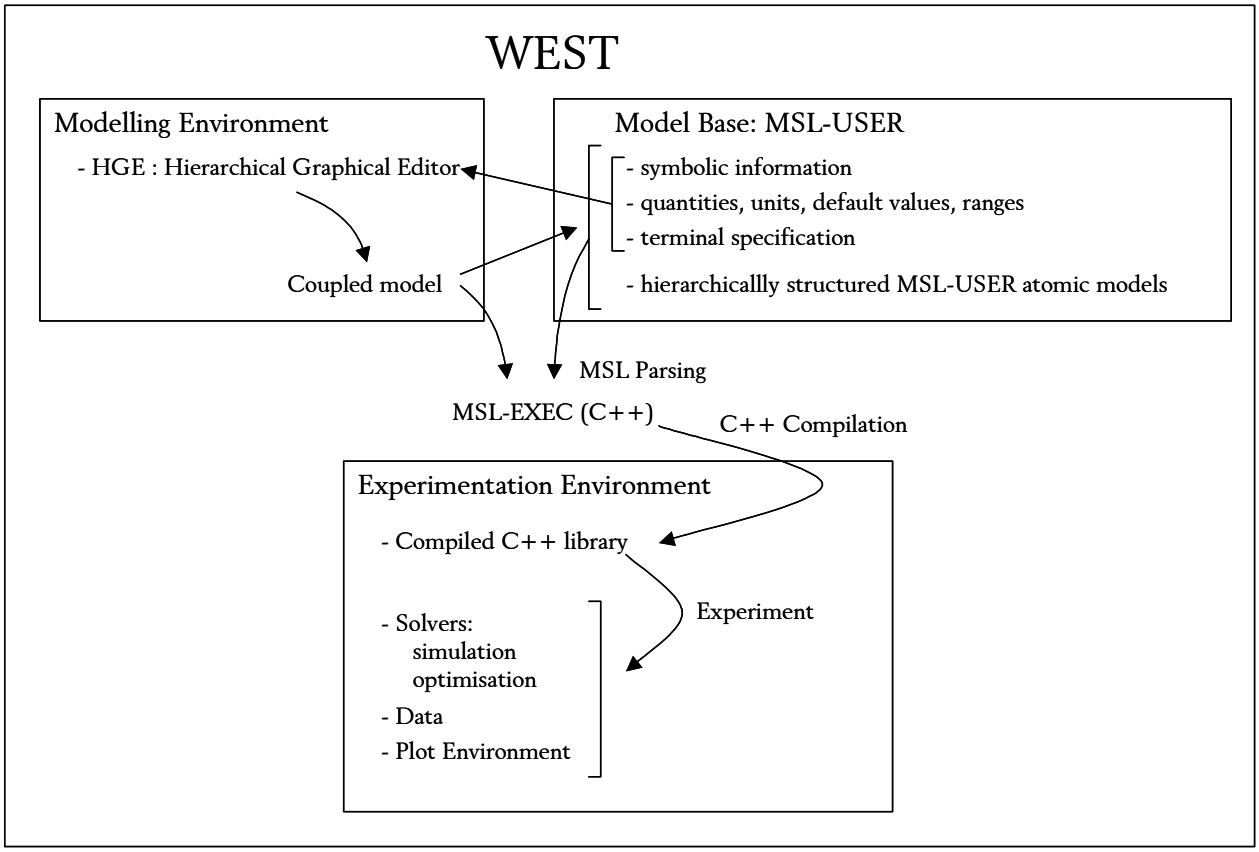

Figure 1: Functional WEST architecture

In a next step (step 4), the model parser generates low-level (C++) MSL-EXEC code, which after C++ compilation (step 5) can be used for execution within the experimentation environment. The parser therefore uses the coupled model together with the atomic model representations in the model base. These steps are

especially oriented towards simulation performance and accuracy. Finally, the solvers within the experimentation environment generate data which can be used for plotting, model calibration, process optimisation, output to file, etc. (step 6).

The model specification language MSL-USER

The language MSL-USER that is used in the WEST model base, is an object-oriented language, which allows for the declarative representation of the dynamics of systems. 'Declarative' means that the model (*what*) is presented without specifying *how* to solve it. As mentioned above, a compiler (MSL-parser) is provided to transform MSL-USER model representations into a low level representation (MSL-EXEC based on C++).

The MSL-USER parser is written in lex(flex), yacc(bison) and C++ and makes use of LEDA (Library of Efficient Data structures and Algorithms).MSL-USER follows the major principles of object-oriented programming in that it uses TYPES, CLASSES and OBJECTS to represent the hierarchy of the items in the model base. The relation between these representations can be visualised like a tree. Basically, types provide a way to describe the structure of an expression in the sense that it is a template to which classes and objects add more information. Indeed, a class is derived from a type definition, further defining the properties of the template. That way, classes provide a way to describe the behaviour of values. For example, a class in MSL-USER is mostly a type to which default values have been assigned. It is clear that one type can have multiple classes derived from it. A class is a template itself for the instantiation of objects that give final values to the structures defined. An object, however, can not only be instantiated from a class, but also directly from a type.

Apart from being object-oriented, MSL-USER is also a multi-abstraction language. It allows one to represent abstract models of the behaviour of systems using different methods of abstraction. This includes the possibility to make use of "abstractions" such as differential and algebraic equations, state transition functions, C++ code, ... (Vangheluwe, 2000).

Other characteristics of MSL-USER are:

- Re-use of models is possible thanks to the EXTENDS inheritance mechanism. This allows for the extension of an existing model. Thus, starting from generic models, a tree of extended models can be built.
- Classification is made possible through the SPECIALISES mechanism. Hereby, it is possible to indicate that a particular type is a sub-type of another type. This not only allows for classification, but also for rigorous type-checking.

Some examples will surely clarify the explanation above. The basic types found in MSL-USER are integer, real, string, char and boolean. Based on these basic types, a number of extended type structures were built. Some type structures are the Record type, the Vector type, the Enumerated type, … For example a vector type is used to specify vectors and matrices. A matrix can be specified as a vector of vectors. A column vector is declared as follows:

```
TYPE type_name = type[dimension ;];
```

An enumerated type is a type structure consisting of a set of unique identifiers called enumerators, and is declared as:

```
TYPE type_name = ENUM {id_1, id_2, ...,id_n} ;
```

These basic types and structure types can now be used to create user-defined types, such as UnitType, QuantityType and RealIntervalType. The first two are defined as strings while RealIntervalType is defined as a record of two real values and two booleans, describing if the bounds are included in the interval.

```
TYPE UnitType
"The type of physical units"
= String;

TYPE QuantityType
"The different physical quantities"
= String;


TYPE RealIntervalType
"Real Interval"
= RECORD
{
  lowerBound: Real;
  upperBound: Real;
  lowerIncluded: Boolean;
  upperIncluded: Boolean;
};
```

Furthermore, existing types can be extended. For example, the Record type can be extended with extra fields. In the following example, the ExtendedType is a type extended from BasicType.

```
TYPE BasicType "Basic type"
=
RECORD
{
   value: Real;
};
TYPE ExtendedType "Extended type"
EXTENDS BasicType WITH
RECORD
{
  unit: UnitType;
  quantity: QuantityType;
  interval: RealIntervalType;
};
```

The mechanism of specialisation is somewhat different. A class that is specialised from another class or type has the same signature, but the objects in the class are assigned (replaced). For example:

```
CLASS Concentration "A class for concentration" SPECIALISES ExtendedType :=
{:
   quantity  <- "Concentration";
   unit      <- "g/m^3";
   interval <- {: lowerBound <- 0; upperBound <- PLUS_INF :};
:};
```

A class such as Concentration, can further be instantiated as an object, where a value is assigned to one of the elements of the vector:

```
OBJ S_O_Sat "Oxygen saturation concentration"
: Concentration := {:value <- 8:};
```

MSL-USER thus allows one to express *physical* knowledge such as units (m, kg, …), quantity type (Mass, Length, …), boundary conditions, … The semantics of these are known by the parser which will check model consistency and, where appropriate, apply this knowledge in the translation to MSL-EXEC. Also some other object attributes are interesting to note here. When the value of a parameter or the initial condition of a variable depends on the value of other parameters, it is possible to declare this parameter or variable as *fixed.* In this case, the user cannot change its value in the experimentation environment. When, in an MSL-USER model, a parameter or variable object has the annotation *hidden* this object is not shown in the experimentation environment.

During the translation from MSL-USER to MSL-EXEC, the different abstractions used in the models created by the user will be translated into C++ representations. Algebraic equations and Differential equations (using the DERIV statement) will be recognised directly by the parser, since they are available in the MSL-USER library. Other built-in statements in MSL-USER are for example FOREACH, SUMOVER and IF-THEN-ELSE structures. Moreover, during the subsequent compilation of the generated MSL-EXEC code, some standard C libraries are automatically linked to the generated model. This way, functions that are not built-in in MSL and that are not defined in the MSL-USER function libraries can be used as long as they are available in these standard libraries. It is even possible to use user defined C++ functions.

Building the model base

To allow for computer aided model building and subsequent simulation/experimentation, a *model base* must be constructed. The models in this model base will be used for modular construction (*i.e.* by connecting component blocks as described above) of complex models describing the behaviour of WWTP's. The steps listed below form a general method for constructing a model base for any application domain:

1. Choose an appropriate level of abstraction.
2. Identify relevant quantities.
3. Identify input-output structures.
4. Build a model class hierarchy starting from general (conservation and constraint) laws and refining these for specific cases.

In the following paragraphs, these steps will be treated in more detail.

Level of abstraction

As is commonly the case, we will choose an appropriate level of abstraction, upon which Idealised Physical Models (IPM's) will be built. Idealised Physical Models (Broenink, 1990) represent behaviour at a certain level of abstraction. This often means using lumped parameter models (ordinary differential equations or ODE's), even though the physical system has a spatial distribution (which would require partial differential equation or PDE modelling), when the homogeneity assumption is a reasonable approximation.

Relevant quantities

Secondly, the quantities of interest must be identified. These quantities can be subsequently used to describe the *types* of entities used in modelling: constants, parameters, interface variables and state variables.

In MSL-USER, the type of physical quantities is encoded as a PhysicalQuantityType, a structure as given below:

```
TYPE PhysicalQuantityType
"The type of any physical quantity"
=
RECORD
{
  quantity   : QuantityType;
  unit       : UnitType;
  interval   : RealIntervalType;
  value      : Real;
  causality  : CausalityType;
};
```

For numerical computation purposes it is sufficient to specify whether an entity is of real, integer, boolean or string type. When modelling a particular application domain, however, more expert information is available, and it would be very helpful to the modeller if it could be stored (represented) in the model base. For example, information can be available about upper and lower bounds of variables and parameters (*e.g.* stating that concentration, through the definition of its interval, is always positive). Also information about the causality of a quantity (input or output) can be included, since this information is of importance when developing a-causal models. As can be seen in the PhysicalQuantityType structure, this information can easily be integrated in MSL-USER. Once represented in a model, the model parser can make use of it to determine the legitimacy of the model (*e.g.* checking if the dimensions of parameters that are coupled match) and to generate efficient code (*e.g.* by means of constraint propagation based on lower and upper bound information). The constraints integrated in MSL-USER are transferred to the symbolic part of the MSL-EXEC representation and is used to protect the user for constraint violations during simulation or user input.

Basic quantities

Using the methodology introduced earlier, the PhysicalQuantityType structure can be specialised as classes for specific quantities. Here, alike the class ‘Concentration’, the physical quantity ‘Area’ is defined:

```
CLASS Area
"A class for area"
SPECIALISES PhysicalQuantityType :=
{:
  quantity <- "Area";
  unit <- "m^2";
  interval <- {: lowerBound <- 0; upperBound <- PLUS_INF:};
:};
```

Definitions of physical quantity types are used to instantiate objects of those types. The ISO 1000 standard also defines physical constants such as the universal gravity constant whose MSL-USER description is given as an object declaration below:

```
OBJ UniversalGravityConstant
"Universal gravity constant" : PhysicalQuantityType :=
{:
  quantity  <- "G";
  unit      <- "m^3/(g*s^2)";
  value     <- 6.67259E-11;
:};
```

It should be noted here that in the WEST environment, the units are not only used for dimensional checking during model compilation, but are also passed on to the experimentation environment where the user is presented with variable names, descriptions, values as well as their units. This way, a variable or parameter description, a default value and an interval that have been defined by the expert developing the model, is available for the user. In this way, the user is protected against erroneous parameter values and is warned when a variable evolves out of its boundaries during a simulation run.

Quantities typical for WWTP's

Simulation of wastewater treatment system behaviour, incorporating phenomena such as carbon oxidation, nitrification, denitrification and phosphorus removal, must necessarily account for a large number of reactions between a large number of components (Henze *et al.*, 2000). Several Activated Sludge Models (ASM 1, 2, 2d and 3) have been developed by the task group on mathematical modelling of the International Water Association (IWA). As will be described in the sequel, each of the variables in these models, denoting a component of the wastewater, indexes a column in the model stoichiometry matrix. In MSL-USER, the components of *e.g.* ASM1 are easily described as an enumerated type:

```
TYPE Components = ENUM {H_2O, S_S, S_O, S_NO, S_ND, S_NH, S_ALK, X_I, X_S, X_BH, X_BA, X_P, X_ND};
```

Thus, the modeller refers to the components by their name, while, where necessary, the corresponding integer index is used. Though WEST's simulator uses the numerical values of the Components indexes to address matrix elements, the experimentation environment presents the symbolic name of the index to the user. This reverse mapping is performed by the model compiler when generating MSL-EXEC code. Note how H_2O is explicitly modelled as a component.

Other quantities typical for WWTP modelling are stoichiometric and kinetic parameters. Kinetic parameters characterise the rate of reaction of the conversions in the model (*e.g.* maximal specific growth rate, decay rate, …); stoichiometric parameters indicate the stoichiometric relations between the different components in the model (*e.g.* yield coefficient, …). In MSL, these parameters can easily be declared as objects of a certain, more general, class specification:

```
CLASS Yield
"A class for Yield"
SPECIALISES PhysicalQuantityType :=
{:
  quantity <- "Yield";
  unit <- "-";
  interval <- {: lowerBound <- 0; upperBound <- 1:};
:};

CLASS GrowthRate
"GrowthRate"
SPECIALISES PhysicalQuantityType :=
{:
  quantity <- "GrowthRate";
  unit <- "1/d";
  interval <- {: lowerBound <- 0; upperBound <- 20:};
:};

CLASS SaturationCoefficient
"Saturation coefficient"
SPECIALISES PhysicalQuantityType :=
{:
  quantity <- "K";
  unit <- "-";
  interval <- {: lowerBound <- 0; upperBound <- 100:};
:};

OBJ Y      "Yield For Heterotrophic Biomass"
  : Yield := {:value <- 0.67:};
OBJ mu     "Maximum Specific Growth Rate For Heterotrophic Biomass"
  : GrowthRate := {:value <- 4.00:};
OBJ K_S      "Half-velocity Constant For Heterotrophic Biomass"
  : SaturationCoefficient :={:value <- 20.00:};
```

Transferred input-output quantities: terminals

The ultimate goal is to build complex models by connecting more primitive sub-models or blocks, possibly built up of coupled models themselves. In the case of WWTP models, the sub-model types mostly correspond in a 1-to-1 relationship to physical entities such as aeration tanks, clarifiers, pumps, splitters and mixing tanks. This ensures *structural validity* of the assembled models. Note how the building blocks need not match physical objects directly but may rather correspond to abstract concepts such as processes.

To connect sub-models, these sub-models require connection *ports* or *terminals*. This implies that interaction between the sub-models is assumed to *only* occur through the connections made between their terminals. When parsing a coupled model, the connections are replaced by appropriate algebraic equalities.

In our WWTP models, different terminal types are used. DataTerminals represent information to be used in sensor and controller blocks. However, the main terminal type is the WWTPTerminal. In the basic model base discussed here, only flux of biochemical material is considered. Heat flow for example is not considered. This is one of the modelling assumptions mentioned in the discussion of the ASM1 model and is obvious from the WWTPTerminal definition.

The WWTPTerminal is a vector of mass fluxes for each of the components taken into consideration in the model. The size of the vector is given by the number of identifiers (the cardinality) in the enumerated type ‘Components’ and hence depends entirely on how many components the user includes in this type. Note how

the actual Component declaration may be given after all other declarations. MSL-USER interprets the equations and declarations in a model as a set rather than as a sequence of statements. Basically, this means the order in which the declarations or equations are included in the model base is of no importance. This evidently facilitates model base development and may enhance clarity.

```
OBJ NrOfComponents
"
 The number of biological components considered in the WWTP models
"
: Integer := Cardinality(Components);

CLASS WWTPTerminal
"
 The variables which are passed between WWTP model building blocks
"
= MassFlux[NrOfComponents;];
```

While the model compiler will check whether (type-)compatible terminals are connected and how many connections are allowed to/from a terminal, the graphical modelling environment will already perform a check during interactive modelling. Normally the same terminals for biochemical transport are used everywhere in a configuration. If other terminals need to be used (*e.g.* for modelling a river system), explicit conversion blocks converting the elements of the different component vectors need to be foreseen. Direct coupling of a river compartment model, using another set of components, to a wastewater treatment model is not possible.

Building a model class hierarchy starting from general laws

Introduction to the general mass conservation law

The choice to transfer mass fluxes via the terminals instead of the mostly used concentrations and flow rates has different reasons. In processes where next to water or a water suspension also gasses and carrier materials may be transferred from one unit to another, only the concentration in the water phase is measured in reality. Denoting the concentration in units of $M.L^{-3}$, the factor L^{-3} indicates only the water or the suspension and not the entire transferred volume (including gas and carrier material). This can easily be the source of errors during the model development. Also the easy formulation of mass conservation when masses rather than concentrations are used is an advantage of this choice. The mass conservation law can easily be formulated as dM/dt. This conservation can be calculated for the different components *i* of the WWTPTerminal, so that elemental balances for carbon and nitrogen are easily derived. The user should however still have the possibility to interact with the model through output variables like concentration and flow rather than mass fluxes. For example, a mass balance of an ideally stirred tank reactor (CSTR) with volume V (L^3), components *i*, and terminals α, can be written as:

$$C_i = \frac{M_i}{V} \tag{1}$$

$$\frac{dM_i}{dt} = \sum_{\alpha} \Phi_{i_\alpha} + R_i V \tag{2}$$

$$\frac{dV}{dt} = \sum_{i} \left(\frac{1}{\rho_i} \sum_{\alpha} \Phi_{i_\alpha} \right) \tag{3}$$

In the case of an aeration tank with components dissolved at a low concentration in the water phase, the following simplifying assumption can be made:

$$\forall i \neq H_2O : \rho_i = \infty \tag{4}$$

stating that it is assumed that only water occupies a finite space. In case the density of the suspensions is different from 1 kg/l or 10^6 g/m^3 (= ρ_{H2O}), this assumption will no longer suffice. In that case the density of the individual components needs to be known.

In case for example heat transport should be modelled, the same assumptions will be used, *i.e.* heat flux will be transferred at the terminals.

Modelling biochemical conversion: the Petersen matrix

Introduction

Crucial in modelling the biochemical conversions in a wastewater treatment plant is to realistically model the inter-component biochemical reactions. These reactions must be representative of the most important fundamental processes occurring within the system. Furthermore, the model should quantify both the *kinetics* (rate-concentration dependence) and the *stoichiometry* (relationship that one component has to another in a reaction) of each process. Identification of the major processes and selection of the appropriate kinetic and stoichiometric expressions for each are the major conceptual tasks during development of a mathematical conversion model.

The IWA task group mentioned above (Henze *et al.*, 1987) chose the *matrix format* introduced by Petersen (1965) for the presentation of its models. The first step in setting up this matrix is to identify the *components* of relevance in the model. The second step in developing the matrix is to identify the biological *processes* occurring in the system; *i.e.* the conversions or transformations which affect the components listed.

A simple example

Consider the situation in which heterotrophic bacteria are growing in an aerobic environment by utilizing a soluble substrate for carbon and energy. In one simple conceptualisation of this situation, two fundamental processes occur: the biomass increases by cell growth and decreases by decay. Other activities, such as oxygen utilization and substrate removal, also occur, but these are not considered to be fundamental because they are the result of biomass growth and decay and are coupled to them through the system stoichiometry. The simplest model of this situation must consider the concentrations of three components: biomass, substrate and dissolved oxygen. The matrix incorporating the fate of these three components in the two fundamental processes is shown in Table 1.

Table 1. Process stoichiometry and kinetics for heterotrophic growth in an aerobic environment

Process j	Component i			Process Rate r_j ($ML^{-3}T^{-1}$)
	1. Biomass X_B	2. Substrate S_S	3. Oxygen S_O	
1. Growth	1	$-\frac{1}{Y}$	$-\frac{1-Y}{Y}$	$\frac{\mu S_S}{K_S + S_S} \cdot X_B$
2. Decay	-1		-1	$b.X_B$
Stoichiometric Parameters: Growth yield Y	$M(COD).L^{-3}$	$M(COD).L^{-3}$	$M(-COD).L^{-3}$	Kinetic Parameters: Maximum specific growth rate μ Half-velocity constant K_S Specific decay rate b

As mentioned in the introduction, the first step in setting up the matrix is to identify the *components* of relevance in the model. In this scenario these are biomass, substrate and dissolved oxygen, which are listed, with units, as columns in Table 1. In conformity with IWA nomenclature (Grau *et al.*, 1982), particulate constituents are given the symbol *X* and the soluble components *S.* Subscripts are used to specify individual components: *B* for biomass, *S* for substrate and *O* for oxygen.

The second step in developing the matrix is to identify the *biological processes* occurring in the system; *i.e.* the conversions or transformations which affect the components listed. Only two processes are included in this example: aerobic growth of biomass and its loss by decay. These processes are listed in the leftmost column of the table. The *kinetic expressions* or *rate equations* for each process are recorded in the rightmost column of the table in the appropriate row. Process rates are denoted by r_j where *j* corresponds to the process index.

If we were to use the simple Monod-Herbert model for this situation, the rate expressions would be those in Table 1. The Monod equation, r_1, states that growth of biomass is proportional to biomass concentration in a first order manner and to substrate concentration in a mixed order manner. The expression r_2 states that biomass decay is first order with respect to biomass concentration.

The elements within the table comprise the stoichiometric coefficients, ν_{ij}, which set out the mass relationships between the components in the individual processes. For example, growth of biomass ($+1$) occurs at the expense of soluble substrate ($-\frac{1}{Y}$, Y is the yield parameter); oxygen is utilized in the metabolic process ($-\frac{1-Y}{Y}$). The coefficients ν_{ij} can easily be deduced by working in consistent units. In this case, all organic constituents have been expressed as equivalent amounts of chemical oxygen demand (COD); likewise, oxygen is expressed as negative oxygen demand. The sign convention used in the table is negative for consumption and positive for production.

In matrix form, we obtain a stoichiometry matrix

$$\nu = \begin{pmatrix} 1 & -\frac{1}{Y} & -\frac{1-Y}{Y} \\ -1 & 0 & -1 \end{pmatrix}$$

and a kinetics vector

$$r = \begin{pmatrix} \frac{\mu S_S}{K_S + S_S} \cdot X_B \\ b \cdot X_B \end{pmatrix}$$

Within a system, the concentration of a single component may be affected by a number of different processes. An important benefit of the matrix representation is that it allows rapid and easy recognition of the fate of each component, which aids in the preparation of mass balance equations. This may be seen by moving down the column representing a component.

As mentioned before, the basic equation for a mass balance within any defined system boundary is equation 2. The flux terms are transport terms and depend upon the physical characteristics of the system being modelled. The system reaction term, R_i, is obtained by summing the products of the stoichiometric coefficients ν_{ij} and the process rate expression r_j for the component i being considered in the mass balance (*i.e.* the sum over a column):

$$R_i = \sum_i \nu_{ij} r_j \tag{5}$$

For example, the rate of reaction, R, for oxygen, S_O, at a point in the system would be:

$$R_{S_O} = -\frac{1-Y}{Y} \frac{\mu S_S}{K_S + S_S} X_B - bX_B \tag{6}$$

To create the mass balance for each component within a given system boundary (*e.g.* an ideally mixed reactor), the conversion rate would be combined with the appropriate transport terms for the particular system. For instance in an ideally mixed tank reactor with one input, a constant volume V and an influent flow rate Q, the following mass balance would emerge for S_O.

$$V \frac{dS_O}{dt} = \frac{dM_{S_O}}{dt} = \sum_\alpha \Phi_{S_{O,\alpha}} + V \cdot R_{S_O} = Q \cdot S_{O,in} - Q \cdot S_O + V \cdot R_{S_O} \tag{7}$$

Another benefit of the Petersen matrix is that continuity may be checked per process by horizontally moving across the matrix. This can only be done provided consistent units have been used, because then the sum of the stoichiometric coefficients must be zero. This can be demonstrated by considering the decay process. Recalling that oxygen is negative COD so that its coefficient must be multiplied by -1, all COD lost from the biomass through decay must be balanced by oxygen utilization. Similarly, for the growth process, the

substrate COD lost from solution due to growth minus the amount converted into new cells must equal the oxygen used for cell synthesis.

Inheritance hierarchy

Using the general mass conservation law introduced above, models must be constructed for each type of building block. This is achieved in the form of a class *inheritance* hierarchy. Hereby, maximum *re-use* and *clarity* is achieved. Clarity is a direct result of the relationship between the inheritance hierarchy on the one hand and the different levels of specificity of the models on the other hand. In the generic model base, GenericModelType is defined:

```
TYPE GenericModelType
=
RECORD
{
 comments   : String;
 interface  : SET_OF (InterfaceDeclarationType);
 parameters : SET_OF (ParameterDeclarationType);
};
```

It shows how any model has a description (*comments*) part, an *interface* set and a *parameter* set. The interface set describes which terminals serve as an input to the model and which variables are transferred to a subsequent model via an output terminal. The parameters of the model are a set of invariant values that are given a value at the beginning of a simulation.

For basic models in the DAE formalism, PhysicalDAEModelType prescribes the structure:

```
TYPE PhysicalDAEModelType
EXTENDS GenericModelType WITH
RECORD
{
 independent : SET_OF (ObjectDeclarationType);
 state       : SET_OF (PhysicalQuantityType);
 initial     : SET_OF (EquationType);
 equations   : SET_OF (EquationType);
 terminal    : SET_OF (EquationType);
};
```

Time is mostly used as the *independent* variable. In the case of PDE modelling, multiple independent variables can be defined. Dependent (both algebraic and derived) *state* variables are defined in the *state* section. The *initial* section contains algebraic equations that will be solved only once during simulation. The result of this initial calculations can for example be used to define the initial values of derived state variables used in the *equations* section. This section contains the algebraic equations and ODE's that define the model. Equations in the *terminal* section are only calculated once at the end of the simulation run. The GenericModelType can also be extended to describe the essence of coupled models:

```
TYPE CoupledModelType
EXTENDS GenericModelType WITH
RECORD
{
 sub_models : SET_OF (ModelDeclarationType);
 coupling   : SET_OF (CouplingStatementType);
};
```

In a *coupled model*, the *sub_models* section enumerates the set of models to be coupled. In the *coupling* section, statements are included that describe how to couple these models. This can be done using two statements. The *connect* statement is used to connect the interface variables of the coupled model to the interface of one of the sub-models or to connect the interfaces of two sub-models. The *control* statement is to indicate that a parameter of a sub-model is controlled by an interface variable of a second model. It is important to note that the MSL-USER parser will then automatically transform the controlled parameter into a new interface variable, since this model component will no longer be time-invariant and therefore, by definition, becomes a variable.

Both CoupledModelType and DAEModelType are extensions of GenericModelType which means they inherit its structure (and add to it). The resulting top-level inheritance hierarchy is given in Figure 2.

Figure 2: Top level inheritance hierarchy in the WEST model base

In the WWTP model base hierarchy, some of the model classes are derived directly from PhysicalDAEModelType (Figure 3). The ones listed directly below are models of the settler. The Takács model, for instance, is a discretised (10-layer) model of the settling process. It should be noted that the dedicated WEST-PDE parser is able to automatically discretise a class of PDE models of, for instance, the settling process using orthogonal collocation (Indrani and Vangheluwe, 1998). Once discretised, these models are of the ordinary PhysicalDAEModelType and are fitting in the hierarchy of Figure 3.

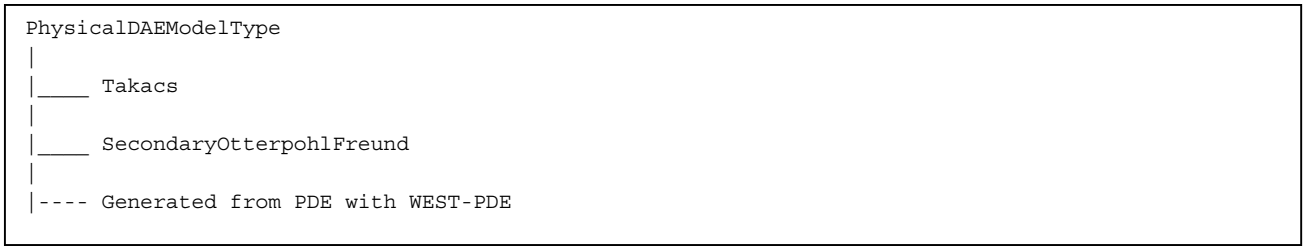

Figure 3: Settler models directly derived from PhysicalDAEModelType

Sensor, controller, data filter and transformer models are also derived from PhysicalDAEModelType (Figure 4). These models do not describe physical processes involving (transport of) matter and energy and hence do not adhere to physical laws. Though not subject to physical constraints, they do deal with the values of physical variables.

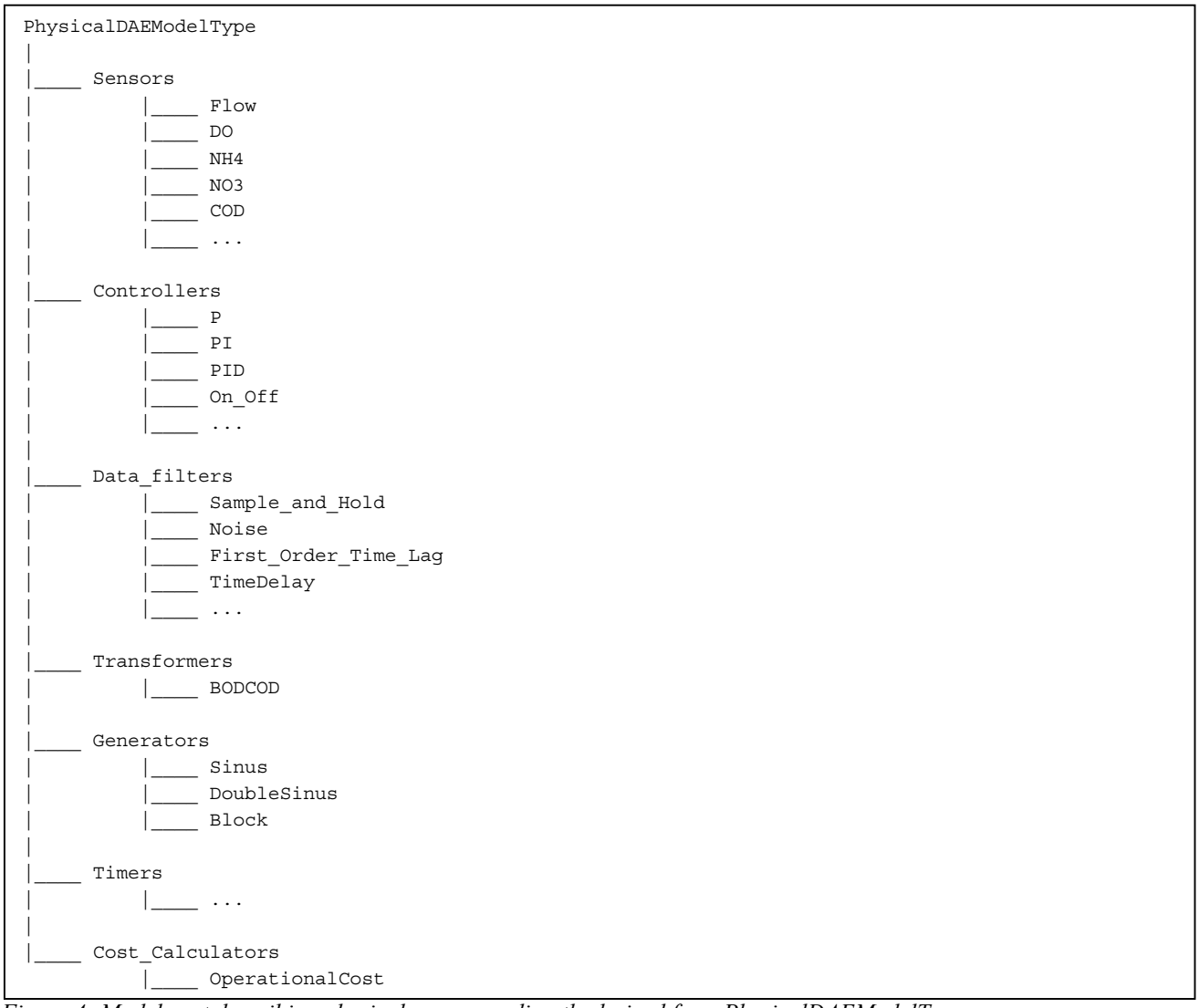

Figure 4: Models not describing physical processes directly derived from PhysicalDAEModelType

As mentioned before, WEST is not only used to model wastewater treatment processes but also parts of the environment, in particular the river in which the treated effluent is discharged (Figure 5).

Figure 5: Some river models developed in WEST

The shallowness of the above inheritance hierarchy reflects the diverse nature of this subset of model types used in wastewater engineering, not allowing for much re-use.

Now we will look into the development of WWTPAtomicModel, derived using the mass conservation law, from which many other model types are derived. This will illustrate the powerful re-use capabilities of the developed system. First of all, note that the matrix of the simple example could be implemented in MSL-USER in the following easy way:

```
TYPE Components = ENUM {H_2O, S_S, S_O, X_B};

TYPE Reactions = ENUM {Growth, Decay};

parameters <-
  {
   OBJ Y    "Yield"  : Yield := {:value <- 0.67:};
   OBJ mu   "Maximum Specific Growth Rate" : GrowthRate := {:value <- 4.00:};
   OBJ K_S  "SaturationCoeff" : SaturationCoefficient :={:value <- 20.00:};
   OBJ b    "Decay Rate" : DecayRate := {:value <- 0.40:};
  };

initial <-
  {
   parameters.Stoichiometry[Growth][X_B] := 1;
   parameters.Stoichiometry[Growth][S_S] := - 1/(parameters.Y);
   parameters.Stoichiometry[Growth][S_O] := - (1 - parameters.Y)/parameters.Y;
   parameters.Stoichiometry[Decay][X_BH] := - 1;
   parameters.Stoichiometry[Decay][S_O]  := - 1;
  };

equations <-
  {
   state.Kinetics[Growth] := parameters.mu *
       (state.C[S_S]/(parameters.K_S+state.C[S_S])) * state.C[X_B];
   state.Kinetics[DecayOfHetero] := parameters.b*state.C[X_B];
  };
```

The basic mass balance equation 2 for each of the components can also be rewritten in MSL format. First, the flux for each component *i* is calculated as $\sum_{\alpha} \Phi_{i_\alpha}$.

```
{FOREACH Comp_Index IN {1 .. NrOfComponents}:
state.FluxPerComponent[Comp_Index] =
   (SUMOVER In_Terminal IN {SelectByType(interface,InWWTPTerminal)}:
    In_Terminal[Comp_Index])+
   (SUMOVER Out_Terminal IN {SelectByType(interface,OutWWTPTerminal)}:
    Out_Terminal[Comp_Index]);};
```

Next, the reaction (conversion) $R_i V = V \cdot \sum_i \nu_{ij} r_u$ is encoded in a straightforward manner as:

```
{FOREACH Comp_Index IN {1 .. NrOfComponents}:
  state.ConversionTermPerComponent[Comp_Index] =
   SUMOVER Reaction_Index IN {1 .. NrOfReactions}:
    (parameters.Stoichiometry[Reaction_Index][Comp_Index]
    *state.Kinetics[Reaction_Index])
    *state.V;};
```

Finally , the complete mass balance $\frac{dM_i}{dt} = \sum_{\alpha} \Phi_{i_\alpha} + R_{iV}$ is written for each component:

```
{FOREACH Comp_Index IN {1 .. NrOfComponents}:
  DERIV(state.M[Comp_Index],[independent.t]) =
   state.FluxPerComponent[Comp_Index]
   + state.ConversionTermPerComponent[Comp_Index];};
```

The rate of change of a component's mass thus consists of the net result of incoming and outgoing mass flux augmented with a reaction term due to biochemical interactions between different components. The MSL-USER compiler will expand the above few lines into the appropriate equations based on the matrix given. These equations will subsequently be manipulated to generate correct and efficient simulation code. Note that components which are transported but do not react (*i.e.* only hydraulics, no physico-chemical nor biological processes) have a column of zeroes in the stoichiometry matrix. In MSL-USER, by default, when a variable or a parameter is not given a value, the initial value is 0. Thus, if we don't assign anything to elements of the stoichiometry matrix, it is a matrix of zeroes, which means no biochemical reactions take place.

Note how the use of this matrix representation is not limited to this simple example or even to the ASM1 model. Also the models ASM2, ASM2d, ASM3 and RWQM1 developed by IWA task groups in the mean time have been implemented (Henze *et al.*, 2000; Reichert *et al.*, 2001). The user can also easily implement mass balance models himself using this general approach. Only the component vector, the reaction vector and the stoichiometric and kinetic coefficients need to be specified.

Logically, the next level (below WWTPAtomicModel) of classification would be to distinguish between models without volume (point-model abstractions where no mass is accumulated and hence no reactions occur) and models with volume. For models with volume, the distinction must be made between models where volume is considered constant and those where volume may vary. This class hierarchy is depicted in Figure 6.

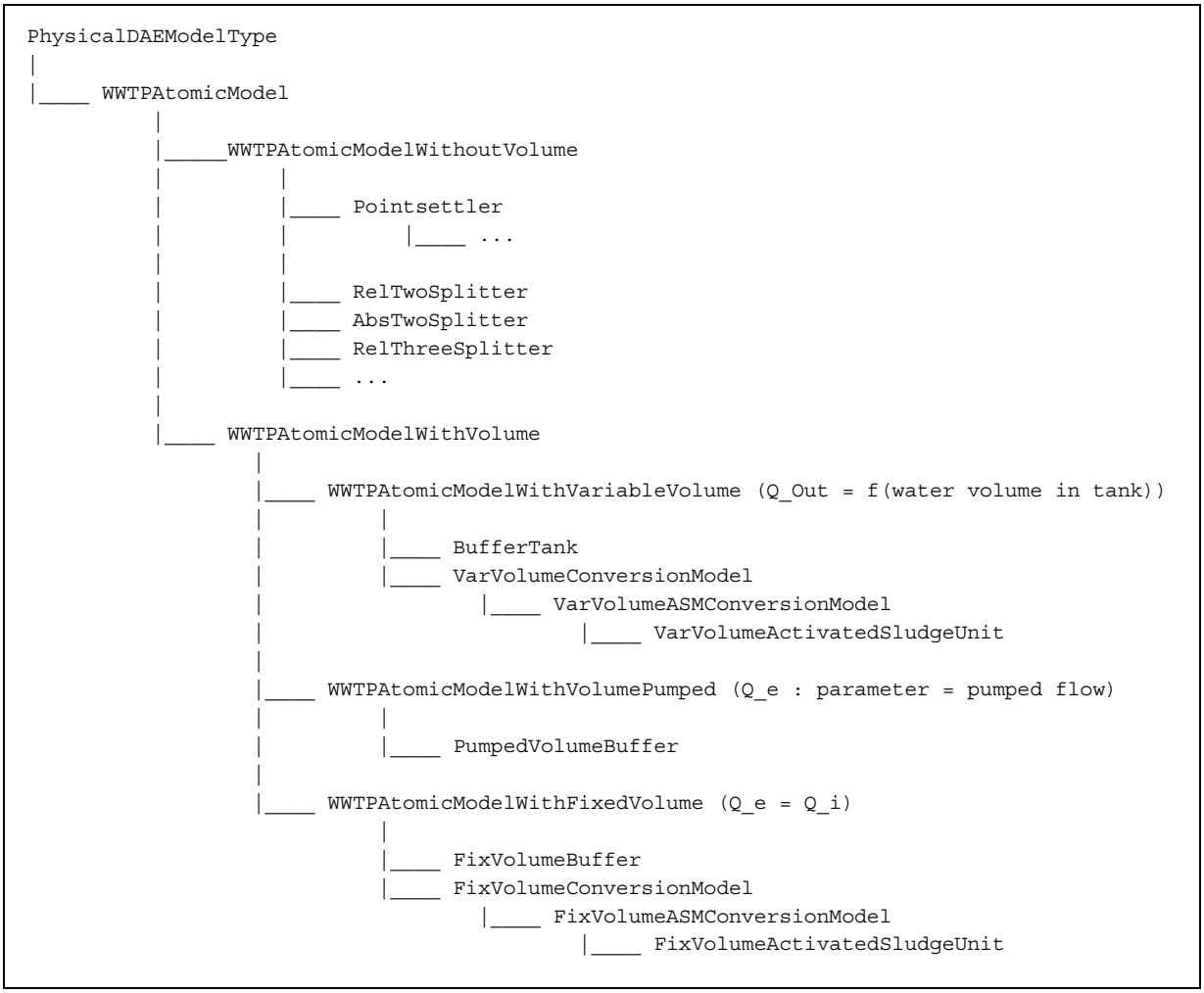

Figure 6: Class hierarchy of models without and with volume derived from WWTPAtomicModel

The modelling environment: Building a graphical configuration and a coupled model

As mentioned above, the WEST modelling environment allows for graphical component-based modelling. A hierarchical graphical editor (HGE) was especially designed for the interactive building of complex configurations from basic building blocks. The user can entirely rebuild the physical configuration of the wastewater treatment plant in the HGE (Figure 7). Each of the components (aeration basins, clarifiers, …) are symbolically represented by an icon with one or more input and outputs (*terminals*). The program uses two types of terminals: data terminals and physical terminals. Physical terminals represent a physical connection between two components in the configuration. Data terminals on the contrary, represent a dataflow in the system. This can be a measurement signal from a sensor to a control system, or a calculated control action from the control system to the manipulated variable in the configuration.

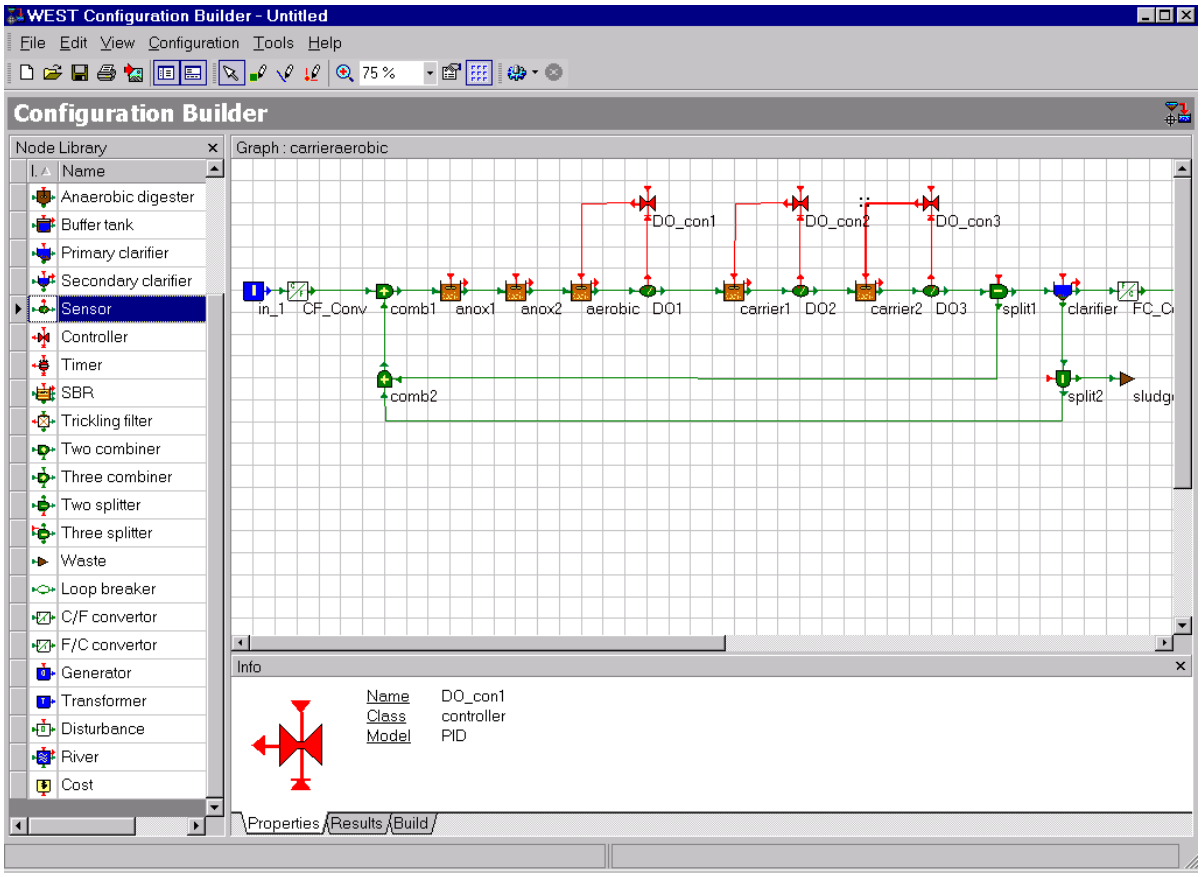

Figure 7: Representation of a WWTP model in the HGE (Hierarchical Graphical Editor)

At this point, only a graphical representation has been made of the wastewater treatment plant to be modelled. Nothing has been specified on its behaviour. In a next step, once the configuration has been built graphically, each component of this configuration should be linked to a model from the model base. Each of these models is a structured collection of DAE's representing the time dependent behaviour of the components in question. The complete set of models together with the parameter values chosen by the users then specifies the dynamic behaviour of the model. A model base may contain multiple reasonable candidate models based on model features and user requirements. WEST leaves the final choice to the user, so model selection is mostly done on a manual basis. However, ongoing research tries to find and validate methodologies to accomplish automatic model selection based on measurements performed on the real process (Vanrolleghem and Van Daele, 1994; Cooney and McDonalds, 1995; Takors *et al.*, 1997; Dochain and Vanrolleghem, 2001).

Now from this graphical specification, together with the models chosen from the model base, a coupled model is produced. Some of the MSL-USER code corresponding to the coupled model represented in Figure 7 is shown below.

```
CLASS SuspendedCarrierWWTPClass SPECIALISES CoupledModelType :=
{:
 interface <-
 {
  OBJ In_1 (* terminal = "In1" *) "InfluentConc" : InWWTPConcTerminal := {:causality <- CIN:},
  OBJ Out_1 (* terminal = "Out1" *) "EfflunetConc" : OutWWTPConcTerminal := {:causality <- COUT:},
 };

 parameters <-
 {
  OBJ Y_A "Autotrophic Yield" : YieldForAutotrophicBiomass := {: value <- 0.24 :},
  OBJ Y_H "Heterotrophic Yield" : YieldForHeterotrophicBiomass := {: value <- 0.67 :},
  ...
 };

 sub_models <-
 {
  OBJ CF_Conv : CtoF,
  OBJ comb1 : TwoCombiner,
  OBJ anox1 : SuspendedCarrierASU,
  ...
  OBJ aerobic : FixVolumeASU,
  OBJ DO1 : DO,
  OBJ DO_con1 : SaturationPI,
  ...
 };

 coupling <-
 {
  // parameter coupling
  ...
  sub_models.anox1.parameters.Y_A.value := parameters.Y_A.value,
  sub_models.anox1.parameters.Y_H.value := parameters.Y_H.value,
  ...
  sub_models.aerobic.parameters.Y_A.value := parameters.Y_A.value,
  sub_models.aerobic.parameters.Y_H.value := parameters.Y_H.value,
  ...
  // sub-model coupling
  connect(interface.In_1, sub_models.CF_Conv.interface.Inflow),
  connect(sub_models.CF_Conv.interface.Outflow, sub_models.comb1.interface.Inflow1),
  ...
  // control statements
  control(sub_models.DO_con1.interface.u, sub_models.aerobic.parameters.Kla),
  ...
 };
:};

OBJ SuspendedCarrier "": SuspendedCarrierWWTPClass;
```

As indicated earlier, each icon put on the canvas results in the instantiation of an MSL-USER object of the appropriate class in the coupled models *sub_models* section. If the user decides to define parameters of the coupled model in the HGE, they are stated in the *parameters* section. In the *coupling* section, statements are included that describe how the sub-models are connected to each other. First of all, the relations between the parameters of the sub-models and the user-defined parameters of the coupled model are indicated. Following this, the *connect* and *control* statements are listed. The *connect* statement is used to connect the interface

variables of the coupled model to the interface of one of the sub-models or to connect the interfaces of two sub-models. The *control* statement is used to indicate that a parameter of a sub-model is controlled by an interface variable of a second model. However, parameters are invariant values to be declared at the beginning of a simulation run. In case a controller is used, the parameter serves as a manipulated variable. Therefore, it will automatically be transformed from a parameter to an input variable by the MLS-USER parser.

The graphical editor and the coupled model introduce a second level of hierarchy in WEST. Indeed, next to the hierarchical structure of the model base, aimed at maximal re-use of knowledge, also coupled models and their graphical representations can be re-used. All coupled model have an interface completely alike the sub-models from which they are composed. Consequently, the user can decide to add a coupled model to the model base and re-use it in yet another coupled model. This way, a model can be structured as a tree of coupled models and atomic models from the original model base. Again a maximal level of re-usability and transfer of knowledge is obtained here. When coupling the models of the sewer system, treatment plant and receiving water, one can build and test the models separately. Afterwards, they can easily be linked in the graphical editor by re-using the models created before (Figure 8). When creating large models, it is very useful to first test the submodels and only afterwards connect them to create the integrated large model.

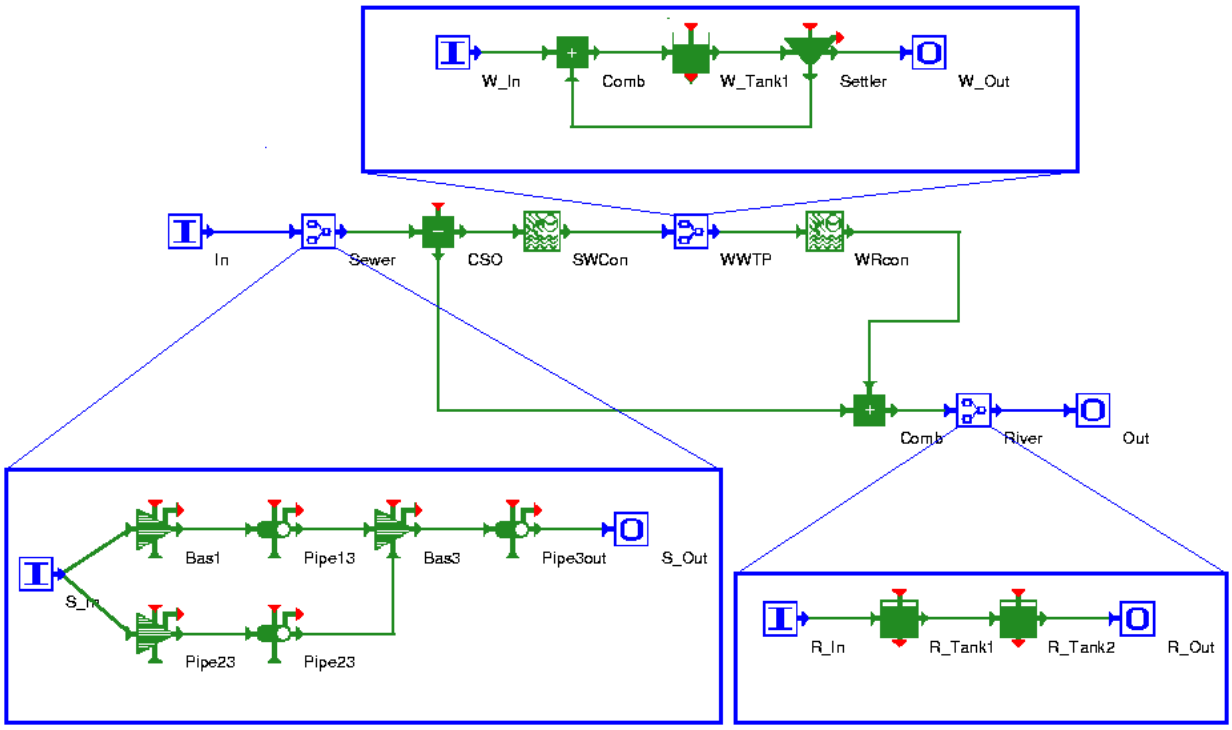

Figure 8: Representation of the re-use of models in the graphical editor of WEST

Parsing from MSL-USER to MSL-EXEC

After constructing a coupled model in the HGE, the parser generates MSL-EXEC from this model for use in the experimentation environment of WEST. It therefore uses the coupled model itself along with the models stored in the model base.

During this (parsing) process, the syntax and the semantics of the MSL-USER representation are checked automatically as well as the compatibility of the nature (the units) of the variables passed on between the different sub-models. This way, some model coding errors may be detected here and not only when simulating the model.

MSL-EXEC contains both code to describe dynamics as code to represent the symbolic information ("knowledge"). The model dynamics are specified as a set of ordinary differential equations (ODE's) and algebraic equations. As the order of the equations is of no importance in MSL-USER, the correct sorting of the differential and algebraic equation has to be done by the parser. The different built-in statements, are recognised by the parser and translated into their equivalent C++ formulations.

The symbolic information is used to display the model information in the WEST experimentation environment. For example, based on the annotations *hidden* and *fixed*, a variable or parameter will not be shown in the experimentation environment or the user will not be able to change its value. As mentioned before, a controlled parameter will automatically be transformed from a parameter to an input variable by the MLS-USER parser and will therefore no longer be visible in the parameter listing. Also the constraints on variables as integrated in the MSL-USER model base are transferred to the symbolic part of the MSL-EXEC representation and are used to protect the user from constraint violations during simulation or user input. Furthermore default values, units and descriptions are visible in the experimentation environment.

Before the MSL-EXEC code can be used in the experimentation environment, an extra compilation step has to be performed. In this compilation step, a library file (executable code) is generated that can be loaded into the experimentation environment. This compilation step guarantees code that is optimised for simulation performance and accuracy. During this compilation, standard C libraries are linked to the generated model, enabling the user to include all functions available in these libraries in the MSL-USER models. Even user-defined C++ functions can be used and linked during parsing.

During parsing symbolic manipulation can be performed too. Symbolic manipulation is concerned with finding symbolic or exact solutions to mathematical problems. This avoids rounding errors and the need for an error analysis. Exact or symbolic computation has the disadvantage of being more compute-intensive than numerical calculation. However, as symbolic manipulation is performed only once as opposed to numerical code, which gets executed time and again during simulation, the one-time intensive symbolic computation cost at parse time is largely compensated by the performance gain at simulation time.

The following very simple example will illustrate the usefulness of symbolic manipulation. Imagine you are trying to solve an equation for an unknown variable, such as: $x - 5 = 0$. An analytical solution is found if the equation can be solved explicitly for the unknown variable. In this case this is easy to see. However, we might develop an *algorithm* on a computer to solve this equation numerically. The algorithm would test various values for x, and then stop with a solution when the equation is satisfied to some chosen tolerance. For example, we might demand that the computer should solve this equation to an accuracy of 0.5. Then the computer would follow the algorithm until it found a solution. Given an initial guess $x = 1$, depending on the algorithm it might come up with the following guesses: $x = 2.2$, $x = 3.3$, $x = 4.6$, and return the solution $x = 4.6$. Note that if we want to be more accurate, it takes more time to solve this equation to this level of accuracy. If, through symbolic manipulation we were able to find the exact analytical solution $x = 5$ immediately, an enormous gain at simulation time would result.

When the equations to be solved are large and complex, one has to deal with some issues about how to reach the solution in the most efficient way. Several problems can be tackled both in a numerical and a symbolic way. Getting the solution using one method rather than the other, has advantages and disadvantages. The advantages of symbolic manipulation in the case of WEST are:

- Performance, if you know a quantity analytically, you can avoid some computations and decrease the computation time.
- More accurate numerical results, because by pre-processing data with symbolic manipulations, more advanced numerical techniques can be exploited.

On the other hand, analytical solution methods do not exist for quite some problems. However, symbolic methods can still be used to derive expressions necessary for performing numerical computations – such as gradients and Jacobian and Hessian matrices. Thus, the traditional roles of numerical and symbolic computations are not distinct and many benefits arise from merging the two.

Working with the model: the experimentation environment

The experimentation environment depicted in Figure 9 enables the user to perform experiments on compiled models. As such, it is the interface between the user and the "simulator". During simulation, the solver communicates efficiently with the model dynamics part of the MSL-EXEC model. The simulator as a whole can be asked to perform a numerical simulation. In that case the solver is used to generate a state trajectory for the MSL-EXEC model. Different numerical solvers can be chosen interactively.

The experimentation environment also queries the simulator for symbolic information. This information will be retrieved from the symbolic information part of the MSL-EXEC model. Examples of such symbolic information are the model structure and the parameter listing in Figure 9. In this listing, the unit, a description and a default value of the parameter can be found together with its lower and upper bounds.

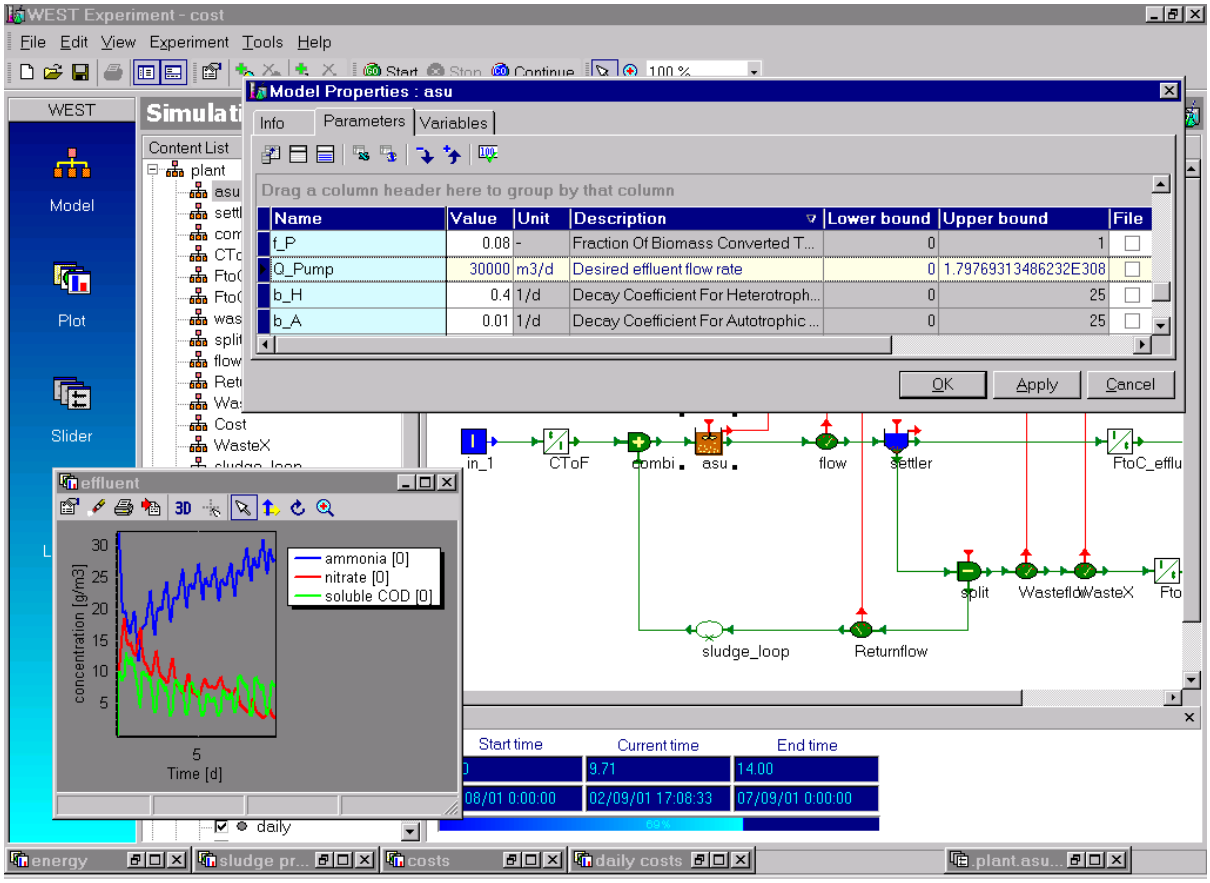

Figure 9: The WEST experimentation environment, showing a plot and a parameter listing

The following distinguishes between different *experiment types* as implemented in the WEST environment. The user thinks in terms of different *virtual experiments* with the model of a system. The following experiment types are currently implemented in WEST:

1. Simulation experiment

   Currently, there are two types of simulation experiments:

   - Initial value problem: state variable values are given at time t_{ini}. The simulator calculates the trajectory over $[t_{ini},t_{fin}]$. This is implemented using a set of forward integrators the user can select among.
   - Terminal value, end value or *shooting* problems: state variable values are given at t_{fin}. The simulator calculates the trajectory over $[t_{ini},t_{fin}]$. Solving the shooting problem is implemented in WEST using an optimisation algorithm whereby the varied entities are the unknown initial conditions and the goal function is the sum of absolute or squared values of differences between simulated end-value and known/specified end-value.

   Sometimes it is necessary to "synchronise" with external data. This is for example the case when the input *u(t)* is given as a table of measurements, for instance the influent composition or a pump schedule. The integrator can determine its own integration times and when an input value is needed, interpolation is used. When the input is given as a continuous function (via an generator model), no interpolation is required.

2. Trajectory optimisation experiment
   Certain model parameters are varied by a number of search algorithms the user can select from to minimise the *distance* between a simulated trajectory and a given (measured or desired) trajectory. This is mostly done for (constrained) parameter estimation (model calibration), but it can also be used for controller tuning and process design optimisation. The distance measure is typically a sum of squares of differences between measured and simulated values though absolute values can also be used. The difference between measured and simulated values can be calculated at different points in time: as described above, the simulator can be forced to *synchronise* with external data or interpolation can be used. In general, the differences can be weighted to account for measurement accuracy and possible differences in the order of magnitude of the different values in the objective function. The optimisation experiment also provides confidence information (covariance matrix) about the quality of the parameter estimations. The covariance matrix is calculated with the method of Nelder and Mead (1964). The confidence information can then be used, for instance, to draw confidence ellipses or give parameter confidence bounds.

3. End value optimisation experiment
   Here the optimiser is used to vary where some parameters (possibly constrained) to extremise a goal function that only evaluates variable at t_{fin}, for instance total economic cost.

4. Sensitivity analysis experiment
   The sensitivity of the model with respect to model parameter variations can be investigated. The calculation of sensitivity functions is based on the finite difference method. This method calculates the difference between two experiments, a reference experiment and a perturbation experiment. The perturbation experiment is performed by perturbating a model parameter by a small factor (the perturbation factor). Dividing the difference in model outputs between these experiments by the parameter change results in the sensitivity function. To make sure the sensitivity functions are calculated properly ,a third experiment is performed, the control experiment. For this experiment the parameter perturbation factor is doubled. If the resulting sensitivity function is within an allowed error band it can be assumed that the nonlinearity of the model did not influence the calculations. The error between both sensitivity functions is calculated with different criteria such as the sum of squared errors, the largest absolute difference, etc.
   Sensitivity functions form the basis of optimal experimental design because they indicate where the measurements are most sensitive to the parameters. Moreover, the Fisher information matrix which is an important cornerstone of experimental design is calculated using sensitivity functions. This matrix is a measure for the information content of the simulated experiment.

5. Monte Carlo experiment
   The uncertainty of the model output due to input (parameter and variable) uncertainty can be calculated in a Monte Carlo experiment. For each model input that is considered to be a random variable, a probability distribution is specified out of a range of possible distributions (normal, log-normal, uniform, triangular, …). Random samples are taken for each of the input distributions, and the set of samples ('shot') is entered into the deterministic model. The model is then solved as it would be for any deterministic analysis. The model results are stored and the process is repeated until the specified number of model iterations is completed (Cullen and Frey, 1999). From all stored model results, statistical properties (mean and standard deviation) and histograms are produced. These can subsequently be used in decision making, *e.g.* risk analysis (Rousseau *et al.*, 2001)

The experimentation environment can also be controlled via scripting languages (Tcl scripting, Visual Basic scripting). Scripting enables the user to perform several scenarios in an automated way. It is possible to automatically perform a series of experiments using a predefined set of parameter values. Output and integrater options can be controlled interactively. Among others, the Monte Carlo simulation engine has been constructed using such relatively simple scripts.

Conclusions

The mathematical modelling of biological wastewater treatment plants can be used during the design and optimisation phase. WEST is a general modelling and simulation environment and can, together with the developed model base, be used for this task. The model base is written in MSL-USER in which symbolic information can be included in the code. In the graphical modelling environment, the physical layout of the plant can be rebuilt, and each building block can be linked to a specific model from the model base. The graphical information is then combined with the information in the model base to produce MSL-EXEC-code, which can be compiled with a C++-compiler to generate fast, executable code. In the experimentation environment, the user can design different experiments like simulations, optimisations. The main advantages of the use of this software are the following. First, the modelling and simulation environment are strictly separated since these have different objectives (i.e. flexibility and model re-use vs. accuracy and performance). The MSL-USER language is a high level language which is easy to learn and to use, while information about boundaries and units of parameters and variables can be implemented. Furthermore, an extensive model base for the modelling of WWTP's is available. The parser uses symbolic manipulation to create numerically efficient code. Finally, the experimentation environment can be easily used to perform different types of experiments with the models. The user can extend these experiments by scripting.

Acknowledgements

Henk Vanhooren and Jurgen Meirlaen are Research Assistants of the Fund for Scientific Research - Flanders (Belgium). The authors would like to give special thanks to the Fund for Scientific Research (G.0102.97) for the financial support. This research was furthermore financially supported by the Flemish Institute for the Promotion of Scientific-Technological Research in Industry (IWT).

References

Broenink, J.F. (1990) *Computer aided physical modeling ans simulation: a bond graph approach.* PhD Thesis, University of Twente, Enschede, The Netherlands.

Cooney, M.J. and McDonalds, K.A. (1995) Optimal dynamic experiments for bioreactor model discrimination. *Appl. Microbiol. Biotechnol.*, **43**, 826-837.

Cullen, A.C. and Frey, H.C. (1999) *Probabilistic techniques in exposure assessment. A handbook for dealing with variability and uncertainty in models and inputs.* Plenum, New York.

Dochain D. and Vanrolleghem P.A. (2001) *Modelling and Estimation in Wastewater Treatment Processes*. IWA Publishing, , London, UK. ISBN 1-900222-50-7 (in Press).

Grau, P., Sutton, P.M., Henze, M., Elmaleh, S., Grady, C.P., Gujer, W. and Koller, J. (1982) Recommended notation for use in the description of biological wastewater treatment processes. *Wat. Res.*, **16**, 1501-1505.

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

Henze, M., Gujer, W., Mino, T. and van Loosdrecht, M.C.M. (2000) Activated Sludge Models ASM1, ASM2, ASM2d and ASM3. Scientific and Technical Report No. 9., IWA Publishing, London, UK.

Indrani, A.V. and Vangheluwe, H.L. (1998) *WEST++, Transformation of a given PDE to a DAE using the orthogonal collocation method on finite elements.* Internal Report, BIOMATH, Ghent University, Ghent, Belgium.

Meirlaen, J., Huyghebaert, B., Sforzi, F., Benedetti, L. and Vanrolleghem, P.A. (2001) Fast, simultaneous simulation of the integrated urban wastewater system using mechanistic surrogate models. *Wat. Sci. Tech.*, **43**(7), 301-310.

Morgenroth, E., van Loosdrecht, M.C.M. and Wanner, O. (2000) Biofilm models for the practitioner. *Wat. Sci. Tech.*, **41**(4-5), 509-512.

Nelder J.A. and Mead R. (1964) A simplex method for function minimization. *Comp. J.*, **7**, 308-313.

Petersen B., Gernaey K., Henze M. and Vanrolleghem P.A. (2001) Evaluation of an ASM1 model calibration procedure on a municipal-industrial wastewater treatment plant. *J. Hydroinformatics* (in Press)

Petersen, E.E. (1965) *Chemical reaction analysis.* Prentice-Hall, Englewood Cliffs, New Jersey.

Reichert P., Borchardt D., Henze M., Rauch W., Shanahan P., and Somlyody L., Vanrolleghem P. (2001) River water quality model No. 1 (RWQM1). Scientific and Technical Report No. 12., IWA Publishing, London, UK.

Takors, R., Wiechert, W. and Weuster-Botz, D. (1997) Experimental design for the identification of macrokinetic models and model discrimination. *Biotechnol. Bioeng.*, **56**, 564-576.

Vangheluwe, H.L., Claeys, F. and Vansteenkiste, G.C. (1998) The WEST++ wastewater treatment plant modelling and simulation environment. In: *10th European Simulation Symposium*, (Eds: Bergiela, A. and Kerckhoffs, E.) Society for Computer Simulation (SCS), Nottingham, UK.

Vangheluwe, H.L. (2000). *Multi-Formalism Modelling and Simulation*. PhD Thesis, Faculty of Sciences. Ghent University. Ghent.

Vanrolleghem, P.A. and Van Daele, M. (1994) Optimal experimental design for structure characterization of biodegradation models: on-line implementation in a respirographic biosensor. *Wat. Sci. Tech.*, **30**(4), 243-253.