ELSEVIER

0266-9838(95)00010-0

Environmental Software, Vol. 10, No. 3, pp. 199–210, 1995
Copyright © 1995 Elsevier Science Limited
Printed in Great Britain. All rights reserved
0266-9838/95 $9.50+0.00

Design techniques of a computer program for the identification of processes and the simulation of water quality in aquatic systems

Peter Reichert

Swiss Federal Institute for Environmental Science and Technology (EAWAG), CH-8600 Dübendorf, Switzerland

(Received 16 January 1995; accepted 24 July 1995)

ABSTRACT

In order to support environmental scientists in finding an "adequate" model for the system they are investigating, a computer program is necessary that allows its users to perform simulations for different models, to assess the identifiability and to estimate the values of model parameters (using measured data), and to estimate prediction uncertainty. These requirements, especially that of providing much freedom in model formulation, are difficult to realize in such a program. In this paper, it is shown how object-oriented program design techniques were employed to facilitate the realization of an identification and simulation program for aquatic systems (AQUASIM) that is very flexible with regard to model formulation and that provides methods of sensitivity analysis, parameter estimation and uncertainty analysis in addition to simulation. It is the goal of this paper to encourage developers of environmental software to revise previously used program structures and to employ modern program design techniques.

KEYWORDS

Aquatic systems, modelling, simulation, system identification, parameter estimation, object-oriented programming.

SOFTWARE AVAILABILITY

Name:	AQUASIM
Developers:	Peter Reichert and Jürg Ruchti, Computer and Systems Sciences Department, Swiss Federal Institute for Environmental Science and Technology (EAWAG), CH - 8600 Dübendorf, Switzerland
Contact address:	Mail: Peter Reichert, EAWAG, CH-8600 Dübendorf, Switzerland E-mail: reichert@eawag.ch Fax: +41 - 1 - 823 5398
Year first available:	1994
Hardware required:	- Sun SparcStation or SparcServer (Solaris 2.x) - IBM RS/6000 (AIX 3.2.x; character and batch interface versions only) - HP 700 workstations (HP-UX 9.x; character and batch interface versions only)

doi:10.1016/0266-9838(95)00010-I

	- Intel 80486/Pentium (DOS/Windows 3.1, Windows 95 or Windows NT 3.x) - Apple PowerMacintosh (MacOS 7.x)
Program language:	C++
Availability and cost:	The program is available at a fee of SFr. 1000.-- for one platform. The license agreement allows multiple installations within the research group and for teaching. The distribution includes the binary program versions and a detailed technical report (400 pages, of which 150 pages for the user manual with tutorial). There is no technical support on program usage (however, program errors are corrected), but e-mails are distributed to program users registered in the AQUASIM user group to facilitate exchange of experience between users.

INTRODUCTION

Scientific knowledge expands as measurements are compared with conclusions drawn from hypotheses on mechanisms occuring in the investigated system (Popper, 1982). Such comparisons either lead to the rejection of the hypotheses or they show that the hypotheses are compatible with the observed behavior of the system. Hypotheses on functional relationships in environmental systems are usually stated in the form of mathematical models. With the exception of very simple (but important) models, a computer program is required for the calculation of the consequences of model assumptions. Many computer programs are available that can be used for this purpose. These programs can be grouped roughly into the three categories of

- general purpose simulation software,
- conventional environmental simulation programs,
- tools for model identification.

This classification is not strict, because there exist also programs covering tasks of more than one category. Nevertheless, this classification correctly characterizes the majority of currently available programs. General purpose simulation software (e.g. Carver et al., 1978; Schiesser, 1991) gives its users a large freedom in model formulation, but it is difficult to use and does not usually support model identification and uncertainty estimation. Conventional environmental simulation programs (e.g. Ambrose and Barnwell, 1989) use a communication "language" more familiar to environmental scientists and are much easier to use, but most programs of this category are too specific to allow their users to perform comparisons of different models, a task that is necessary for system identification. Finally, almost all programs for model identification (e.g. Jamshidi and Herget, 1985), mainly developed for control engineering, are limited to single-input single-output and/or linear models. This survey of available software demonstrates the need for a new generation of environmental system identification software, which combines the functionality of all three program categories mentioned above.

As a step towards more universally applicable environmental software, the program AQUASIM was developed to identify and simulate aquatic systems in the environment, in technical plants and in the laboratory. A description of the capabilities of this program, stressing the users point of view, and an example of the application of the program to parameter identification of activated sludge models are given elsewhere (Reichert, 1994a; Reichert et al. 1995). It is the goal of the present paper, after summarizing the program concepts, to discuss the most important techniques employed for designing this program. Apart from a short discussion of general program structure, the paper concentrates on demonstrating the utility of object-oriented program design techniques for improving program structure, for increasing its robustness and extensibility and for saving development time. This paper reviews the most important design concepts; more details on model structure, differential equations solved by the program, selection of program functionality, numerical methods and design concepts as well as examples of applications are given in a more detailed technical report (Reichert, 1994b). More examples of program applications are given in Wanner et al. (1994), Albrecht et al., (1995), Reichert et al. (1995) and Wanner and Reichert (1995).

SUMMARY OF PROGRAM CONCEPTS

As mentioned in the introduction, it was the goal of the program AQUASIM to extend the capabilities of conventional environmental simulation programs mainly in two ways: The program should not implement a specific model, but allow users to specify models as freely as possible and it should offer system identification tasks in addition to simulation. The general model structure designed to fulfill the first requirement for a class of aquatic systems and the program functionality selected to make a step towards fulfilling the second requirement are described in the following two subsections.

General Model Structure

AQUASIM model formulation is based on a division of the aquatic system into compartments that describe regions with restricted mutual interactions exchanged by links connecting well-defined interfaces. Within the compart-

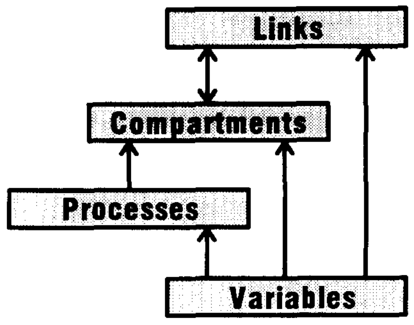

Fig. 1: Logical structure of AQUASIM systems consisting of four subsystems.

ments, arbitrary transformation processes can be specified for an arbitrary number of state variables. Process rates and compartment and link definitions are specified parametrically using previously defined variables. Fig. 1 shows the logical structure of AQUASIM systems, consisting of the four subsystems of variables, processes, compartments and links. Variables form the basic subsystem because they are required for the formulation of processes, compartments and links. Processes can be activated in compartments and links connect compartments. The mutual dependences of subsystems are indicated by arrows in Fig. 1.

Variables form the basic subsystem of AQUASIM model structure. Their common property is that they return a numerical value that depends on the calculated system state. The user of the program can define an arbitrary number of variables, which are identified by their names. The subsystem of variables serves as a pool of variables for use in the other subsystems and for calculating results. Six types of variables belonging to three categories are distinguished:

- system variables:	state variables, program variables,
- data variables:	constant variables, real list variables,
- function variables:	variable list variables, formula variables.

The first category of variables, system variables, represent quantities to be determined by the program (state variables usually represent concentrations of substances transported in dissolved or suspended form with water or attached to a surface) or which make properties of a compartment, space coordinates or time available in the system of variables (program variables). The second category of variables, data variables, are used to make measured data available to the program. It is possible to specify single measured parameters (constant variables) or series of argument-value pairs (real list variables), which can be used as continuous functions interpolating given data or as series of isolated data points to be approximated by variables during parameter estimation. The third category of variables, function variables, are used to build functional relationships using other variables. Interpolations between variables (variable list variables), and algebraic expressions formulated in the usual algebraic syntax, extended by logical branching, can be realized (formula variables). Function variables are used for the formulation of processes and the definition of compartments and links.

To allow the user to separate time scales of slow and fast processes, two types of processes are distinguished:

- dynamic processes,
- equilibrium processes.

Dynamic processes are formulated as source terms in differential equations, whereas equilibrium processes describe very fast processes, the transient phase of which is not important on the time scale of interest, and which, therefore, can be characterized by algebraic equations. The user of the program can define an arbitrary number of processes, which are identified by their names. The subsystem of processes serves as a pool of processes, which can individually be activated in compartments.

In contrast to the generality of the realization of transformation processes, compartments are designed much more specifically in order to make it possible to use efficient numerical discretization techniques, which take advantage of the known type of the partial differential equations describing the dynamics of state variables in the compartment. In the first version of AQUASIM, three types of compartments are available:

- mixed reactor compartments,
- biofilm reactor compartments,
- river section compartments.

A mixed reactor compartment models a fluid (or gas) volume in which concentration gradients can be neglected (e.g., a stirred laboratory reactor, a mixed basin of a waste water treatment plant or a well mixed lake), a biofilm reactor compartment represents a mixed reactor on the wall of which a biofilm grows. The population dynamics of microorganisms and the concentration gradients of dissolved substances within the biofilm are calculated according to the equations given by Wanner and Reichert (1995), which are extensions of the biofilm model of Gujer and Wanner (1989). A river section compartment describes water flow and transport and transformation of substances in a reach of a river. One-dimensional hydraulics is calculated according to the kinematic or diffusive approximation to the St. Venant equations of open channel flow (e.g. Yen, 1973; Yen, 1979); transport and transformation processes are calculated with a set of advection-dispersion-reaction equations for all transported substances.

The last subsystem of model structure consists of links, which are used for connecting compartments. The following two types of links are distinguished:

- advective links,
- diffusive links.

Advective links are used to describe water flow and advective substance transport between compartments. These links make it also possible to model bifurcations, junctions

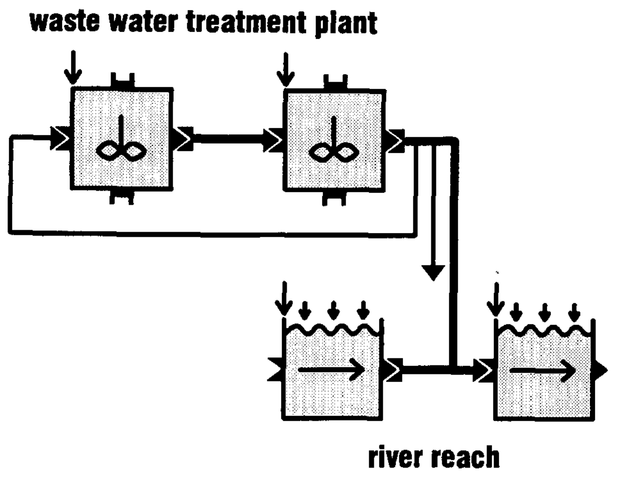

Fig. 2: Pictogram of an AQUASIM configuration representing a waste water treatment plant together with the river into which the treated water is released.

and separation of fluxes of different substances (e.g., separation of particulate substances from dissolved substances). Diffusive links can be used to model molecular boundary layers or membranes, which can be penetrated by certain substances.

The model structure framework described in this section gives the user of the program great freedom in defining the model. Fig. 2 shows an example of an AQUASIM configuration representing a waste water treatment plant modelled with two mixed reactor compartments, and the river into which the treated water is released modelled with two river section compartments. Transport of water and substances between the compartments is described by advective links.

Program Functionality

In order to address the needs of an environmental system identification tool, as described in the introduction, the following four functionalities are supported by AQUASIM:

- simulation,
- identifiability analysis,
- parameter estimation,
- uncertainty analysis.

Due to the similarity of mathematical techniques, identifiability and uncertainty analyses are combined in the single task of sensitivity analysis.

The first of AQUASIM's functionalities is model simulations. By comparing calculated results with measured data, such simulations can be used to validate model assumptions. Systematic deviations between calculated and measured values may suggest additional important processes to be considered or corrections needed in the formulation of processes. AQUASIM easily allows the user to change model structure and parameter values.

A second functionality of AQUASIM is sensitivity analysis with respect to a set of selected variables. This feature enables calculation of linear sensitivity functions of arbitrary variables with respect to each of the parameters included in the analysis. These sensitivity functions help to assess the identifiability of model parameters. Furthermore, sensitivity analysis allows the user to estimate the uncertainty of any calculated variable according to the linear error propagation formula. The calculation of the contribution of each parameter to total uncertainty facilitates identification of major sources of uncertainty.

The third important functionality of AQUASIM is automatic parameter estimation for a given model structure using measured data. These estimations are done using the weighted least-squares technique. This is not only important for getting neutral estimates of parameters, but it is a prerequisite for efficiently comparing different models. Several calculations with several target variables, and universal as well as calculation-specific parameters, can be combined to a single parameter estimation process. This is a very important feature not present in most of the statistical software.

PROGRAM DESIGN TECHNIQUES

In this main section of the paper, the most important techniques used for the design of the identification and simulation program AQUASIM, described in the previous paragraph, are outlined. Since the design of flexible model formulation for simulations is the most difficult part with respect to program structure, the discussion is mainly focused on this part of the program. The problems to be solved for the design of system identification functionality are more of numerical nature and are not discussed in this paper. Readers interested in this field should refer to the more detailed technical report on program concepts (Reichert, 1994b).

An important decision to be made before starting program development is the choice of the programming language. Due to the rapid change of computer hardware, it is essential to select a language which is standardized and for which compilers are available on all important platforms. Apart from historical reasons, this is the main reason why the majority of simulation software is written in FORTRAN. Because this choice would have made object-oriented program design impossible, another language had to be selected for the project described in this paper. The following four reasons led to the selection of the object-oriented extension of C (e.g. Haribson and Steele, 1991), C++ (e.g. Lippman, 1989; Ellis and Stroustrup, 1991) as the programming language for the implementation of AQUASIM:

- C++ was (at the start time of this project in 1991) and still is the best standardized object-oriented programming language, and C++ compilers are available on all important platforms,
- C++ implements object-oriented concepts very efficiently,

- graphical user interface libraries, which are usually written in C, can easily be used from C++ programs (there is now an increasing number of graphical user interface libraries directly available in C++),
- there exist automatic source code translation programs from FORTRAN to C or to (non object-oriented) C++, which can be used to convert well tested numerical software written in FORTRAN. Therefore, linking of object files based on different languages can be avoided (this facilitates program portability).

Note that the reasons for selecting C++ are of technical nature and that the object-oriented design concepts discussed in this paper could also be realized with any other object-oriented programming language.

General Program Structure

Due to the rapid change of important computing platforms and in order to be able to account for the habits of the users, portability is a main requirement for an environmental system identification program. The choice of a well-standardized programming language, as discussed in the previous paragraph, is a good basis for fulfilling this requirement. Because standardization of input and output procedures in programming languages only covers the level of primitive character-oriented communication, and because the application programmer interfaces of graphical user interface libraries on different platforms are completely different, the choice of a universally available standardized programming language does not guarantee portability of user-friendly programs using such libraries.

There exist several software companies, which sell graphical user interface (GUI) libraries that offer a common application programmer interface (API) on different platforms (e.g. Côté, 1992; Apiki, 1994). With the use of such a library, a program can be written that is portable to all platforms supported by the library. The disadvantage of this solution to the portability problem is, that there are no standards for such universal API's. Therefore, portability of the program is achieved at the expense of dependence on decisions and fate of a (additional) software company.

In order to save as much of tested source code as possible in the case of changes of graphical user-interface libraries, AQUASIM was implemented using the program architecture shown in Fig. 3. The program was divided into a large portable core program and relatively thin user-interface layers. The core program provides functions for all program functionality: editing the model, simulation, system identification and reporting results. This part is completely independent of user-interfaces. The job of a relatively thin user-interface layer is to provide communication between the user and the core program. Such a program architecture not only limits the effort needed for adapting the program to changes in user-interface libraries, but it makes it easily possible to implement more than one user-interface. For these reasons, this program architecture is advantageous, independent of whether a third party

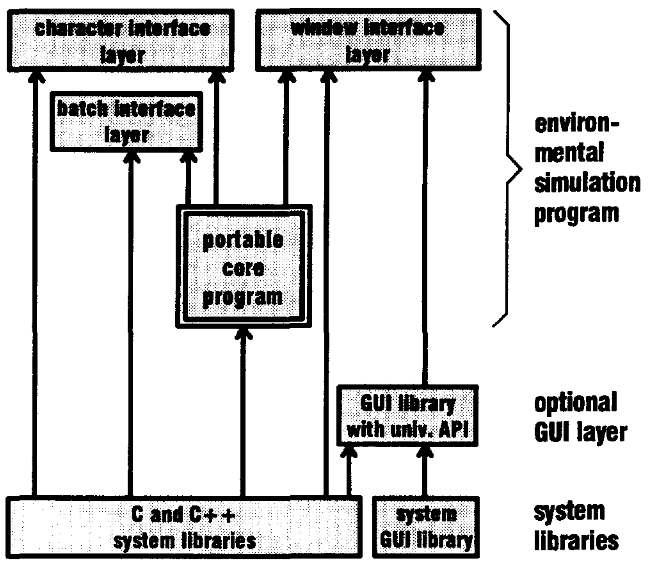

Fig. 3: Recommendable program architecture for environmental simulation programs. Division of the program into a large portable core and relatively thin user-interface layers makes the program transferable to other platforms with a reasonable programming effort. Usage of a GUI library with a universal API increases portability but makes the program dependent on the company selling the library.

user-interface library is used or not. In the case of the program AQUASIM, three user-interface layers were written (cf. Fig. 3). The window interface layer, which leads to the most user-friendly version of the program, is written for the API of the third party interface library XVT (e.g. Rochkind, 1989; Carpenter, 1991). This program version is recommended for editing models, defining sensitivity analyses and parameter estimations, specifying plot definitions, performing short calculations and viewing results. The second version is the character interface version designed for using AQUASIM with the aid of a simple terminal without graphical capabilities. This version, which only uses standard C++ input and output procedures, can be transferred easily to any platform without the need for additional libraries (cf. Fig. 3). The third program version is the batch version, which only parses the command line and executes one of five possible jobs (performing a simulation, a sensitivity analysis or a parameter estimation or plotting or listing results). This version was designed to allow the user to submit long calculations as batch jobs. The different parts of the program have the following approximate numbers of lines of source code (version 1.0; including comments and empty lines):

portable core program:	50'000	lines
character interface layer:	7'500	lines
window interface layer:	17'000	lines
batch interface layer	500	lines
initializations and resource definitions:	6'500	lines

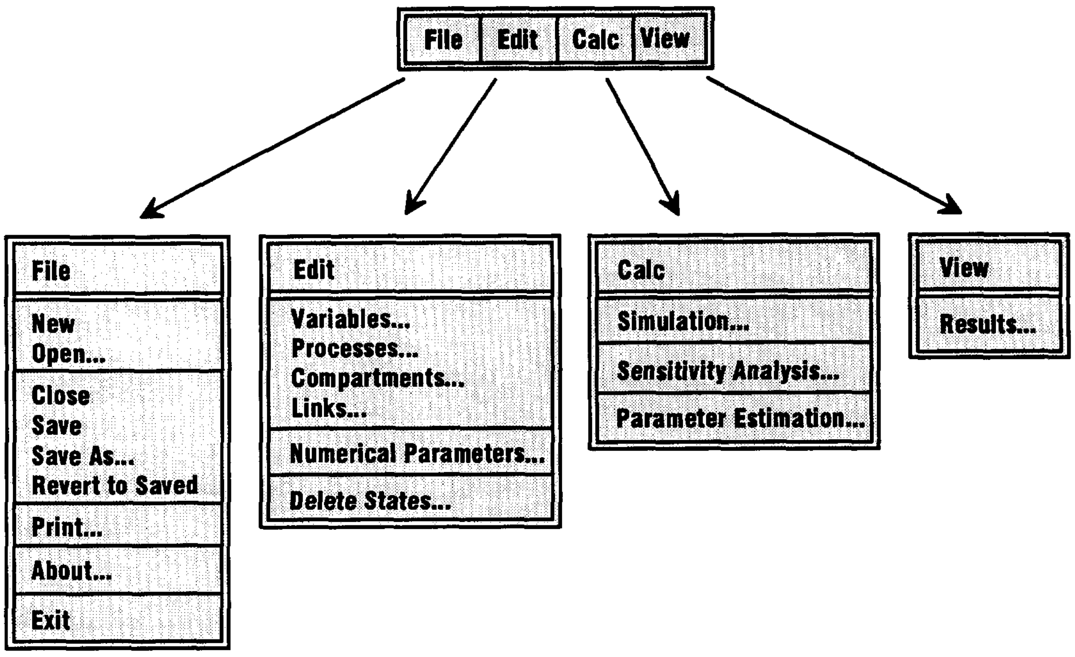

Fig. 4: Menus of AQUASIM.

This list shows that, in spite of the attempt to make the user interface layer as thin as possible, the large number of dialog boxes (more than 60) forces the user-interface layer also to require a large programming effort. It should be noted, however, that the time needed for writing the same number of program lines is much shorter for interface layer code than for core program code. The batch interface layer is much smaller than the layers of the interactive versions, because editing of a model is not supported by this version. A survey of the menus of the window interface version (which are the same as in the character interface version) is given in Fig. 4.

Main Class Hierarchy

The essential point and at the same time the most difficult problem of object-oriented program design is to find a good class hierarchy, which allows the programmer to optimally profit from the key concepts of object-oriented programming (e.g. Marty, 1988; Meyer, 1990; Budd, 1991):

- information hiding (shielding unnecessary information from the programmer who uses a program segment and limiting communication to a well defined interface),
- abstract data types (combination of data structures with operations allowed for the data),
- inheritance (derivation of new abstract data types from previously defined types so that features of the old type remain usable and only the modifications necessary for the new type must be implemented),
- polymorphism (capability of a class hierarchy that calling the same method of a base class can result in different actions for different derived classes).

In this section, the main class hierarchy of the program AQUASIM is reviewed briefly; examples demonstrating the use of these concepts are given in the next section.

Fig. 5 shows the main class hierarchy of AQUASIM. The classes VARSYS, PROCSYS, COMPSYS and LINKSYS implementing the model subsystems of variables, processes, compartments and links shown in Fig. 1, the classes SENSSYS, FITSYS and PLOTSYS collecting definitions of sensitivity analyses, parameter estimations and plots, the class STATESYS used for storing calculated states and the class AQUASYS collecting all system definitions, are all derived from a base class FILEIO, which provides basic procedures for loading, saving and printing. Because all important objects created by the user are identified by their names, a class SYMBOL is introduced to hold such a name. To make it possible to insert such objects into lists, this class is derived from the class NODE, which provides connection mechanisms for dynamic lists. The class SYMBOL is also derived from the class FILEIO to inherit basic procedures for loading, saving and printing. For the elements of each of the AQUASIM subsystems of variables, processes compartments and links, a base class VAR, PROC, COMP and LINK is provided which is derived from the class SYMBOL and from which the classes for specific types of objects are derived (STATEVAR for state variables, PROGVAR for program variables, CONSTVAR for constant variables, REALLISTVAR for real list variables, VARLISTVAR for variable list variables, FORMVAR for formula variables, DYNPROC for dynamic processes, EQUPROC for equilibrium processes, MIXCOMP for mixed reactor compartments, FILMCOMP for biofilm reactor compartments, RIVCOMP for river section compartments, ADVLINK for advective links and DIFFLINK for diffusive links). The

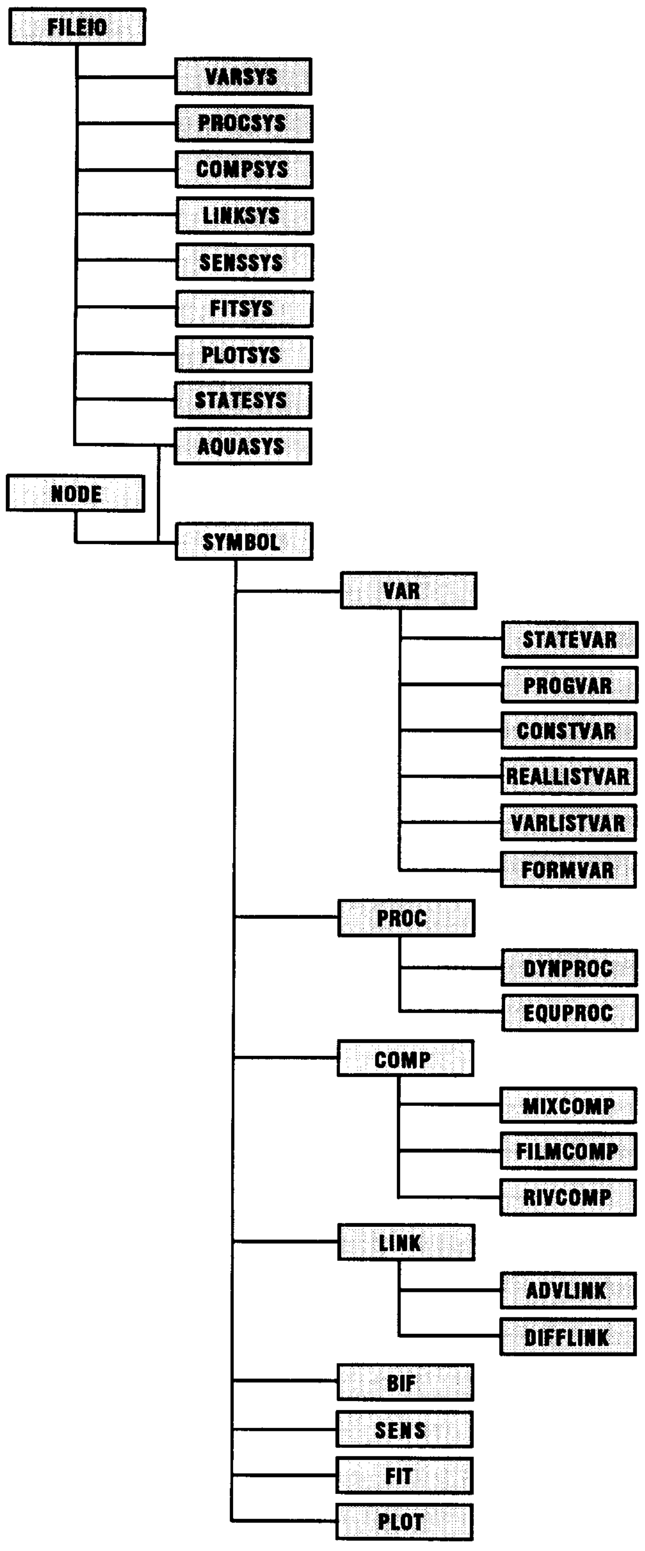

Fig. 5: Main class hierarchy of AQUASIM.

class BIF implementing bifurcations within advective links, and the classes SENS, FIT and PLOT holding definitions of sensitivity analyses, parameter estimations and plots, are also derived from the class SYMBOL. Table 1 gives an overview of the most important categories of methods (member functions) declared or realized at different class levels. It is clear, that classes, which are end nodes of the derivation hierarchy (no class is derived from such a class) do not have declared methods that are not realized. Methods realized in a non-end node class are inherited from all derived classes; methods only declared in a class and not yet realized at this level can be used as normal methods of the class (such methods are realized differently for different derived classes: polymorphism). Some highlights demonstrating the profit of object-oriented design which are based on the class hierarchy given in Fig. 5 and Table 1 are discussed in the next subsection.

Profit from Object-Oriented Design

In this section, the most important points are discussed, where the class hierarchy described in the previous section allows the programmer to profit from object-oriented design techniques. The four key concepts of object-oriented programming mentioned in the previous section, information hiding, abstract data types, inheritance and polymorphism are very closely interrelated. Nevertheless, the following discussion of key design concepts is grouped into these four categories.

Information Hiding

The advantage of defining interfaces to program segments and shielding details of the realization from the programmer using such a program segment has been recognised since the beginning of development and use of high level programming languages. The success of FORTRAN (e.g. Müller and Streker, 1984) subroutine libraries for numerics and graphics are an example for the use of this technique in non object-oriented languages. Modula-2 (Wirth, 1985), with its clear separation of definition and implementation modules, makes it possible to design much more general interfaces as compared to subroutine headers. This may be the most perfect implementation of this technique in non object-oriented languages. In object-oriented languages, the class interface is the most important level, at which information hiding is employed (header files are not a very important additional level, because they are mainly used for the definition of class interfaces).

In the case of the program AQUASIM, information hiding is consequently applied. The public member functions of the classes shown in Fig. 5 and briefly described in Table 1 define the interfaces to data elements that are not directly accessible. The level of abstraction of member functions from the underlying data structures differs considerably between member functions of different classes. An example of a low level of abstraction are the member functions of the class SYMBOL for storing and recalling the name of the object. The data structure passed as an argument of these functions, as well as the data structure stored in the class, are both of the same type of a string. An example of a high level of abstraction are the member functions declared in the class COMP defining the differential-algebraic equations of temporal evolution of the

Table 1: Categories of methods declared or realized in the classes of the hierarchy shown in Fig. 1.

class(es)	declared methods not yet realized at this level	realized methods
FILEIO	- procedures for saving and printing all data of the object	- procedures for loading, saving and printing elementary data types
VARSYS PROCSYS LINKSYS SENSSYS FITSYS PLOTSYS		- procedures for adding, replacing and deleting objects of the corresponding subsystem - procedures for updating changes made in other subsystems
STATESYS		- procedures for adding, recalling and deleting state-arrays
AQUASYS		- procedures for saving and printing all data of the system (declared in class FILEIO) - procedure for loading all data of the system - procedures for adding, replacing and deleting objects of AQUASIM subsystems (these procedures only call the procedures for performing the desired change of the subsystem of the object given as argument and the procedures for updating the other subsystems involved) - procedures for specifying calculation parameters and for executing program tasks (simulations, sensitivity analyses, parameter estimations) - procedures for getting calculated data and for listing and plotting results
NODE		- connections to neighbouring nodes - procedure for checking if a node is element of a list
SYMBOL		- procedures for storing and recalling the name and description of the object including syntax check for the name
VAR	- procedures for checking and replacing arguments (variables) of the variable - procedure for evaluating the value of the variable	- procedures for storing and recalling the unit of the variable - mechanism for remembering previously calculated values
STATEVAR PROGVAR CONSTVAR REALLISTVAR VARLISTVAR FORMVAR		- procedures for saving and printing all data of the variable (declared in class FILEIO) - procedures for checking and replacing arguments (variables) of the variable (declared in class VAR) - procedure for evaluating the value of the variable (declared in class VAR) - procedure for loading all data of the variable - procedures for editing variable-specific data
PROC	- procedures for checking and replacing arguments (variables) of the process	
DYNPROC EQUPROC		- procedures for saving and printing all data of the process (declared in class FILEIO) - procedures for checking and replacing arguments (variables) of the process (declared in class PROC) - procedure for loading all data of the process - procedures for editing process-specific data
COMP	- procedures for checking and replacing arguments (variables, processes) of the compartment	- procedures for maintaining the list of active variables - procedures for maintaining the list of active processes - procedures for handling water inflow and substance input - procedures for handling initial conditions

Table 1—*contd.*

	- procedures formulating the differential-algebraic equations of time evolution of the spatially discretized compartment equations - procedures for calculating values characteristic for connection interfaces to links - procedures for evaluating calculated results	- procedures for calculating rates of active processes
MIXCOMP FILMCOMP RIVCOMP		- procedures for saving and printing all data of the compartment (declared in class FILEIO) - procedures for checking and replacing arguments (variables, processes) of the compartment (declared in class COMP) - procedures formulating the differential-algebraic equations of time evolution of the spatially discretized compartment equations (declared in class COMP) - procedures for calculating values characteristic for connection interfaces to links (declared in class COMP) - procedures for evaluating calculated results (declared in class COMP) - procedure for loading all data of the compartment - procedures for editing compartment-specific data
LINK	- procedures for checking and replacing arguments (variables, compartments) of the link	
ADVLINK DIFFLINK		- procedures for saving and printing all data of the link (declared in class FILEIO) - procedures for checking and replacing arguments (variables, compartments) of the link (declared in class LINK) - procedures for loading the link - precedures for editing link-specific data
BIF		- procedures for saving and printing all data of the bifurcation (declared in class FILEIO) - procedures for checking and replacing arguments (variables) of the bifurcation - procedure for loading all data of the bifurcation - procedures for editing bifurcation-specific data
SENS		- procedures for saving and printing all data of the sensitivity analysis definition (declared in class FILEIO) - procedure for loading all data of the sensitivity analysis definition - procedures for editing sensitivity analysis definition-specific data
FIT		- procedures for saving and printing all data of the parameter estimation definition (declared in class FILEIO) - procedures for checking and replacing arguments (variables, compartments) of the parameter estimation definition - procedure for loading all data of the parameter estimation definition - procedures for editing parameter estimation definition-specific data
PLOT		- procedures for saving and printing all data of the plot definition (declared in class FILEIO) - procedures for checking and replacing arguments (variables, compartments) of the plot definition - procedure for loading all data of the plot definition - procedures for editing plot definition-specific data

spatially discretized compartment equations. The data structures required for calculating these equations are completely hidden to the programmer using this interface to implement temporal evolution of the differential equations. As discussed later (polymorphism), this interface is universal to all currently implemented compartment types as well as to unknown future compartment types. Such a universality can only be achieved if a high degree of abstraction from underlying data structures is used for program design.

Abstract Data Types

The combination of data structures with operations manipulating the data enables information hiding to be employed at the class level, as described in the previous paragraph, and makes it possible to realize consistency checks and automatic initialization (construction) and destruction of data structures. This feature is extensively used in the realization of the AQUASIM classes shown in Fig. 5.

An example of the realization of consistency checks is given by the member function of the class SYMBOL designed for storing a new name of an object. This member function only accepts a new name, if it does not violate the syntax rules necessary to make parsing algebraic expressions possible (additional consistency checks for uniqueness of the name are necessary at the subsystem level). An example for the application of member functions for initialization and for avoiding operations corrupting system consistency is given for the class NODE: Nodes contain pointers to neighboring nodes. These pointers are set to zero for nodes which are not yet element of a list (by the constructor at the time of initialization and by the function removing a node from a list). The pointer to the previous node of the first list element, as well as the pointer to the next node of the last element, are set to the address of the node itself instead of zero, as is the conventional way of list realization (e.g. Wirth, 1986). Because the address of the node itself is identifiable and different from the addresses of all other nodes, this procedure also leads to a possible realization of a list. In addition to the conventional way, this realization allows an object derived from the class NODE to decide if it is already an element of a list (by checking if the address of the neighboring node is zero). This makes it possible to avoid insertion of a node, which is already an element of a list, into a list (an operation which would corrupt the first list). Furthermore, with this concept, it is possible to lock member functions that can lead to system inconsistency for objects, which are already elements of a list. As an example, it is not possible to change the name of an object derived from the class SYMBOL, if it is a member of a list. This locking mechanism of member functions facilitates maintaining the uniqueness of object names as described below. Objects of subsystems are edited as follows. With the aid of a copy constructor (realized for all classes derived from the class NODE), a copy of the object to be edited is generated. The copy constructor does not copy the pointers to the neighboring nodes, but it sets them to zero, so that this copy is not element of a list. For this reason, all editing operations can be applied to the copied node. After finishing the editing operation, the original object is replaced by the edited object. With this editing strategy, two goals can be achieved simultaneously:

- Consistency checks have only to be realized for the elementary editing operations of adding, replacing and deleting objects.
- It is very simple to realize cancellation also of multistage editing operations: Because changes have only been performed on a copy of the original object, this copy can be deleted instead of replacing the original object.

The features described above make program realization robust, because they help avoid programming errors.

Inheritance

Derivation of new classes from previously defined classes can be accomplished using inheritance. Inheritance permits the properties of the old classes to be used as defaults of the new classes. This method can substantially increase reuse of tested program code. Instead of programming the whole class from scratch, only additional features must be added. This approach can significantly shorten program development time.

Fig. 5 and Table 1 allow the reader to easily locate profit of the class hierarchy of AQUASIM: All methods realized in a non-end node class are inherited from derived classes. These methods belong to the following categories:

class FILEIO:	Procedures for loading, saving and printing elementary data types provided by this class guarantee usage of a standard file format by all classes shown in Fig. 5. An ASCII file format with line breaks guarantees compatibility of system files between platforms and makes it possible to transfer these files by electronic mail.
class NODE:	The list connection mechanism provided by this class is inherited from all derived classes. Therefore, all objects of derived classes can be inserted into dynamic lists. Furthermore, a member function of this class allows an object of a derived class to decide if it is element of a list. This member function can be used to lock member functions of derived classes in order to avoid the possibility that editing an object makes the system inconsistent, as described in the previous subsection.
class VAR:	This base class for variables provides procedures for storing and recalling the unit of a variable and for

remembering previously calculated values. Although the value of a variable has to be calculated differently for different variables (cf. next subsection on polymorphism), the mechanism for remembering old values, which increases program efficiency by avoiding unnecessary evaluations, can be realized universally at this level.

class `COMP`: Each compartment must maintain lists of active variables and processes, and store water inflow, substance input and initial conditions. Procedures for these tasks are universally implemented at this level for all compartment types. Because the set of active processes is known at this level, a procedure for calculating transformation rates can also be realized in this class.

All features listed above are realized for use in various derived classes and thus allow for reuse of program code.

Polymorphism

Polymorphism is by far the most important concept of object-oriented programming. It makes it possible to realize an extensible program structure by the declaration of methods in base classes, the implementation of which is different in different derived classes. Program sections that only use such dynamically bound methods can be written completely independent of the realization of these methods in derived classes. This yields an extensible program structure, because additional derived classes can be implemented later without the necessity of changing previously written program code.

The use of polymorphism in the model structure of AQUASIM can be located easily in Table 1: All methods in the column entitled "declared methods not yet realized at this level" are dynamically bound methods. They serve the following purposes:

class `FILEIO`: Dynamically bound methods for saving and printing objects of AQUASIM subsystems make it possible to realize procedures for saving and printing the whole AQUASIM system without knowledge of the structure of the objects saved or printed. No changes to these procedures are necessary, if new object types are added in later program versions (it is not possible to realize a similar mechanism for loading, because the type of an object read from the system file is not known to the program in advance).

classes `VAR`, `PROC`, `COMP`, `LINK`: Dynamically bound procedures of these classes for checking and replacing arguments (variables, processes or compartments) make it possible to realize type independent consistency checks and subsystem updates.

class `VAR`: The declaration of a member function for calculating the value of a variable at this level makes it possible to realize numerical procedures and functions for plotting and listing results in a variable type independent way. Addition of a new variable type only requires realization of the class corresponding to the new type (derived from the class VAR) in the core program and of its editing dialog box in the user interface layers (cf. Fig. 3). All numerical procedures and plotting and listing functions can remain unchanged.

class `COMP`: Because the realization of additional compartment types is a very important extension planned for future program versions, the application of polymorphism to the base class `COMP` of compartments is of special importance. Dynamically bound procedures for formulating the differential-algebraic system characterizing temporal evolution of the spatially discretized compartment equations make it possible to realize temporal integration (simulation), sensitivity analysis and parameter estimation in a compartment-type independent way. Due to the declaration of connection interfaces in this base class of compartments, links can be realized independent of compartment types. Finally, evaluation of calculated results is also realized compartment-type independently. Due to these features, addition of a new compartment type only requires implementation of the class corresponding to the new type (derived from the class `COMP`) in the core program and of its editing dialog box in the user interface layers (cf. Fig. 3). Procedures for the program functionalities simulation, sensitivity analysis and parameter estimation, realization of links connecting also the new compartment and processing of results need not be changed.

These examples clearly demonstrate the benefit of object-oriented programming techniques, especially of the use of polymorphism, for designing an extensible environmental identification and simulation program.

CONCLUSIONS

In this paper the techniques employed for the realization of a program for the identification and simulation of aquatic systems in the environment, in technical plants and in the laboratory are described (this paper does not describe program applications; examples of applications can be found in Reichert, 1994a; Reichert, 1994b; Wanner et al. 1994; Albrecht et al., 1995; Reichert et al., 1995; Wanner and Reichert, 1995). It is shown that the application of object-oriented program design techniques helps making the program more robust, clarifies program structure, shortens development time by reuse of inherited program code and leads to a program, which can better be extended to future needs. This makes it possible to realize a more universal program as compared to conventional environmental simulation programs. This program combines simulation with identifiability analysis, parameter estimation and uncertainty estimation. I hope that the publication of this example of the application of modern program design techniques to an identification and simulation program for aquatic systems encourages developers of environmental software to employ similar techniques in order to realize more universal simulation tools as it was usual until now.

ACKNOWLEDGMENTS

I would like to thank Jürg Ruchti for inspiring discussions on object-oriented programming, Georg Smehil and Werner Simon for useful comments to a preliminary version and Martin Omlin for his comments to the final version of the manuscript.

REFERENCES

Albrecht, A., Reichert, P., Beer, J. and Lück, A. (1995) Evaluation of the importance of reservoir sediments as sinks for reactor-derived radionuclides in riverine systems. *J. Environ. Radioactivity*, in press.

Ambrose, R.B., Jr. and Barnwell, T.O., Jr. (1989) Environmental Software at the U.S. Environmental Protection Agency's Center for Exposure Assessment Modeling. *Environ. Software* 4(2), 76-92.

Apiki, S. (1994) Paths to Platform Independence. *Byte*, 19(1), 172-178.

Budd, T. (1991) *An Introduction to Object-Oriented Programming*. Addison Wesley, Reading.

Carpenter, S. (1991) A GUI for all systems. *Byte*, 16(6), 144.

Carver, M.B., Stewart G.D., Blair J.M., and Selander, W.N. (1978) *The FORSIM VI simulation package for the automated solution of arbitrarily defined and partial and/or ordinary differential equation systems.* Report No. AECL-5821, Atomic Energy of Canada Ltd., Chalk River, Ontario, Canada.

Côté, R.G. (1992) Code on the Move, *Byte*, 17(7), 206-224.

Ellis, M. and Stroustrup, B. (1990) *The annotated C++ reference manual.* Addison-Wesley.

Gujer, W. and Wanner, O. (1989) Modelling Mixed Population Biofilms. in: Characklis, W.G. and Marshall, K.C., eds. *Biofilms*. John Wiley & Sons, New York.

Haribson, S.P. and Steele, G.L., Jr. (1991) *C - A Reference Manual.* Prentice Hall, Engelwood Cliffs, Third ed..

Jamshidi, M. and Herget, C.J. (1985) *Computer-Aided Control System Engineering*. North-Holland, Amsterdam.

Lippman, S.B. (1989) *C++ primer*. Addison-Wesley.

Marty, R. (1988) *Von der Subroutinentechnik zu Klassenhierarchien - Eine Schrittweise Hinführung zu objektorientierter Programmierung*. Institut für Informatik der Universität Zürich, Nr. 88.04, CH-8057 Zürich, Switzerland.

Meyer, B. (1990) *Objektorientierte Softwareentwicklung*. Hanser, München.

Müller, K.H. und Streker, I. (1984) *FORTRAN 77 - Programmieranleitung*. B.I. Hochschultaschenbücher, Mannheim.

Popper, K. (1982) *Logik der Forschung*. 7th edition, J.C.B. Mohr, Tübingen (first edition 1934).

Schiesser, W.E. (1991) *The Numerical Method of Lines Integration of Partial Differential Equations*. Academic Press, San Diego.

Reichert, P. (1994a) AQUASIM - A tool for simulation and data analysis of aquatic systems. *Water Sci. Technol.* 30(2), 21-30.

Reichert, P. (1994b) *Concepts underlying a computer program for the identification and simulation of aquatic systems*, Schriftenreihe der EAWAG Nr. 7, Swiss Federal Institute for Environmental Science and Technology (EAWAG), CH-8600 Dübendorf, Switzerland.

Reichert, P., von Schulthess, R. and Wild, D. (1995) The use of AQUASIM for estimating parameters of activated sludge models. *Water Sci. Technol.* 31(2), 135-147.

Rochkind, M.J. (1989) *Technical overview of the extensible virtual toolkit (XVT)*. XVT Software Inc., Box 18750, Boulder, CO 80308, USA.

Wanner, O., Debus, O. and Reichert, P. (1994) Modelling the spatial distribution of a xylene-degrading microbial population in a membrane-bound biofilm. *Water Sci. Technol.* 29 (10-11), 243-251.

Wanner, O. and Reichert, P. (1995) Mathematical Modeling of Mixed Culture Biofilms. *Biotechnology & Bioengineering*, in press.

Wirth, N. (1985) *Programmieren in Modula-2*. Springer Verlag, Berlin.

Wirth, N. (1986) *Algorithmen und Datenstrukturen mit Modula-2*. Teubner, Stuttgart, 4. Auflage.

Yen, B.C. (1973) Open-channel flow equations revisited. *J. Eng. Mech. Div. ASCE*, 99, 979-1009.

Yen, B.C. (1979) Unsteady flow mathematical modeling techniques. Chapter 13 of Shen, H.W., ed. *Modeling of Rivers*. John Wiley, New York.