Distributed virtual experiments in water quality management

F. Claeys*, M. Chtepen, L. Benedetti*, B. Dhoedt** and P.A. Vanrolleghem***

*BIOMATH, Ghent University, Coupure Links 653, B-9000 Ghent, Belgium
(E-mail: *Filip.Claeys@biomath.ugent.be*; *Peter.Vanrolleghem@ugent.be*)

**INTEC, Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium
(E-mail: *Maria.Chtepen@ugent.be*; *Bart.Dhoedt@.ugent.be*)

Abstract Since the complexity of virtual experiments (VEs) and their underlying models is constantly increasing, computational performance of monolithic software solutions is rapidly becoming insufficient. Examples of VEs are probabilistic design, model calibration, optimal experimental design and scenario analysis. In order to tackle this computational bottleneck, a framework for the distributed execution of VEs on a potentially heterogeneous pool of work nodes has been implemented. This framework was named WDVE (WEST distributed virtual experimentation) and is built on top of technologies such as C++, XML and SOAP. It was designed for stability, expandability, performance, platform-independence and ease of use.

Complex VEs are most often composed of mutually independent sub-experiments, which can be run concurrently. With WDVE, a complex VE that is executed on a so-called Master machine will therefore attempt to execute its sub-experiments on Slave machines that have previously registered with the Master. The process of submitting requests for the execution of sub-experiments is transparent and involves the transfer of a description of the experiment to be executed, and the resources that are needed for the execution (i.e., model and input data). WDVE is in many ways similar to the Grid Computing paradigm, which is currently receiving widespread attention. However, WDVE is more geared towards application within the scope of water quality management.

Keywords Distributed virtual experiments; mathematical modelling; water quality management

Water Science & Technology **Vol 53 No 1 pp 297–305** © IWA Publishing 2006

Introduction

Computational burden of water quality management tools

In water quality management, interesting methodologies have recently emerged that suffer from heavy computational demand. These include probabilistic design, model calibration, optimal experimental design and scenario analysis, all of which are introduced below.

As a common practice, deterministic dynamic models are used to evaluate different design and renovation scenarios of urban drainage structures (trunk sewers, storm water retention tanks, waste water treatment plants, etc.). One of the remaining issues when dealing with these deterministic models is the degree of uncertainty linked to their predictions. Probabilistic design, which is the combination of probabilistic modelling techniques with the currently available deterministic models (steady-state or dynamic models), provides a solution (Bixio *et al.,* 2002). By building a probabilistic shell around the deterministic models, one can quantify the uncertainty contained within the model predictions. For example, a goal can be to determine the probability of exceeding the legal effluent standards of a WWTP. This percentage of exceedance should be accompanied by confidence intervals indicating the inherent uncertainty of influent characteristics and model parameters.

The quantification of the uncertainty of the system as a whole, is carried out by the following steps (Rousseau *et al.,* 2001):

doi: 10.2166/wst.2006.032

1. Assigning information about the probability distribution of each parameter and input variable in the system.
2. For every calculation, the simulation uses a value for each parameter and input variable that is randomly selected by a Monte Carlo engine from the appropriate probability density function. The set of random samples ("shots") is entered into the deterministic model.
3. The deterministic model is then solved for each shot, as it would be for any deterministic analysis. Since these simulations are all independent from each other, they can be run concurrently.

This iterative process generates a probability density function or cumulative density function of the output. Based on the distribution of the output, a risk level representing the high-end (e.g. 95th percentile), central tendency (median or mean), or any other desired level of probability can be identified. Characterization of uncertainty allows decision-makers to choose whether to actively take measures or to conduct additional research.

Considering the challenging task of model calibration, for cumbersome models with many parameters such as river water quality models (Reichert *et al.,* 2001) and integrated urban wastewater models (Meirlaen *et al.,* 2001), a trial-and-error calibration is very tedious, if not impossible. Automatic calibration procedures also pose problems, mainly due to the high correlation that often exists between the many parameters. A two-step, semi-automated procedure was therefore suggested (Vandenberghe *et al.,* 2001). In the first step, a global sensitivity analysis for the model parameter is performed (Saltelli *et al.,* 2000). Sensitivity analysis of models with a large numbers of parameters has become feasible through the Latin Hypercube Sampling technique (McKay, 1988), whereby the number of model runs can be limited to ca. 4/3 times the number of parameters. An automated calibration with an optimization algorithm is then performed in a second step, including only those parameters that have a significant effect on the model output.

Optimal experimental design is another tool to support the calibration of complex dynamic models, trying to systematically determine the experiments to perform on the modelled system in view of collecting maximum information with limited resources (Dochain and Vanrolleghem, 2001; De Pauw and Vanrolleghem, 2004). This approach uses local sensitivity analysis and multi-objective genetic algorithms (Fonseca and Fleming, 1998). This implies that a large number of simulations are to be run, but not necessarily in sequence. For example, using any evolutionary optimization algorithm, the simulations needed to assess the performance of individuals of each population can be run concurrently.

The fourth type of VE taken as an example is scenario analysis, in which the operational parameter space is explored in the search for optimal operational strategies. To provide an idea of the computational requirements: in order to improve the operation of an SBR system for N and P removal (Sin *et al.,* 2004), 2800 simulations had to be performed lasting 25 minutes each with a 1 GHz Pentium III processor, leading to a total simulation time of 7 weeks.

WEST and distributed processing

WEST (Vanhooren *et al.,* 2003) is a versatile yet powerful modelling and simulation system, which so far has mainly been applied to water quality management. The system consists of a clearly separated modelling environment and experimentation environment. Both environments are self-contained and consist of a graphical front-end, a computational back-end and control logic. The modelling environment allows for the creation of executable models on the basis of high-level model descriptions, through the application of model compiler techniques. These executable models are subsequently used as a basis

for the creation of VEs in the experimentation environment. The reason why the term "VE" was adopted is related to the fact that WEST goes beyond plain simulation. In fact, the types of VEs that are currently supported are: simulation, steady-state analysis, parameter estimation, confidence analysis, scenario analysis, sensitivity analysis and uncertainty analysis.

Historically, complex computational problems have often been tackled in a centralized manner, using supercomputer infrastructures. However, with the advent of microcomputers, interest in distributed architectures (such as clusters and peer-to-peer networks) has been steadily growing. It was therefore suggested to take on the computational problem of VEs in a distributed manner.

The first question that arises when trying to provide a distributed solution to the execution of VEs is related to the way the overall problem is decomposed into a number of subproblems, to be executed on different machines. At first sight, one might envisage splitting up all types of experiments: composite as well as atomic. However, due to the tightly coupled nature of atomic experiments (i.e. simulations), one must conclude that these experiments are not suited for such purposes. Distributed simulation (Fujimoto, 2000) is a complex issue that requires a complex infrastructure. Only composite experiments are therefore fit for distribution, and evidently it seems most logical as well as efficient to follow the natural hierarchical structure of an experiment during decomposition.

A second question that arises is whether one wants to be able to remotely execute all types of experiments (composite as well as atomic). In principle, the computational problem could be resolved to a large extent by merely executing atomic experiments remotely. However, for reasons of orthogonality and future enhancement, it seems most appropriate to allow for the remote execution of all types of experiments.

Methods

Requirements

The following are the requirements that were taken into account during design and implementation of the WEST distributed virtual experimentation (WDVE) system:

- In order to be able to use distributed virtual experimentation in a diverse array of applications, it was to be conceived as a library (which offers the desired functionality to encapsulating applications) rather than as a stand-alone program.
- In practice, WDVE will be mainly used to distribute the load of a complex VE over a number of (unused) work nodes that one has available at one's organization. Most often (especially in the case of scientific/academic organizations), this pool of work nodes will be heterogeneous in the sense that different hardware and operating systems will be used. WDVE therefore had to be conceived as a cross-platform system that allows for interoperability.
- As mentioned before, all types of VEs (atomic as well as composite experiments that are encapsulated by a larger composite experiment) should be remotely executable.
- During the remote execution of experiments, progress monitoring should be possible.
- As for any kind of distributed system, fail-safety is a key issue.
- Dynamic registration of work nodes is required. This means that additional work nodes should be able to register with the WDVE at run-time. Also, work nodes should be able to deregister at will. The latter may imply that experiments have to be rescheduled.
- Intelligent selection of experiments and work nodes is required.

- Since WDVE is targeted towards a non-computer science user community, transparent installation and deployment are very important. In fact, this requirement may in many cases outweigh the obvious performance requirement.
- In order to guarantee transparent installation and deployment as well as cross-platform support, it is important for WDVE to be based on as few third-party components as possible. Vendor and product independence can therefore also be stated as a requirement.
- The last requirement is one that has far-stretching consequences. In fact, it concerns the fact that, albeit primarily intended for the execution of WEST VEs, it should also be possible to apply the WDVE framework to other types of problems (not necessarily related to water management or even simulation as such) leading to a need for application independence.

Architecture

WDVE consists of two main modules: Master and Slave. A Master receives requests for the execution of a number of Jobs from the Application front-end (Figure 1). The Master buffers the Jobs it receives and distributes them to a number of Slaves that have registered with the Master. Slaves perform the actual (possibly concurrent) execution of Jobs received from one or more Masters. The entity within the Master that performs the distribution of Jobs to Slaves was named Dispatcher. The entity within the Slave that awaits incoming Jobs was labeled Acceptor. For the actual transfer of data over the wire, traditional middleware solutions can be used. In order for the specifics of the middleware not to be visible by the rest of the code, an abstraction layer was introduced.

Figure 2 presents a more detailed view of the internal structure of Master and Slave. Next to the Dispatcher and the Application front-end, the Master also contains a Registry,

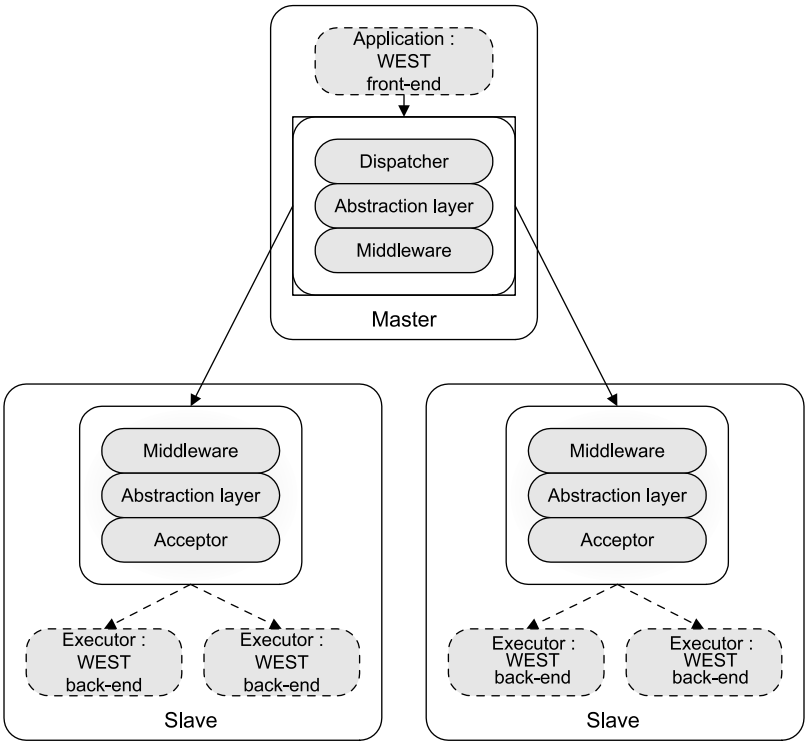

Figure 1 High-level architecture

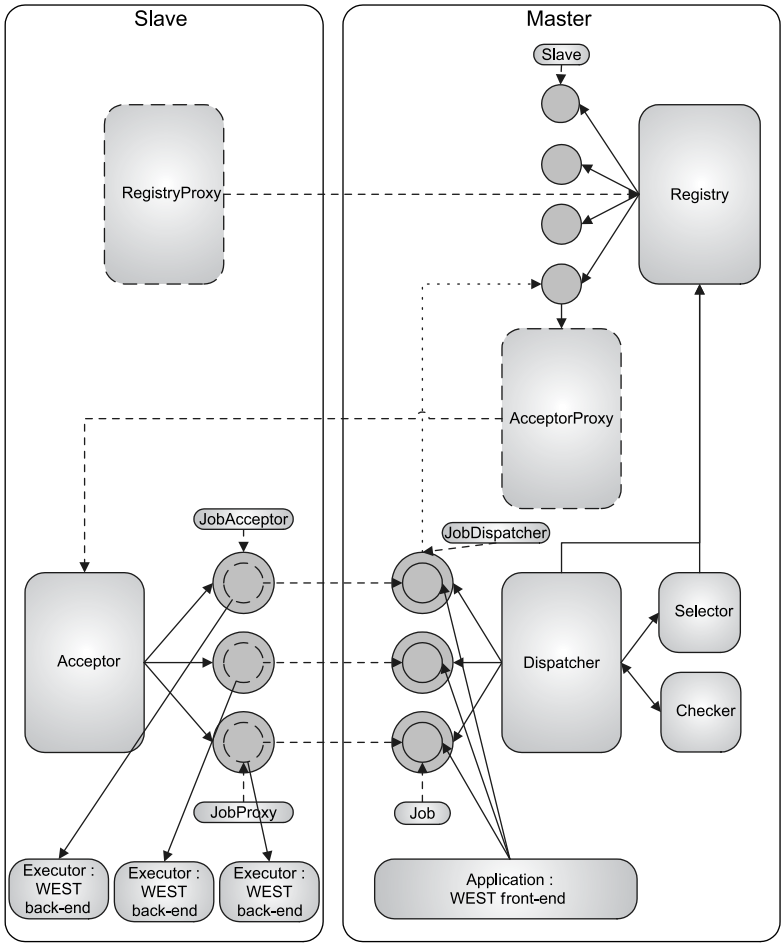

Figure 2 Internal Master and Slave structure

Selector and Checker. The Registry is where information is maintained about Slaves that have registered. The Selector implements the selection algorithm that determines the next Job to be distributed, and the Slave it is to be sent to. Finally, the Checker implements a background thread that ensures consistency by constantly monitoring the status of various entities such as Jobs and Slaves. Jobs handed to the Dispatcher by the Application front-end are first wrapped by a so-called JobDispatcher entity before they are actually stored into the Dispatcher's buffer. One last entity that is relevant in the context of the Master is the AcceptorProxy. This is a local representation of the remote Acceptors that are part of the respective Slaves that have registered with the Master. Through the proxy approach, the Master can interact with these Acceptors as if they were local objects.

From an architectural point of view, the Slave module is less complex than the Master module. Next to the Acceptor and the various instances of the Application back-end or Executor (used to handle the respective Job execution requests), there is also a RegistryProxy, which represents the remote Registry that the Slave registers with. Local counterparts of remote Jobs are represented by JobProxy entities. Before these are stored in the Acceptor's buffer, they are wrapped by a so-called JobAcceptor entity, much in the same way Jobs are wrapped by a JobDispatcher in the Master. An important architectural question was how to represent Jobs in their various incarnations. It was decided to consistently use XML for this purpose.

In the context of WDVE, Job descriptions are made up of Executor-dependent and Executor-independent information. The Executor-independent information consists of a set of Resources including various types of input and output files. The Executor-dependent information contains other pieces of information that are relevant to the specific Executor that is to execute the Job.

Technology

WDVE was entirely developed in C++, in conjunction with the STL library. The reasons for opting for C++ are performance, portability and consistency with the WEST back-end, which was also developed in C++.

SOAP was selected as middleware, and more specifically the gSOAP implementation. Reasons are again performance as well as portability. In addition, gSOAP is freely available and has a proven track record.

For threading and synchronization, XML parsing and URL management, the OpenTop class library is used. This open source, commercial library is free of charge for academic use. It is well structured, heavily object-oriented and conveniently documented.

Results and discussion

Deployment and test results

Since WDVE is conceived as a library, it is normally deployed from within applications such as WEST. However, in order for the WDVE functionality to be demonstrated and tested, two command-line programs were developed. These programs were named wmaster and wslave and respectively represent a Master and Slave as earlier depicted in the architectural overview.

So far, four types of tests were run on WDVE, using the aforementioned command-line programs:

1. Through the first set of tests the correct behavior of WDVE under normal circumstances was demonstrated. This involves amongst other tests correctness of parsing, transmission and execution of Jobs.
2. The second set of tests was intended to verify the correct behavior of the system under faulty conditions, which is of the utmost importance in the case of distributed systems. During these tests it has been demonstrated that in the event that a Slave crashes, the Master will eventually redistribute the Jobs that were in execution on that Slave. Also, in case no Slave is available to execute Jobs, the Jobs remain buffered in the Master until one or more Slaves become available.
3. The third series of tests was targeted towards cross-platform support. It was demonstrated that the WDVE software correctly installs and operates on platforms such as Windows NT, Windows 2000, Windows XP and Linux. Interoperability between Windows and Linux Masters and Slaves was equally demonstrated. However, there is one issue related to cross-platform application that complicates matters (although it is not related to WDVE as such). This issue concerns the fact that the executable models that are at the heart of simulations are most often composed of machine-specific binary code. In order to overcome this problem, one can follow either of two approaches. The first approach entails the use of operation system emulators. Indeed, through the use of a tool such as wine (http://www.winehq.com/) it is possible to run Windows software (in this case executable models) on Linux machines. It is therefore possible to distribute experiments that are based on a Windows executable model over a heterogeneous (Windows/Linux) pool of machines, provided a Windows emulator is available. The second approach does not require the use of advanced techniques such

as emulation. In this approach multiple binary versions of the same executable model are being sent to Slaves.

4. The final series of tests was intended at measuring the performance of the system. The results that were found are in line with the typical performance behavior of other distributed systems. In Figure 3 are some of the results that were obtained from a test using three identical Slaves that execute Jobs assigned by a Master that has a total of 1 to 100 Jobs to execute. From Figure 3 (left) one easily concludes that in case the Jobs are short (in this case: artificially short), there is nothing to be gained from distributing Jobs over multiple Slaves. Actually, the total execution time of distributed execution is worse than in the case that the Jobs are sequentially executed on one machine. However, in the case of Jobs of realistic complexity, one does get a significant reduction of the total execution time (Figure 3, right). As the number of Slaves increases, this reduction becomes more pronounced, up to a point however where no further improvement is possible. The speed-up factors obtained in practice are always lower than those that one would theoretically expect. Evidently, this is due to the overhead inflicted by WDVE itself and the transmission of data over the network.

WDVE versus Grid Computing

There are some obvious similarities between the Grid Computing paradigm, which has recently received widespread attention, and WDVE. These similarities include the high-level functionality (the distribution of a number of tasks over a number of work nodes), the importance of the fail-safety aspect, and the fact that work nodes should be able to dynamically register and deregister.

However, there are also a number of differences between Grid Computing and WDVE, which are important enough to justify the development of WDVE. These differences include the following:

- *Scope*: Grid software is highly complex and goes far beyond what WDVE attempts to deliver. As a result, the WDVE code base is less than 50 k LOC whereas Grid software code bases are most often many times larger. The latter of course imposes a serious threat to the manageability and maintainability of Grid projects.
- *Consistency*: WDVE was developed in a consistent manner, using as few third-party components as possible. Grid software on the other hand is built on top of a large number of external tools, thereby again hampering its manageability and maintainability.
- *Installation*: WDVE installs in a matter of minutes. Installing Grid software is currently a lengthy and cumbersome experience.

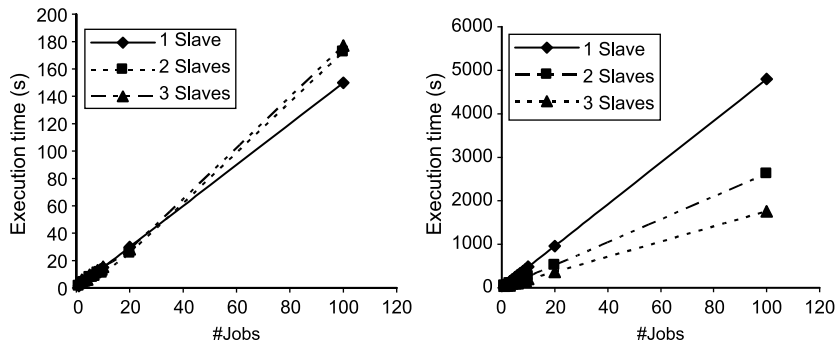

Figure 3 Performance of WDVE with short (left) and long (right) Jobs

- *Intelligence*: The intelligence claimed by WDVE when distributing Jobs is quite limited at this point, but work is ongoing. In this respect Grid software – at least in theory – goes much further.
- *Authentication*: WDVE is mainly intended for deployment within one organization. Issues such as security and confidentiality of data have therefore not been taken into account. Also in this respect, Grid software goes much further since it has a wider scope.
- *Integration*: WDVE is conceived as a library that allows for straightforward integration into applications. Grid software is usually based on stand-alone servers and executables.
- *Portability*: Although most Grid software is intended to be portable, this proves not to the case in practice (because of the many restrictions imposed by the external components that are used, security aspects, system performance monitoring aspects, etc.). WDVE however was proven to be portable.

WDVE versus SuperMUSE

- SuperMUSE (Babendreier and Castleton 2002) is a supercomputer with a set of applications that is targeted to deliver a specific capability (uncertainty and sensitivity analyses). WDVE is a general software system aimed at integration in different distributable applications.
- SuperMUSE is regarded to be a cheap solution, which can be deployed by non-computer science experts. The same arguments are even more applicable to WDVE, for it requires no extra investments and is extremely easy to install.
- SuperMUSE needs a cluster infrastructure. WDVE works fully in a heterogeneous environment.
- Experiments (jobs) are executed sequentially on work computers (no multi-threading as in WDVE).
- There is no registry for work nodes ("pooling" is used instead), which should result in increased network load.
- SuperMUSE's distributor doesn't possess a complete view of the available work nodes and their load (due to the absence of a registry), so it should be more difficult to find a good load-balancing algorithm.
- Detection of computer failures is implemented in hardware (KVM switches). WDVE provides a software solution.

Conclusions

WEST is a flexible and powerful system for modelling biological processes, as well as for defining and executing so-called VEs on the basis of these models. Since the complexity of VEs (optimal experimental design, global sensitivity analysis, probabilistic design, scenario analysis, etc.) and their underlying models is constantly increasing, performance of monolithic software solutions is rapidly becoming insufficient. In order to tackle this computational bottleneck, a framework for the distributed execution of VEs on a potentially heterogeneous pool of work nodes has been implemented. This framework was named WDVE and has been built on top of technologies such as C++, XML and SOAP. It was designed for stability, expandability, performance, platform-independence and ease of use. In many ways WDVE can be regarded as similar to the Grid Computing paradigm, but is does not suffer from many of the problems that often make the deployment of Grid Computing applications a cumbersome experience. Although not explicitly mentioned in the article, initiatives for further research on WDVE and related topics have been taken.

References

Babendreier, J.E. and Castleton, K.J. (2002). Investigating uncertainty and sensitivity in integrated, multimedia environmental models: tools for 3MRA. In *Proceedings of iEMSs 2002*, Lugano, Switzerland, 24–27 June 2002.

Bixio, D., Parmentier, G., Rousseau, D., Verdonck, F., Meirlaen, J., Vanrolleghem, P.A. and Thoeye, C. (2002). A quantitative risk analysis tool for design/simulation of wastewater treatment plants. *Water Science and Technology*, **46**(4–5), 301–307.

De Pauw, D.J.W. and Vanrolleghem, P.A. (2005). Designing and performing experiments for model calibration using an automated iterative procedure. *Wat. Sci. Tech.*, **53**(1), 117–127 (this issue).

Dochain, D. and Vanrolleghem, P.A. (2001). *Dynamical Modelling and Estimation in Wastewater Treatment Processes*, IWA Publishing, London, UK.

Fonseca, C.M. and Fleming, P.J. (1998). Multiobjective optimization and multiple constraint handling with evolutionary algorithms - Part I: A unified formulation. *IEEE Transactions on Systems, Man, and Cybernetics*, **28**, 26–37.

Fujimoto, R. (2000). *Parallel and Distributed Simulation Systems*, Wiley Interscience, New York.

McKay, M.D. (1988). Sensitivity and uncertainty analysis using a statistical sample of input values. In *Uncertainty Analysis*, Ronen, Y. (ed.), CRC Press, Inc., Boca Raton, Florida, pp. 145–186.

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. *Water Science and Technology*, **43**(7), 301–309.

Reichert, P., Borchardt, D., Henze, M., Rauch, W., Reichert, P., Somlyódy, L. and Vanrolleghem, P.A. (2001). River Water Quality Model No. 1: II. Biochemical process equations. *Water Science and Technology*, **43**(5), 11–30.

Rousseau, D., Verdonck, F., Moerman, O., Carrette, R., Thoeye, C., Meirlaen, J. and Vanrolleghem, P.A. (2001). Development of risk assessment based technique for design/retrofitting of WWTPs. *Water Science and Technology*, **43**(7), 287–294.

Saltelli, A., Chan, K. and Scott, E.M. (eds) (2000). *Sensitivity Analysis*, Wiley, Chichester, UK.

Sin, G., Insel, G., Lee, D.S. and Vanrolleghem, P.A. (2004). Optimal but robust N and P removal in SBRs: A systematic study of operating scenarios. *Wat. Sci. Tech.*, **50**(10), 97–105.

Vandenberghe, V., van Griensven, A. and Bauwens, W. (2001). Sensitivity analysis and calibration of the parameters of ESWAT: application to the River Dender. *Water Science and Technology*, **43**(7), 295–301.

Vanhooren, H., Meirlaen, J., Amerlink, Y., Claeys, F., Vangheluwe, H. and Vanrolleghem, P.A. (2003). WEST: Modelling biological wastewater treatment. *Journal of Hydroinformatics*, **5**(1), 27–50.