Object-Oriented Ecosystem Modelling

A Case Study: SALMO-OO

A thesis submitted for the degree of Master of Science

Byron He Zhang
Discipline of Environmental Biology
School of Earth and Environmental Sciences
The University of Adelaide

January 2006

Abstract

Object-oriented ecosystem modelling was introduced in the early of 1990s (Silvert, 1992). From that time on, ecosystem models using object-oriented programming (OOP) has earned significant achievements with increasing upgraded information technology. The common purposes of ecosystem modellers are to build a model with flexible structure, which allow continuous modifications on the model content. In last decade, ecosystem modellers have put a large number of efforts to practice the OOP approaches in order to implement a true object-oriented ecosystem model. However, these previous work have not fully take advantage of object-orientation because of misusing more or less this technique. This paper explains the shortcoming of these previous endeavours therewith points out a practical solution that using the methodology of object-oriented software engineering and some relative novel information techniques. A case study SALMO-OO will be presented in this paper to prove Silvert's assumption that OOP play an important role on ecosystem modelling approaches. Moreover, the results of SALMO-OO convince that object-oriented ecosystem modelling can be achieved by using object-oriented software engineering associating with a true object-oriented programming language (Java in this case).

Statement of Originality

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of my thesis, when deposited in the University Libraries, being available for photocopying and loan.

Byron Zhang

Acknowledgements

I would like to thank my supervisors, Prof. Friedrich Rechnagel and Dr. HongQing Cao. Friedrich, my principal supervisor, offers this wonderful opportunity for me to engage in the interdisciplinary project between ecology and software engineering. His instruction benefits me to deepen my understanding of the difference and similarity between the scientific research and the software development. Also he has spent lots of precious time to teach me the knowledge of freshwater ecology with his advice, knowledge, even patience. Likewise, HongQing, my co-supervisor, reinforce my knowledge in software engineering area, and she instructs me one of her intelligent Genetic Algorithms, which make me feel that I have more strength for my future career.

This work would not have been done so quickly without the collaborations of Ms. Lydia Cetin. As my key colleague, she has prepared most of the materials that involve in the software development, and sometimes plays a role during I type source code as a peer programmer, especially thanks for her proofreading my thesis.

I also would like to thank my IBP teacher, Ms. Margaret Cargill. She helps me go through my first taste of writing a scientific paper.

Finally, I must thank my parents, QingZhi and DeJun, for their love, support, and financial aid. None of this work would have been possible without their continuous efforts on me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ecological modelling is a process to represent and simulate natural ecosystems by means of computable methods, such as ordinary differential equations (ODE). Essentially, ecological modelling consists of two elements. One is an ecosystem model that can be represented as an ecosystem class library. Another is a group of ecological data that can be represented by an ecological database. A class library is defined as a set of ready-made software routines (class definitions) that programmers use for writing object-oriented programs (Schach 2002b). An ecosystem class library therefore means a group of computer algorithms that implement the equations of an ecosystem model and are assembled in a computer program. The ecological database stands for information technology for archiving and retrieval of manageable ecological data.

Ecologists benefit from both ecological modelling and novel information technology. Computer systems provide a platform for ecologists to create virtual ecosystems determined by process-based ODE and driven by complex ecological data. Object-oriented design and implementation of ecosystem class libraries is a novel information technology that significantly improves the transparency and flexibility of complex ecological models. Object-oriented software design and implementation have been introduced in the early 1980s and since then broadly applied for commercial and scientific uses (Schach 2002b).

Lake ecosystems are one application area of ecological modelling. Lake ecosystem models need to reflect the basic structure of the pelagic food web and nutrient cycle in order to realistically simulate phytoplankton and zooplankton dynamics in response to seasons and water quality changes. The lake ecosystem model SALMO (Recknagel and Benndorf 1982, Recknagel 1989, Recknagel et al. 1995) meets well these requirements being based on ODE for three functional groups of phytoplankton (diatoms, green algae, blue green algae), herbivorous zooplankton (cladocereans), dissolved inorganic phosphate (DIP), dissolved inorganic nitrate (DIN), detritus, and dissolved oxygen. Through continuous improvements over the past decade, this model is now generic for non-shallow lakes (maximum depth > 5m) and able to simulate daily dynamics of phytoplankton, zooplankton, oxygen, phosphates, nitrate, and detritus concentration in response to lake-specific data of water temperature, solar radiation and nutrient loadings measured for one particular year. The model is a useful tool for scenario analysis and allows forecasting of lake ecosystem responses to management scenarios such as artificial mixing, bio-manipulation and external nutrient control. The pervious model class library of SALMO was designed and implemented in a modular but rigid structure by means of the programming language Fortran IV, which does not support the object-oriented paradigm but allows structured programming. Therefore the functionality of the previous SALMO program was limited in flexibility, user friendliness, and web accessibility. To overcome these limitations, this research aimed at the object-oriented design and implementation of the model SALMO by means of the programming language Java, and to establish a prototype for the object-oriented design and implementations of complex ecological models.

1.1 Literature Review

1.1.1 Introduction of Object-Oriented History

Since the first object-oriented concept and Simula were introduced in 1960s (Dahl and Nygaard 1966), the object-oriented programming language (OOPL) family probably consists of more than 30 members by now. The popular OOPLs could be Simula, Smalltalk-80 (Goldberg and Robson 1983), Eiffel (Meyer 1992), C++ (Stroustrup 1986), Java (Sun Microsystems 1995-2005), and C# (Microsoft 2006a).

These OOPLs are developed based on the fundamental units of object-orientation: object, class, inheritance, encapsulation, and polymorphism (Abadi and Cardelli 1998). These basic concepts are well established and widely cited in the literature. The most accurate definition of an object is a software bundle of variables and related methods (Sun Microsystems 1995-2005). A class is an abstraction data type that represents the instances of all objects (Pugh et al. 1987). In other words, a class is a concept that abstracts a group of relative objects. For example, mankind covers all of the human beings while an individual can be regarded as an instance of the mankind. In this context, a concrete individual is the object of the mankind class. In case of aquatic ecology, phytoplankton is a class that abstracts various algae. Diatoms, green algae, blue green algae are the objects of the phytoplankton class respectively. Inheritance is defined as a subclass that extends the details in its superclass. This idea dates back to the creation of biological taxonomy, which is a example of how computer science is inspired by biology. In the above case, the mankind class comprises two subclasses: the female and the male class. Similarly, the phytoplankton class can descend from an algae species class that extends the growth behaviour. Thus, the extended algae class is the subclass of the phytoplankton class. Logically, the phytoplankton class is the superclass of the algae class. There are inheritance relationship between the phytoplankton superclass and the algae subclass. Further discussion can be found in (Meyer 1996). The idea of encapsulation was introduced as information hiding (Parnas 1971). As the name implies, encapsulation or information hiding allows objects to protect their details from illogical access from other objects. Specifically, an object can hide its own information, including fields (states) and methods (behaviours). For example, a phytoplankton class contains a biomass field, a growth and a grazing method. Another zooplankton class contains a biomass field and a growth method. Obviously, the phytoplankton biomass is different from the zooplankton one. Encapsulation makes these two different biological biomass fields be invisible to each other. A zooplankton object grazes an algae object, in this context, can be explained as a zooplankton object increases its biomass field value by the zooplankton growth method as the same time as an phytoplankton object decreases its biomass field value by the phytoplankton grazing method. Notes that neither a zooplankton object nor a phytoplankton object can directly access others' fields or methods. They have to send and receive a message in order to tell a mutual object to complete this logic. Polymorphism means that programming language of a variable and function have more one type (Cardelli and Wegner 1985). Polymorphism usually accompanies with inheritance. For instance, a superclass has three subclasses. These three subclasses inherit a variable of the superclass but each of this subclass has different variable types and the exactly same variable name. The object of the superclass is able to use the appropriate variable with the desired type during runtime. This mechanism is polymorphism. It is extreme useful in ecological modelling. For example, in the above-mentioned case, a phytoplankton object can be cast as its subclass type such as an algae type. For a concern condition, a model simulation will be automatically informed which algae type needs to be activated. Therefore, the model will run the approximate object methods in the runtime. The extended algae growth model libraries provide either from the original growth model library or these methods. Without exception, polymorphism is one of natural world concepts utilised in computer science.

As far as the motivation of creating OOPLs is concerned, the original intention is to create a new programming language in order to overcome some previous programming problems such as maintainability and readability. It is well known that a software product is made of code. The conventional programming can be described as the larger the software product the more mass code is required. As a result, software developers suffer the difficulty of maintenance and understand the code when the software product is large enough to consist of long list codes.

The solution using object-oriented paradigm simply breaks the code into a number of relative pieces. Each piece is organized in the smallest extent to represent a single object, including its state and behaviour. For example, a car object can represent a Ford sedan. This car object may comprise a number of states such as its colour and the number of wheels, and behaviours such as starting, braking, and parking. Also in this case, a car class represents all cars. Obviously, a Ford sedan is one of the instances of cars. A well-designed class is highly reusable, which can communicate with other objects by sending and receiving message without any modification in all the different special applications. Just like a Ford sedan can be simply used if lending to other drivers.

In recent years, the two popular OOPLs are Java and C#, which are made from private computer companies Sun Microsystems and Microsoft respectively. Java programming language not only is supported by Sun Java group but also is technological shared in open source communities. As the most popular, object-oriented, and free OOPL, Java widespread applies in various scientific research projects besides commercial applications.

1.1.2 Previous Efforts of Object-Oriented Ecological Modelling

Many efforts have been undertaken to improve ecological modelling by using object-oriented technology.

In the early of 1990s, biologists had attempted to consider object-oriented technology as a novel method for ecosystem modelling. The idea of using object-oriented programming (OOP) for ecosystem modelling was introduced by a Canadian biologist (Silvert 1992). From biologists' perspectives, the object-oriented concept is to focus on the abstract representation of real objects in the natural worlds instead of the linear sequence of calculations in some ecosystem model programs. In this paper, Silvert started from a question whether the model structure should reflect the ecosystem structure. Subsequently, the detail introductions about object-oriented programming have been discussed, including the concept of objects, inheritance, interactions, even the way to write code and maintenance. For example, an object-oriented programming language code is listed in the below code fragment.

```
Algae = object(Plant)
        Biomass;
        procedure Growth();
        procedure Grazing();
end;
```

This code fragment describes an Algae object that extends the Plant object. Inside the Algae object, it comprises a Biomass state and the Growth and Grazing behaviour. Obviously, this code fragment reflects the basic biological processes of algae species. Firstly, it defines an Algae object is one of the sub-objects of the Plant object, which represents the inheritance relationship. Secondly, it briefly lists the process of algae life cycle: growth and death. The algae increases biomass value that is represented by the growth behaviour and the value are decreased because of grazing by herbivorous zooplankton. Besides the various code samples, this paper discusses comprehensively the details behind these samples of code in order to instruct other practisers how to programme object-oriented ecosystem models.

Also, this paper pointed out the natural compatibility between some object-oriented concepts and biology. For example, the inheritance of object-orientation originates from biological literature. Under these merits, the author believed that OOP could implement ecosystem models with highly flexibility and maintainability. Therefore, OOP would become a dominated methodology in implementing ecosystem models in the future if there will appear some effective object-oriented programming language.

This assumption not only indicates Silvert's anticipation but also implies the infeasibility of using object-oriented programming languages in developing large-scale ecosystem models at that age. Indeed, object-oriented languages were not ready to be applied until a few years later. The unfortunately previous disadvantages of OOP such as low level of optimisation, difficulty to integration, and inefficiency results in delaying widespread applications of object-oriented ecosystem models. However, Silvert foresaw the perspectives that OOP would be an important approach in ecosystem modelling. The history of object-oriented ecosystem modelling proves Silvert's undoubted prediction.

Another published paper (Sequeira et al. 1991) even further discover the object-oriented simulation in plant applications. This paper introduces a case study to model the plant growth objects by object-oriented paradigm. In addition to the abundant literatures of object-oriented introduction, a specific application discussed that illustrates the way how object-oriented simulation benefits the applications of cotton growth and development. Even the authors of this paper adopt diagrams to represent the design documents in order to clarify the differences between object-oriented and procedural approaches. This outstanding work not only discusses the usage of object-oriented programming approaches for ecological modelling in theory but also illustrates a concrete case study. These outcomes will provide a typical sample for other biologists and modellers although the different ecosystem types.

With rapidly innovation of information technology after mid-1990s, object-oriented technology has been widely applied to terrestrial ecosystems focusing on plant growth models (Acock and Reddy 1997, Chen and Reynolds 1997, Lemmon and Chuk 1997) and, to a lesser extent, to aquatic ecosystems focusing on freshwater food webs and nutrient cycles (Ferreira 1994). Undoubtedly, these previous studies have stimulated both concepts for ecological modelling and ecosystem research in general. However, these efforts demonstrated neither clearly the advantage of applying object-oriented technology to ecosystem models, nor convincingly the application of object-oriented technology to transparently structure complex ecosystem models. Thus, there is a demand to promote both the improved functionality of ecosystem models and the best practice of implementing ecosystem models by object-oriented programming.

The following sections will review previous efforts on ecological modelling using object-oriented technology in terms of their possibilities and shortcomings. Subsequently, I will discuss how the complexity of ecosystems has influenced the choice of modelling methods. The final section will introduce the model SALMO, as well as discuss improvement of SALMO in particular and ecological models in general by means of OOP.

The previous studies on object-oriented programming of ecological models resulted in a few different object-oriented models depending on the degree of achieving both an object-oriented design and implementation. Most of the previous studies focused either on object-oriented design or implementation but failed to achieve both.

1.1.3 Ecosystem Models Using Object-Oriented Design

Some ecosystem models were designed according to the object-oriented paradigm but implemented by procedural or hybrid programming languages such as Fortran, Pascal or C++. For example, FEMIME (Soetaert et al. 2002) was intended to construct an environment for mathematically modelling with reusability, flexibility, and maintainability by using object-oriented design. It separates deterministic ecosystem models into three parts: formulation, numerical solution, and application, which benefits ecologists or modellers to reuse the various model libraries as well as simplify model libraries design. Moreover, FEMIME provides a general platform for numerical solution selection and data interaction. The author of this paper claims that FEMIME has generic performance in any deterministic, complex, object-oriented ecosystem model implementation. However, this paper neither contains any clear diagram to represent the structure of FEMIME, nor programmes it in object-oriented language. It is hard to be convinced that FEMIME fully adopts object-oriented design without displaying any well-designed structure diagram. As far as the usage of programming languages is concerned, FEMIME uses one of the procedural programming languages Fortran, which clearly indicated by the discussion of using 13 routines and programming in Fortran. These evidences show that FEMIME was built based on object-oriented design and implemented in structural procedural language. The limitation of Fortran is because it allows only procedural programming so that cannot take advantage of object-oriented programming approaches, and cannot link models to the Internet. It is clear that FEMIME needs to be implemented in an object-oriented programming language in the future work. In addition, the generality of FEMIME is limited in Fortran-based ecosystem models. A highly reusable model library inside model is beyond the scope of FEMIME because it only focuses only the high-level process of modelling. Therefore, it remains a big gap that needs to be filled in when ecologists or modellers expect more details in ecosystem model implementation such as how to programme an Algae code object.

Other examples such as ECOWIN (Ferreira 1994) successful delivers a well-designed aquatic ecosystem model. It is based on object-oriented paradigm but implemented in Pascal. The most exciting outcome of ECOWIN is a descendent aquatic ecosystem class diagram, which is showed in Figure 1.1, an aquatic ecosystem object inheritance hierarchy by means of object-oriented design. There are four descendent levels in ECOWIN class diagram. Each descendent level class inherit the super-level class, which automatically reuse its public attributes and methods. For example, the Phytoplankton and Phytobenthos object share a common method named Production that belongs to the Producers object. Through benefiting from object-orientation, ECOWIN earns great advantages in maintainability and adaptability. According to its conclusion, ECOWIN can be applied in various environmental affairs such as metal pollution and red tides.

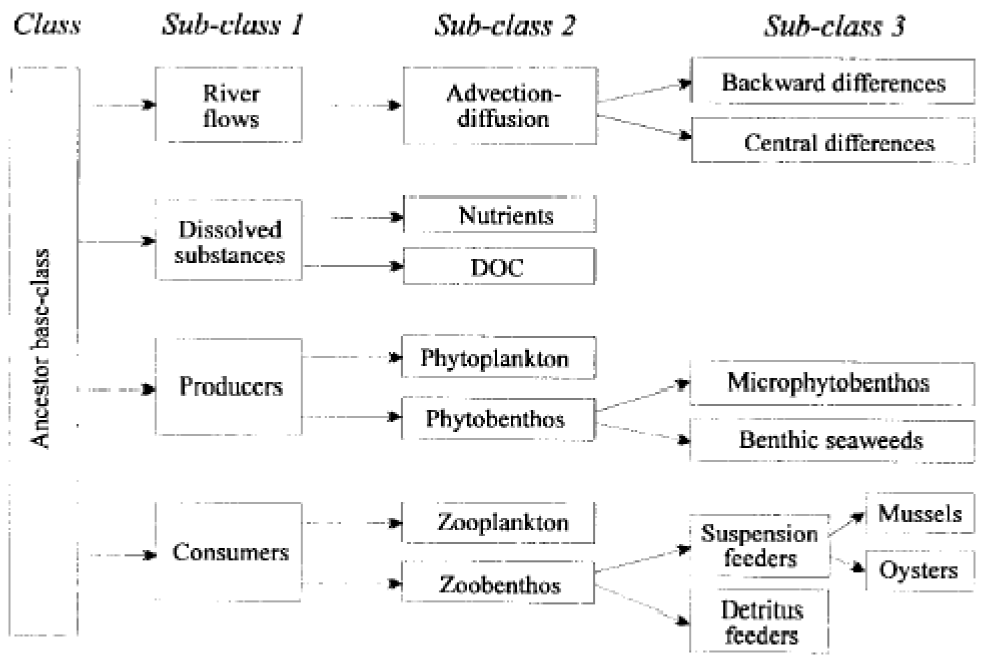


Figure 1.1 An aquatic ecosystem class diagram (Ferreira 1994)

Although ECOWIN takes fully advantages of object-oriented design, one drawback cannot be ignored. ECOWIN is implemented in Pascal family programming language (Turbo Pascal for Windows). In computer science context, Turbo Pascal is not completed object-oriented programming language (McMillan and Collins 1990). Alternatively, ECOWIN can use Delphi, which provides an object-oriented Pascal environment (Gofen 2001). The suggestion to use object-oriented Pascal will benefit ECOWIN take fully advantages of object-orientation in the future work.

Another case is a cotton crop model in C++, which called Cotton++ (Lemmon and Chuk 1997). Cotton++ is designed for agriculture applications and implemented in C++ programming language. This model is the part of a Decision Supporting Systems (DSS) for crop management. As far as C++-based models are concerned, the ecological modellers have to take limitations and problems of the C++ programming language such as memory leaking into account (Heine and Lam 2003). Even some computer scientists doubt that C++ is a true object-oriented language (Jenkins and Hardman 2004) and detail discussions can be found in (Stroustrup 1995), which discusses why C++ is not object-oriented programming language by its creator. The fact is C++ adopt the characters of both object-oriented and procedural programming languages, which can be defined as a hybrid programming language. In addition, the author of Cotton++ has done some interesting comparisons between C++ and Smalltalk. The conclusion is using C++ rather than Smalltalk because of the difficulties to use Smalltalk in numerical analysis.

Although these previous studies focused on different ecosystem types and problems, they all emphasized on the following aspects: (1) the introduction of the nature and benefits of object-oriented technology; (2) the methodology and process of implementing ecosystem models using object-oriented technology or programming language; (3) the design of a software application in order to facilitate the simulation of ecosystems. All these studies provided reasonable background knowledge on the ecosystem type and model purpose. However, most of these studies failed to thoroughly and properly apply object-oriented technology even though paradigms of object-oriented design were promoted. This is a crucial limitation of the previous applications that this research is intended to overcome.

1.1.4 The Object-Oriented Ecosystem Models

The object-oriented ecosystem models require not only object-oriented design but also using an object-oriented programming language. For example, SSEM is designed for shallow-sea fisheries and using Smalltalk (Sekine et al. 1991). The outcomes of this model prove that object-orientation benefit the flexibility of the application. Also, SSEM stores the ecological data into a database instead of the plain text file in the previous applications. After 1997, a new programming language Java is introduced in programming language family. As a result, Java-based ecological modelling earns widespread attractions. For example, the model Eclpss adopted the Java technology to build an ecological component library for parallel spatial simulation (Wenderholm 2003). The model Kraalingen for crops (Papajorgji et al. 2004) was implemented by Java and is associated with the Unified Modelling Language (UML) (Object Management Group 1997-2005), a type of modelling language to design software applications. These efforts successfully delivered some software applications with good performance and such desired attributes as being highly reusable, maintainable, scalable, and portable. Object-oriented programming such as Java for object-oriented modelling facilitates computer-based ecosystem simulation at an advanced level.

In conclusion, these previous efforts on ecological modelling attract increasing applications in various ecological areas. Regardless of using any object-oriented programming languages, ecologists and modellers have already been convinced that ecological modelling can take advantage of object-orientation. Unfortunately, in the case of using procedural or hybrid object-oriented programming language, it is difficult to protect ecosystem models from inflexible model structure. Thus, it is meaningless for ecosystem modellers to pursue the comparison between computer programming languages before attempting to implement an ideal ecosystem model. In fact, there is no benchmark for guiding ecosystem modellers to practically select and apply one object-oriented programming language to the model implementation. Detail discussions about this issue would beyond this paper’s focus. Readers may find more in (Ryder and Burnett 2005). Therefore, I would suggest that ecosystem modellers regard the implementations of ecosystem models as one of the software applications in software engineering domain. Logically, the solutions to achieve object-oriented ecosystem models are using any object-oriented programming language and methodology of object-oriented software engineering. To achieve this gap, the future ecosystem modelling strongly suggests producing comprehensive modelling documents, including the design, the class library reference, and the source code if it is open to public. Ecologists or modellers are encouraged to further introduce the methodology of software engineering into the development of ecosystem models in order to achieve a global standard ecological modelling society in the future.

1.2 Ecological Modelling vs. The Complexity of ecosystems

The complexity of ecosystems is still a big challenge when ecologists attempt to realize various ecosystem models in scientific or practical applications. User-friendly implementation and maintenance of ecological models can be quite complicated, time-consuming and costly.

From the ecological modellers’ point of view, the complexity of ecosystems comes from the non-linear interactions between abiotic and biotic components as well as the stochastic nature of ecosystem that need to be taken into account by the ecological data, mathematical equations and computer programs. Usually, modifications of the program structure can be very time-consuming when an ecosystem model needs to be changed or upgraded if the program is not designed appropriately. For example, it can be expected that future applications of SALMO need to add components such as fish in order to extend the functionality of SALMO. This would currently mean that the Fortran IV source code for the mathematical calculations, the data import and the subroutines invoked have to be modified. Thus, ecological modellers suffer from the maintenance of ecosystem models if they fail to properly structure and hence simplify the complexity of ecosystem models in the early stage of design and programming. Obviously, neither the semi-object-oriented programming languages (e.g. C++ language) are able to suit this purpose, nor the ordinary programming languages (e.g. C or Fortran language). Thus, it is necessary to investigate how to consider and handle the complexity of ecosystems by means of object-oriented ecological modelling.

There are some obvious shortcomings that the Fortran IV version of the model SALMO does not resolve. The lack of flexibility is one of the most significant shortcomings in terms of a SALMO class library even though the model SALMO is designed in a modular structure. In this context, the term class library reflects a group of flexible and transparent computer algorithms. Unlike the normal computer programs, the class library has a well-designed structure. Even though individual algorithms are strongly cohesive the relationships between the algorithms are loosely coupled. The complexity of ecosystems causes the difficulty to pursue this intention. There are nine state variables in the model SALMO, which represent nutrient cycles and food web interactions in lakes. As a result, the modellers have to consider how to keep a high degree of cohesion in every individual state variable while lowering the degree of couplings among these state variables in order to make the whole class library highly flexible. The programming language Fortran does not offer object-oriented concepts (Class, Object, Field, Method, Encapsulation, Inheritance, and Polymorphism) but facilitated a modularly structured version of SALMO properly executing logical and mathematical operations. The Fortran version of the model SALMO cannot easily adapt to various data logic as applications shift from one lake to another lake. In contrast, the object-oriented paradigm considers both specifically mathematical calculation logic and data logic. By means of abstraction, instantiation, inheritance, overload, overwrite and other operations, some object-oriented design methodologies and programming languages can achieve the highly flexible class library. Object-oriented technology represents these two logics respectively by method and field, which are two basic elements in a class. For example, the Zooplankton class contains a common attribute of biomass that needs to store these values, so a field ‘valueOfBiomass’ in the zooplankton class represents the attribute of biomass. Similarly, a method represents the logic that calculates the value of biomass (e.g. here called ‘getBiomass’). Once the zooplankton class is created, it can communicate with other classes via the zooplankton object (an instantiation of the Zooplankton class). By means of similar simulations for all abiotic and biotic state variables over time, the model SALMO realistically mimics nutrient cycles and food web dynamics of natural lake ecosystem in the computer. All the functionalities that belong to a particular state variable will be contained in a class (the class name usually is the same as the state variable), which refers to the purpose of strong cohesion. On the other hand, the parameter transfer represents the interactions amongst these state variables. It would meet the criteria of loose coupling if these parameters were kept to a minimum. Thus, object-oriented technology is an ideal solution that simplifies the complexity of ecosystem modelling.

Another shortcoming of the Fortran version of the model SALMO is the lack of friendly user interface and web accessibility. It is undoubted that object-oriented technology plays an important role in pursuing these purposes, and fortunately the fundamentals are not too much different from the implementation of flexibility. Technologically, most of the details are transparent to the ecological modellers. The current information technology enables any computer model to have friendly user interfaces. Even most non-object-oriented technologies can achieve friendly user interfaces and web accessibility, except the Fortran programming language.

The previous studies clearly indicated that object-oriented technology provides great potential for ecological modelling, assisting in handling and unravelling the enormous complexity of ecosystems comparing to traditional programming methods. However, much work remains to be done to fully utilise and implement the functionality of object-oriented technology. It appears that object-oriented technology was not well interpreted and therefore not fully utilised in previous ecological applications. In particular there are not many examples of true-object-oriented applications to lake ecosystems. Therefore, this research aims to exemplarily organize the complexity of lake ecosystem models by means of object-oriented technology and demonstrated by means of SALMO.

1.3 The Description of SALMO

The model SALMO (Recknagel and Benndorf 1982; Recknagel 1989) is designed as a generic lake ecosystem model. The term generic means that the ODE of the model represent key ecological processes in order to determine mass balances of inorganic nutrients, detritus and oxygen as well as seasonally changing biomasses of phytoplankton and zooplankton. Also the ODE of the model require explicit data of lake and year specific driving variables such as water temperature, solar radiation, mixing depth and nutrient loadings, which facilitate the computer simulation of a broad range of lakes and reservoirs with diverse morphometry, water quality and climate conditions. For example, the model has successfully been applied to simulate lakes in the range from oligotrophic to hypertrophic conditions, as well as inversely stratified temperate lakes with ice cover in winter and temperate to Mediterranean lakes with thermal stratification in summer (Recknagel 1989). The model SALMO gains its flexibility to changing environmental and climate conditions by both causally determined process equations and measured input data of physical and chemical driving variables. However, the so achieved generic properties of the model have their price by causing high complexity of the ODE. Highly complex ODE are typical for process-based deterministic ecosystem models and often affect the model transparency, programming, implementation and maintenance. These factors often impact on user friendliness and acceptance of the models. To overcome such negative impacts of model complexity in the past, structured modular programming by conventional programming languages such as FORTRAN IV was applied. The concept of object-oriented programming and implementation by recent programming languages such as Java provides advanced functionality for programming and implementing complex models that become more transparent, user-friendly and open for development.

1.3.1 The Structure of The model SALMO

Figure 1.1 displays the simplified structure of the ODE of SALMO. The rectangles symbolise the actual mass balance of the state variables such as phosphate P or detritus D, which represent the left hand parts of an ODE. The circles symbolise the actual rates variables of process equations such as growth or mortality, which determine the changes of state variables. The curved arrows represent the causal relationships between the state variables and the rate variables. The straight arrows indicate the direction of the mass flow either gaining mass from a source (e.g. algal growth) or loosing mass to a sink (e.g. algal grazing). The ODE of SALMO calculates for measured physical and chemical input data the resulting output data for the state variables. The Figure 1.2 shows the input and output variables that are typically processed by the ODE of SALMO.

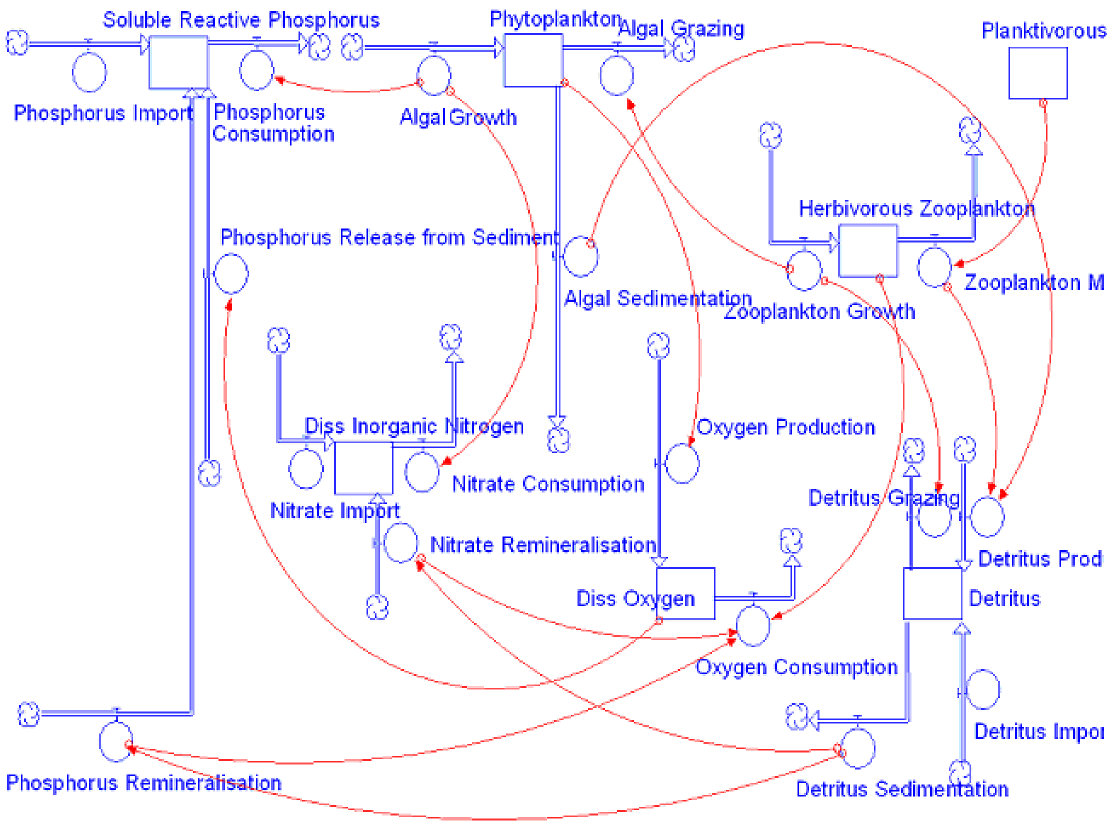


Figure 1.1 The structure of the ODE of SALMO (after Recknagel 1989)

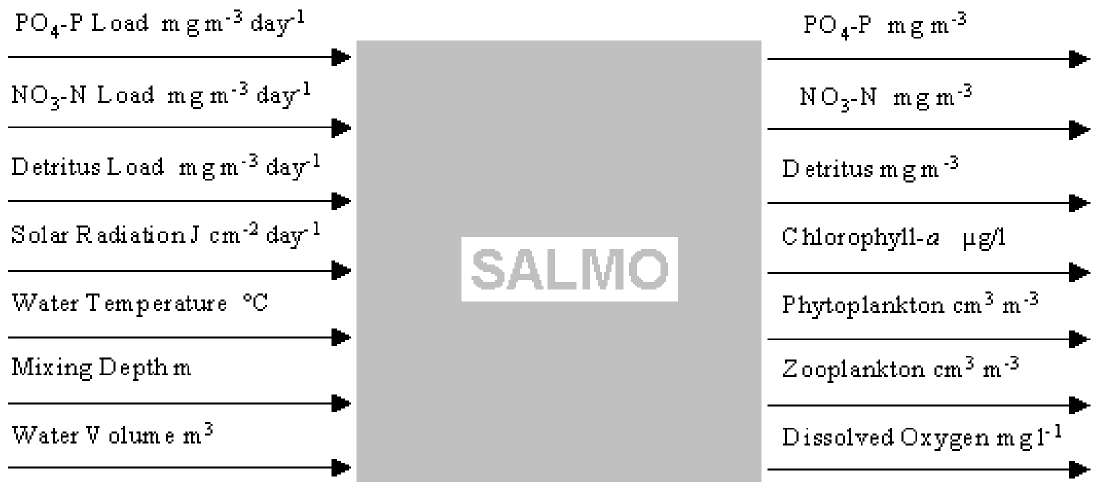

PO_4-P Load mg m^{-3} day^{-1}
NO_3-N Load mg m^{-3} day^{-1}
Detritus Load mg m^{-3} day^{-1}
Solar Radiation J cm^{-2} day^{-1}
Water Temperature °C
Mixing Depth m
Water Volume m^3
SALMO
PO_4-P mg m^{-3}
NO_3-N mg m^{-3}
Detritus mg m^{-3}
Chlorophyll-*a* µg/l
Phytoplankton cm^3 m^{-3}
Zooplankton cm^3 m^{-3}
Dissolved Oxygen mg l^{-1}

Figure 1.2 The input and output variables of the model SALMO

The ODE of the model SALMO considers an additional 128 constant parameters that were defined and specified for the numerous process equations (Recknagel and

Benndorf 1982). These parameter values also proved to be generic for lakes under temperate and Mediterranean climate.

In order to be applicable to thermally stratified lakes in summer the model has been designed to temporarily simulate the epilimnion and hypolimnion of a lake model by separately calculating all ODE for the two layers.

All ODE of SALMO are calculated with a daily time step for 360 days per year. The daily simulations start with calculations of the mean underwater light intensity over the water volume from the photosynthetic active solar radiation at the lake surface by considering light extinction over depth using Lambert-Beers-Law, and the concentrations of phosphate and nitrate. The three ODEs for phytoplankton are applied to diatoms, green algae and blue green algae whereby algal growth is calculated as the difference of photosynthesis and respiration. Photosynthesis is limited by underwater light, water temperature and concentration of phosphate and nitrate. Phytoplankton grazing by zooplankton is considered as a key ecological process determining loss of algal biomass. By programming the daily calculations of the ODE and using daily data of input variables, the computer simulation of food web dynamics and nutrient cycles of lakes by SALMO can be conducted.

1.3.2 Possible Improvements Of Lake Ecosystem Modelling In General And The Model SALMO In Particular By Means Of Object-Oriented Programming

Lake ecosystem modelling using object-oriented programming can facilitate model and data sharing through the Internet. Provided that a generic ecosystem model is available that is applicable to different climate and water quality condition, its access by Internet makes it a global model that can be remotely run by users around the world. The resulting promotion and sharing of modelling research results benefits both the model developers and users.

Object-oriented programming allows making a basic lake ecosystem class library highly reusable. The class library can be designed to consist of classes, fields and methods. Classes are the smallest units of the ecosystem and represent individual plant or animal species. Fields contain the attributes that describe a class. Methods provide information on the ecological behaviour of a class. The application of the general categories ‘class’, ‘field’ and ‘method’ allows classifying, standardising and documenting complex ecological knowledge in a way that is understandable by both ecologists and computer scientists.

The lake ecosystem model SALMO fulfils the requirements of a generic ecosystem model and promises a useful case study for object-oriented programming and implementation by means of Java. The application of object-oriented paradigms of Java to SALMO require the design of a new program structure and functioning compared to the previous program version by means of the conventional programming language FORTRAN IV. As a result SALMO will gain a more transparent, modular and flexible source code open for further development and can be remotely run through the Internet. As there is currently no lake ecosystem model known to be accessible by Internet, the Java version of SALMO called SALMO-OO could become the first global lake ecosystem model.

The present research will be a contribution to the development of a prospective global lake ecosystem model SALMO-OO. The achievement will focus on a flexible, reusable and portable lake ecosystem model by using object-oriented programming and largely open source tools.

Chapter 2

Proposal: Objectives, Hypotheses and Expected Outcomes

The objective of this research is to contribute to an ecological information system (EIS) that consists of a Java runtime environment (JRE), class libraries, database management system (DBMS), and documents (see Figure 3.1). The implementation of the class libraries, DBMS, and documents are going to be completed by this research. As far as the model SALMO is concerned, the object-oriented version SALMO-OO will be developed by means of Java and imbedded in the EIS (see Fig. 3.1).

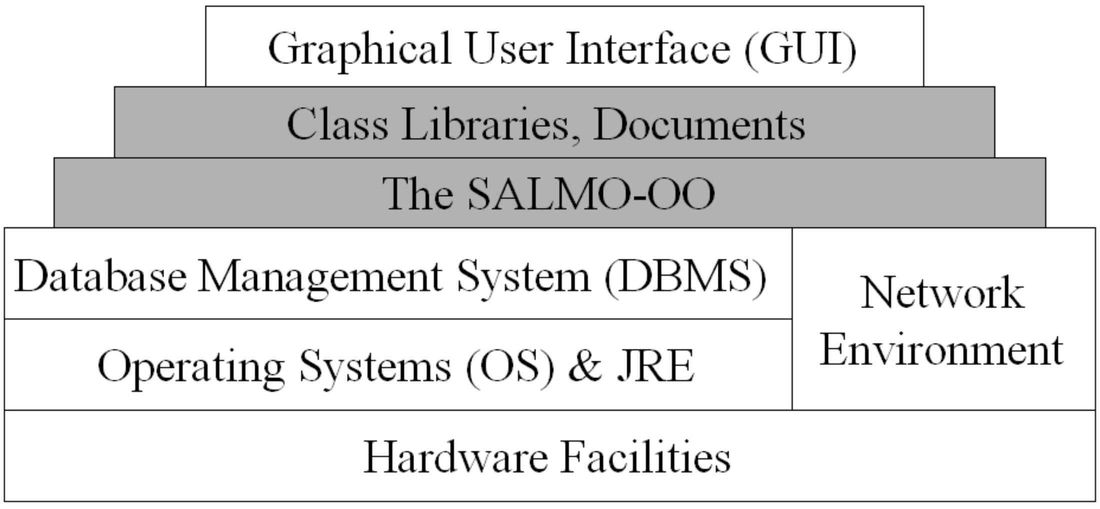

Graphical User Interface (GUI)
Class Libraries, Documents
The SALMO-OO
Database Management System (DBMS)
Network Environment
Operating Systems (OS) & JRE
Hardware Facilities

Figure 2.1 The components of the EIS

The following hypotheses will be tested during this study:

1. A fully functioning object-oriented version SALMO-OO of the model SALMO can be programmed and implemented by Java

2. A user-friendly graphical user interface (GUI) allows users to access SALMO-OO via Internet

3. The object-oriented design, programming and implementation of SALMO-OO by Java, and allows establishing a library of alternative process models such as for algal growth, algal grazing, zooplankton growth or zooplankton mortality.

The expected outcomes of the proposed application of object-oriented programming to ecosystem models can be summarised as follows:

- Providing transparent program structures of complex ecosystem models by means of object-oriented programming and implementation in Java.

- Extending the structure and functioning of complex ecosystem models by establishing flexible libraries for alternative ecological process models such as for algal growth, algal grazing, zooplankton growth, and zooplankton mortality.

- Facilitating data and model sharing by standardised model and data structures.

- Facilitating a web based access and use of ecosystem models.

- Providing user-friendly GUI for ecosystem models as well as documentation and visualisation of simulation results.

The proposed research will thoroughly apply concepts of object-oriented software engineering by means of Java to implement the object-oriented version SALMO-OO according to following procedure:

1. Understand the basic requirements of SALMO-OO by means of object-

oriented programming;

2. Design SALMO-OO that refers to these basic requirements in UML and adopt Model-View-Control (MVC) design pattern by means of object-oriented design (Gamma et al. 1995);
3. Implement the SALMO-OO Application Program Interface (API) in Java programming language (Beck 1999) and object-oriented programming;
4. Deploy the SALMO-OO API in Tomcat (Apache Jakarta Project 1999-2005) web container in order to realize web accessibility;
5. Create the SALMO-OO documents, including UML diagrams and the SALMO-OO API Specification.

Chapter 3

Materials and methods

3.1 Materials

Two types of materials are essential to be used in this project: the freshwater lake data and two lake ecosystem models: the lake model Parker (Parker 1968) and the lake model SALMO (Recknagel and Benndorf 1982). The Parker model will be used to validate GUI, web and database components of SALMO-OO. Obviously, the model SALMO is responsible for the usage of core component of SALMO-OO.

Currently, there are more than twenty databases of freshwater lakes in Australia and overseas available. We will use Saidenbach Reservoir (mesotrophic lake, Germany, bottom one in Figure 3.1) and Bautzen reservoir (hypereutrophic lake, Germany, top one in Figure 3.1) databases as case studies to test and verify the object-oriented version of the model SALMO. All measured input data for lakes are available as 10-day mean value for 36 decades per year. In summer input date will be provided for two layers (epilimnion and hypolimnion), including nutrients (e.g. dissolved inorganic phosphate), solar radiation, water temperature, water volume maximum and mean mixing depth. The validation of the model output is normally conducted by the comparison with measured data for in-lake concentration of phosphate, Chlorophyll-a, and nitrate.

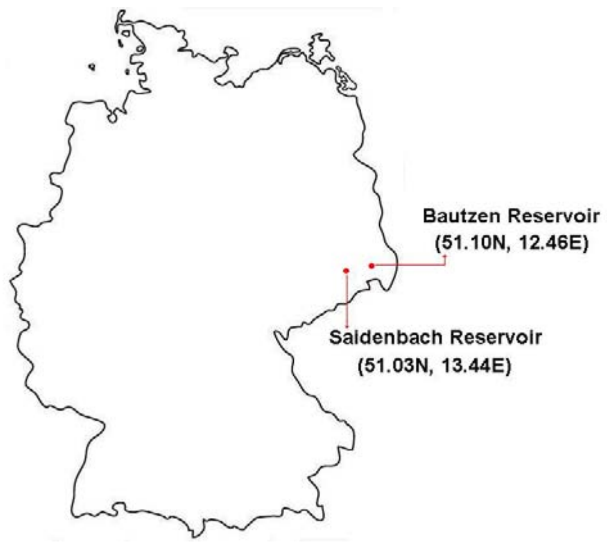


Figure 3.1 The map of Bauzten and Saidenbach reservoir

Both models are documented as sets of ODE (Parker 1968; Recknagel and Benndorf 1982). The model SALMO is also available as computer model programmed and implemented in FORTRAN IV.

3.2 Methods

3.2.1 Object-Orientation

This project will generally adopt object-orientation to realize the hypotheses that are mentioned in Chapter 2. Object-orientation requires using one of the object-oriented programming languages and compulsorily develops the SALMO class libraries that are based on object-oriented paradigm. Practically, it would not be easily to fully follow the rules of object-orientation. As Chapter 1 discussed, the previous efforts more or less failed to take all the advantages of object-orientation although those practitioners were intended to do so. To address this issue, the development of

SALMO-OO needs to use the methodology of object-oriented software engineering (Schach 2002b) associating with some novel information techniques.

The method of object-orientation firstly changes several terms of model. Underlying object-oriented paradigm, the next version of the model SALMO will be different from the Fortran IV one in program structures, designs, programming approaches. For example, the difference between the object-oriented (SALMO-OO) and structured (the Fortran IV SALMO) programming has implications to the different structure and functioning of complex ecological models. In terms of program documentation UML diagrams will be used to represent the object-oriented program structure rather than traditional flow charts that were used to document conventionally structured programs. Objects in the context of object-oriented programming mean subroutines.

Secondly, comparing to Waterfall model in the methodology of classical software engineering (Pressman 2001), object-oriented software engineering comprises three core phases: object-oriented analysis, object-oriented design, and object-oriented implementation and integration. These specific methods are useful to clarify the basic requirements, design the structure, and implement the class libraries of SALMO-OO. In other words, the development of SALMO-OO can benefit from the disciplines, methodologies, and processes of object-oriented software engineering. Consequently, it is easily for the SALMO-OO modeller to simulate the ecological knowledge behind SALMO-OO together with the realization of flexibility, reusability, portability, and interoperability.

3.2.2 The Choice of Programming Languages – Using Java

Programming languages play minor roles in building object-oriented SALMO-OO class libraries. Each programming language has advantages and disadvantages. The benchmark of comparison between them varies from applications to applications. As a result, this project will choose one object-oriented programming language that is appropriately used in this circumstance rather than one has what so called generic performance.

In this project, a number of factors have to be taken into account as the highest priority in programming language chosen. Object-oriented paradigm is the essential factor as the first consideration. Secondly, it is strong recommended using a non-commercial programming language. Non-commercial programming languages easily attract more supporting by open source community, which enable programmers to freely share source code. Open source not only benefits cutting the project budget but also provides unlimited code libraries to be reusable. Thirdly, the candidate programming languages are simple to be used. For example, it is capable of realizing web accessibility and database connection with fewer efforts, as well as platform independence. Table 3.1 lists some available programming languages by above-mentioned factors.

Programming Language	**Object-Orientation**	**Open Source Supporting**	**Simplicity (web, databases, and platform independence)**
Simula	Yes	Poor	Yes, but no platform independence
Smalltalk	Yes	Fair	Yes, but no platform independence

OO Pascal	N/A	Poor	Yes
Visual Basic	N/A	Poor	Yes, but no platform independence
C++	N/A	Fair	No
Java	Yes	Excellent	Yes
C#	Yes	Fair	Yes

Table 3.1 Choice of programming languages

Java programming language will be used underlying the comparisons in this project. Table 3.1 clearly highlights that Java programming language is truly object-oriented, extensively supported in open source community, and web-enabled and database-accessible. The second column of Table 3.1 shows that OO Pascal (Gofen 2001), Visual Basic (Brenner 2005), and C++ (Stroustrup 1995) do not belong to the true-object-oriented programming languages. Furthermore, Java earns the most popularity in the open source community because of its advantages (Cornell et al. 2002) otherwise C# would be another available choice. As far as popularity is concerned, Java and C# is overwhelming against other programming languages. Readers may refer to (Meyer 1988) and look through the popular object-oriented programming languages before last decade. In addition, Java programming language is unified applied from normal applications (e.g. to implement SALMO-OO class libraries), web applications (e.g. to implement SALMO-OO web-enabled GUI), to database connection (e.g. to implement SALMO-OO database transactions). In other words, programmers do not have to use other programming language to implement the whole SALMO-OO suite. The last but not least, the chief developers of SALMO-OO has strong background to use Java technology in this project. It would be more or less face potential risk if looking for other unfamiliar programming languages.

3.2.3 Object-Oriented Software Engineering

SALMO-OO development compulsorily uses the methodology of object-oriented software engineering to build fully object-oriented SALMO-OO class libraries. Thus, it is strongly recommended that SALMO-OO development follow the processes of object-oriented software engineering. In details, it consists of seven phases: requirements, specification, object-oriented analysis, object-oriented design, object-oriented programming and implementation, integration and deployment, and maintenance.

Requirements Phase

The requirements phase aims to deliver a requirement statement document. This requirement statement should contain the following content: the fundamentals of SALMO-OO, the desired function and performance that SALMO-OO runs in a specific computer system, the purposes for developing SALMO-OO. Through interviews, scenarios, and previous outcomes reviews, the SALMO-OO developer needs to write several reports to outline the requirements of SALMO-OO. Subsequently, these reports need to be integrated into the final requirement statement without any ambiguous agreements between the SALMO-OO modeller and the SALMO-OO developer. In addition, building a prototype system to verify whether the requirement statement satisfies the SALMO-OO modeller is suggested. The prototype system simulates the SALMO-OO requirements to some extent, and usually delivers some plain GUI or simple calculation results.

Specification Phase

The specification phase targets a specification document. It is commonly used in classical software engineering. The next section will introduce object-oriented analysis to replace the usage of specification techniques. However, it does not means that the specification phase would be taken away from the methodology of object-oriented software engineering. In this project, we will use the specifications to clarify the contradictory requirement between the ecological modeller of SALMO-OO and the programmer of SALMO-OO. Normally, modellers always want to make target applications perfect against the reality that programmers can achieve them in an easy manner. So it is important for both participants to agree on with each other what exactly the target applications are.

For example, after a number of times conversations, we decide that the SALMO-OO specification document should contain the following content: 1) to use fourth order Runge-Kutta to solve the SALMO-OO ODE calculations; 2) to list all the prerequisite and post-requisite of every class in the SALMO-OO class library; 3) to define the format of input and output data; 4) to propose the implementation of function and performance of SALMO-OO by a serial of prototypes; 5) to realize web accessibility and display only plain GUI.

Object-Oriented Analysis Phase

Object-oriented analysis (OOA) uses over 60 different techniques (Schach 2002a). We adopt UML to describe the analysis results by means of use-case modelling (Jacobson 1992) and class modelling in this project.

Use-case modelling is used to generate UML use-case diagrams. A UML use-case diagram consists of actors and actions. The extraction of actors and actions can be deduced from the scenarios in the requirement phase. The term actor can be a person who starts simulation, or a species of aquatic animal that has been simulated in the ODE equations, or other state variables. As one of the tools of UML, UML use-case diagrams illustrate the results of use-case modelling in the OOA phase. For example, Figure 3.2 shows an alga as actor and its growth as action.

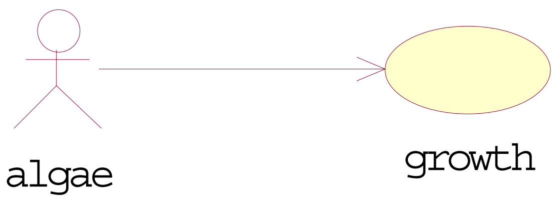


Figure 3.2 UML use-case diagrams for illustrating the relationship between algae and algal growth

Class modelling is used to discover the classes from the requirement statement. *Noun extraction* technique will be used to create the classes. At the beginning of the model development, it is strongly recommended to revise the SALMO-OO requirement statement because a concise statement will simplify the task of noun extraction. There are two ways to discover nouns from the SALMO-OO requirement statement: the explicit way and the implicit way. Usually, the explicit noun extraction focuses on identifying nouns in the requirement statement. Some of these nouns will be abstracted to the classes, and others could be the attributes of the classes or discarded. For example, phytoplankton is a name of an aquatic organism, which is a key component of lake ecosystems, so it is logical to have a class called phytoplankton. However, explicit noun extraction sometimes fails to thoroughly discover the classes that are necessary to be used in the object-oriented design phase. Thus, the implicit noun extraction makes up for this shortcoming. For example, SALMO-OO will extend its growth and grazing model library. It is impossible to represent every individual model by a class, thus it is better to merge each of the growth and grazing model into a class called AlgaeLibrary. From the developer's point of view, there are several implicit classes behind the requirement statement. These implicit classes are essential to be used in the object-oriented design phase. Alternatively, these implicit classes could hide in the form of verb rather than nouns. For example, SALMO-OO needs to import data from databases. It involves database connection transactions. Specifically in Java, it is called Java Database Connection (JDBC). Since our project is named SALMO-OO, a class called SalmoJDBC may represent the verb connect.

Another task of class modelling is to find class fields and class methods. A class field belongs to the attributes of a class, and a class method belongs to the behaviours of a class. Fortunately, the taxonomy of organism in biology was exemplary for the object-oriented paradigm. Therefore, it is not difficult to differentiate the biological attributes and behaviours regarding a species of animal or plant. Thus, we could follow directly the biological categories to complete the SALMO-OO class modelling.

After completing use-case modelling and class modelling, the relationships and interactions among classes need to be determined. This solution will be introduced in the next paragraph.

Object-Oriented Design Phase

According to (Schach 2002b), the aim of object-oriented design (OOD) is to design the product in terms of objects. An object is an instance of a class, and therefore the elements of a class and the relationships and interactions among classes play significant roles in the OOD phase. UML provide UML class, sequence, and communication diagrams to represent the design results.

The UML class diagrams show the inner elements of classes and the relationships between classes. The UML class diagrams contain one or more single class models. A class model comprises a class name, class fields and class methods. The UML class diagrams can be drawn directly from the results of class modelling in the OOA phase. As the model kernel, these classes can be drawn directly via class diagrams such as a UML class diagram is shown in Figure 3.3. However, not every requirement statement has the ability to completely deliver model descriptions. This situation often happens when either a modeller lacks experience in software application development, or the developer lacks ecological knowledge. Thus, OOD needs to discover those implicit classes such as database connection, web transaction, and logic invocation. Although there is no obvious order in the OOD phase, it is recommended to design the UML class diagrams as the first step.

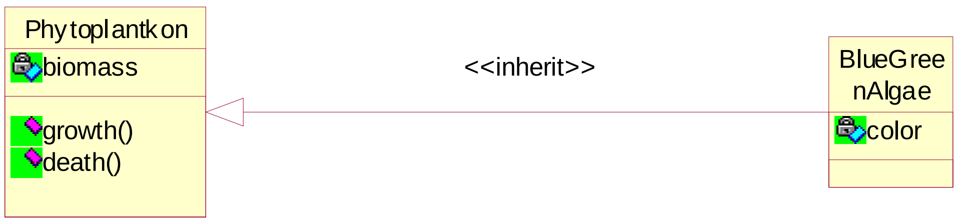


Figure 3.3 A UML class diagrams for phytoplankton and blue-green algae class

The UML sequence diagrams model the flow of logic in a virtual system. They are one of the dynamic models. The structured programming uses flow charts to draw the model SALMO life cycle in the design phase, whereas, OOD simulates the SALMO-OO life cycle via sequence diagrams. Instead of sequence, selection, and loop structure in the structured design, OOD draws UML sequence diagrams in terms of the sequence logic of the classes. In other words, UML sequence diagrams record the order that the classes execute in a computer system. A UML sequence diagrams can start from an actor of a UML diagrams for use-case till the final action of this use case, and then repeat this process generating others, and eventually integrate every individual UML sequence diagrams into one comprehensive one.

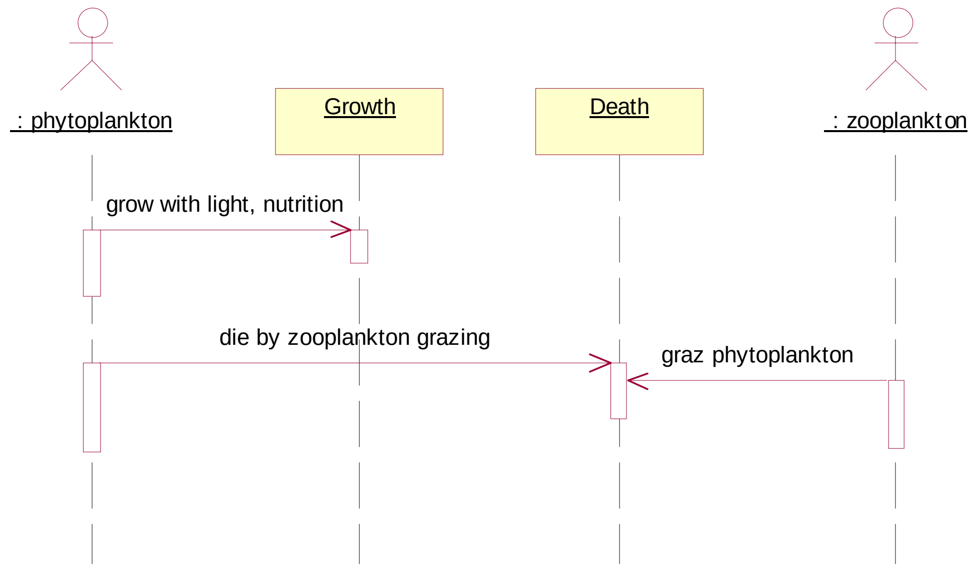


Figure 3.4 A UML sequence diagram for phytoplankton and zooplankton

The UML communication diagrams show the message flow between objects. It is important to draw the dynamic value-passing process in SAMLO-OO because there are a large number of mathematical calculations. Figure 3.4 demonstrates the general value-passing process in four top-level objects.

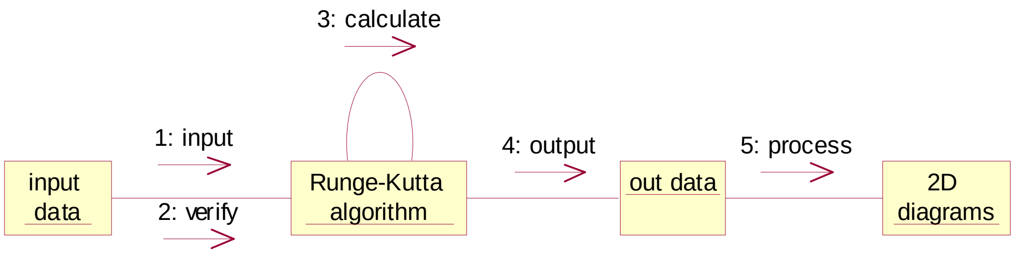


Figure 3.4 A UML communication diagrams for top-level value passing

Other UML diagrams such as UML activity, state machine, and component diagrams can also be applied in the OOD phase. It differs based on the purpose of each specific software application. In this project, we adopt UML class diagrams to describe the static nature of SALMO-OO, use UML sequence diagrams to arrange the execute sequence, and simulate the dynamic value passing process via UML diagrams for communication.

The Implementation and Integration Phase

The task in this phase is to implement the SALMO-OO class library by means of object-oriented programming (OOP) together with a simple SALMO-OO database. Generally, OOP mainly works with one of the object-oriented programming languages. We use Java programming language in this project. In addition, some extreme programming (XP) methods (Beck 1999) and Prototyping will accompany the OOP techniques.

First of all, it is necessary to create a simple relational database management system (RDBMS) that stores the freshwater lake data. Previously, the data were stored in the format of Microsoft Excel file (usually called MS Excel with postfix ‘xls’). For this project the data is restored into the MySQL (MySQL AB 1995-2005) RDBMS in this project. In this context, the process that transfers data from ordinary computer system files (e.g. MS Excel files) to RDBMS (e.g. MySQL) is called data pre-processing. The first step of data pre-processing is to create tables in the MySQL RDBMS via Data Definition Language (DDL is a category of SQL, and is used to create, delete and alter tables in RDBMS). Secondly, it needs to extract the 37-day measured data from the MS Excel files via data transformation and cleaning, and calculate 360-day analogous data by means of mathematical interpolation algorithm. Finally storing these 360-day analogous data into the MySQL tables that was created in the first step. The last step aims to validate the data correctness. It can be suggested using some software tools to complete the task of data validation. Figure 3.5 illustrates the whole process of data pre-processing.

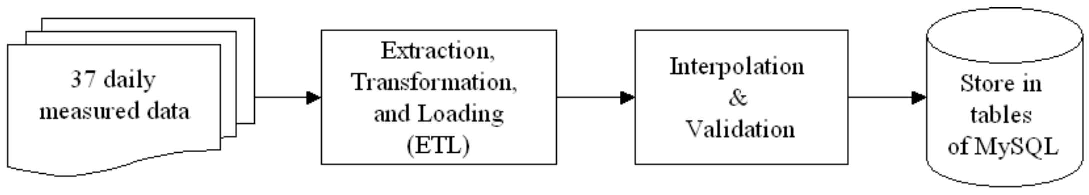

37 daily measured data
Extraction, Transformation, and Loading (ETL)
Interpolation & Validation
Store in tables of MySQL

Figure 3.5 Data pre-processing

The primary task in this phase is to complete the SALMO-OO class library, including its integration and testing. Using Java programming language and well-designed UML UML class diagrams confirm the characters of object-orientation. As a result, there are no more tasks other than typing code in this step. Even in an ideal circumstance, some UML tools (e.g. Rational Rose (Rational Software 2005)) are able to directly convert UML class diagrams to Java source code.

The implementation and integration of SALMO-OO takes advantage of OOP. Nevertheless, it is still not good enough to shorten the life cycle of developing SALMO-OO. Thus, some extreme programming methods (e.g. testing first and peer programming) are introduced to overcome the shortcomings of the OOP techniques.

XP methods permit the SALMO-OO developer to separate the SALMO-OO class library into several smaller sub modules. These sub modules can be extremely small to reflect even a simple task such as database connection. Subsequently, each sub module is set a priority number. The SALMO-OO developer programs testing code for these sub modules by priority number. In addition, another team member accompanies the developer and proofreads the typed source code. After every sub module has been completely verified and tested, they will be integrated into a bigger module following the design diagrams. In this project, a bigger module is a package that represents a group of classes that have similar functionality. Finally, all the packages need to be integrated into one application, which is the SALMO-OO class library.

SALMO-OO Deployment

The deployment of SALMO-OO is the final phase in this project. The task of this step is to install both the SALMO-OO class libraries and its web-enabled applications into a computer that is connected to the Internet. This computer usually is called sever. The discussion of server technology is beyond the purpose of this project, and it is advised that readers refer to other related sources.

3.2.4 Prototyping

Prototyping techniques are needed in developing SALMO-OO class libraries. Those above-mentioned methods of object-oriented software engineering might not specifically guide SALMO-OO programmers how to implement SALMO-OO class libraries but propose the general solutions to simplify the SALMO-OO development. Thus, it needs to use some software processes such as prototyping to solve the goals in practice.

Prototyping has four basic categories: Rapid Prototyping, Throw-Away Prototyping, Incremental Prototyping, and Evolutionary Prototyping. We propose to use certain Incremental Prototyping in order to steadily update SALMO-OO. Thus, the first SALMO-OO prototype system needs to point out how the previous SALMO model works. The next step will create the next version of the SALMO-OO prototype system, which could be built based on the achievements that the first prototype system has acquired but throw away those aspects that do not work. Alternatively, the next prototype system can be built by improving the first one. In general, the choice relies on how much the prototype system meets the SALMO-OO creator's expectation.

3.2.5 Web Tier Techniques

SALMO-OO will adopt some web tier Java technologies to implement web accessibility since SALMO-OO class libraries decide to programme in Java programming language. Web tier techniques in Java family involves in JavaServer Pages (JSP) (Sun Microsystems 1994-2005d), Servlet (Sun Microsystems 1994-2005c), and Applet (Sun Microsystems 1994-2005a), which all of them will be used to build the SALMO-OO web-based applications. The purpose using JSP is to enable to create dynamic web content, but it is difficult for JSP to realize complex GUI such as drawing algae dynamic diagrams. Thus, we will use Applet to solve this shortcoming of JSP. In addition, Servlet encapsulates low-level computer network protocols such as Hyper Text Transfer Protocol (HTTP), and the SALMO-OO developer only needs to take into account flow logic between JSP web pages.

These web tier Java technologies enable platform-independence, but require the assistance of Internet Explorer (IE) (Microsoft 2006b) when the SALMO-OO users access it online. Therefore, the combination of JSP, Servlet, Applet, and IE enables the SALMO-OO to be web accessible.

Chapter 4

Results

4.1 Prototypes

This research successfully completes SALMO-OO as well as achieves the aims and the hypothesis. These outcomes encompass a freshwater lake database, SALMO-OO class libraries, a few plain web-enabled applications, UML diagrams, documents, and a SALMO-OO API specification. This research achieves all of the sub-outcomes in SALMO-OO with fair validation by three prototype systems.

The first prototype is the object-oriented Parker model. The Parker model is an aquatic model that simulates four state variables: algae, cladocera (zooplankton), salmon (fish), and phosphate. This model uses the method of Runge-Kutta to calculate the differential equations of the four state variables, which it is as exact same as the model SALMO does. The reason why we choose this model as the first prototype comes from its simplicity. The Parker model neither involves input data, nor considers seasonality or stratification of the water body. Moreover, it elaborates output data by a few line charts in its document, which is helpful to check the results of the object-oriented Parker model whether correct or not. Thus, we rapidly draw the Parker UML diagrams (Figure 4.1-4.3).

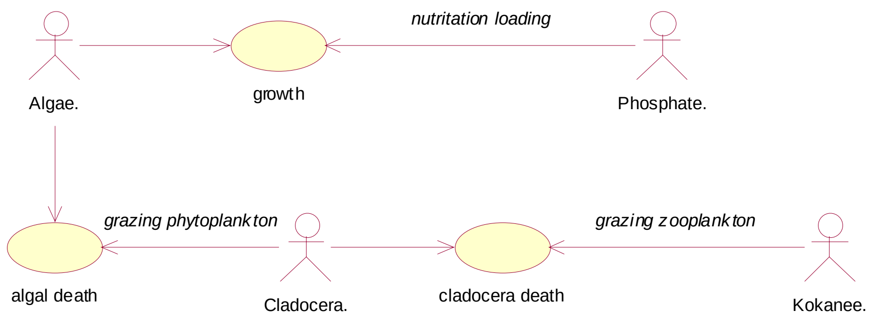


Figure 4.1 A UML use-case diagram for food web in the Parker model

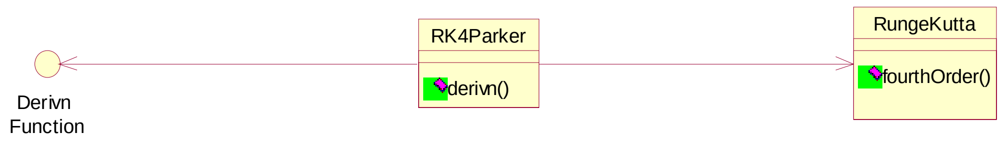


Figure 4.2 A UML class diagram for Runge-Kutta calculations in the Parker model

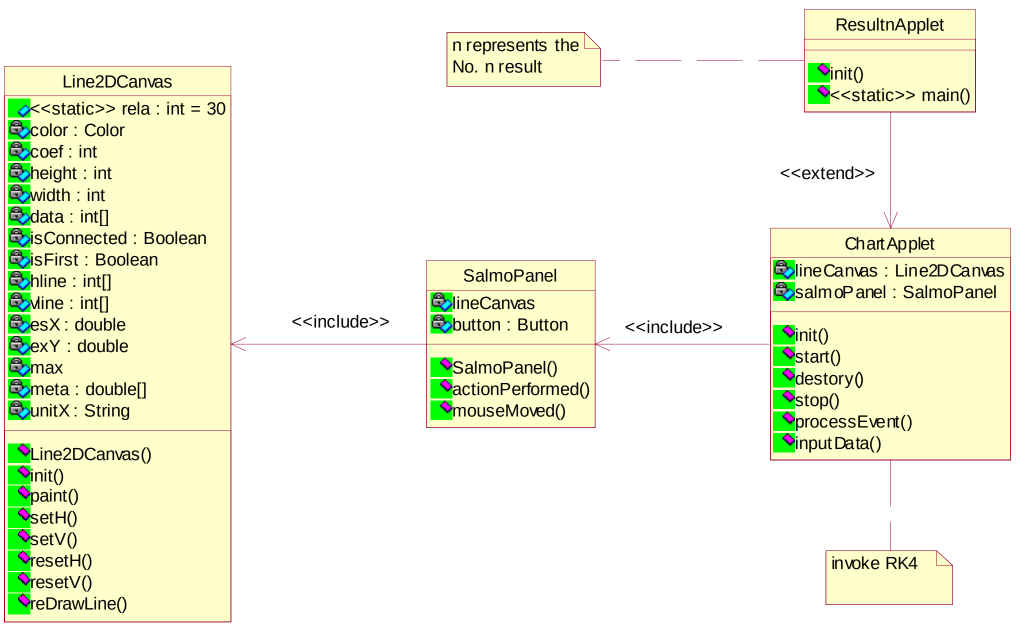


Figure 4.3 A UML class diagram for GUI in the Parker model

The first prototype succeeds in simulating the Parker model by using the fourth order Runge-Kutta algorithm as well as an initial GUI demonstrations, including curve colour, style, and layout. Most importantly, the first prototype is web-enabled. Figure 4.4(a) displays the GUI results. But it remains some problems such as scales in the coordinates, which propose to be solved in the next prototypes.

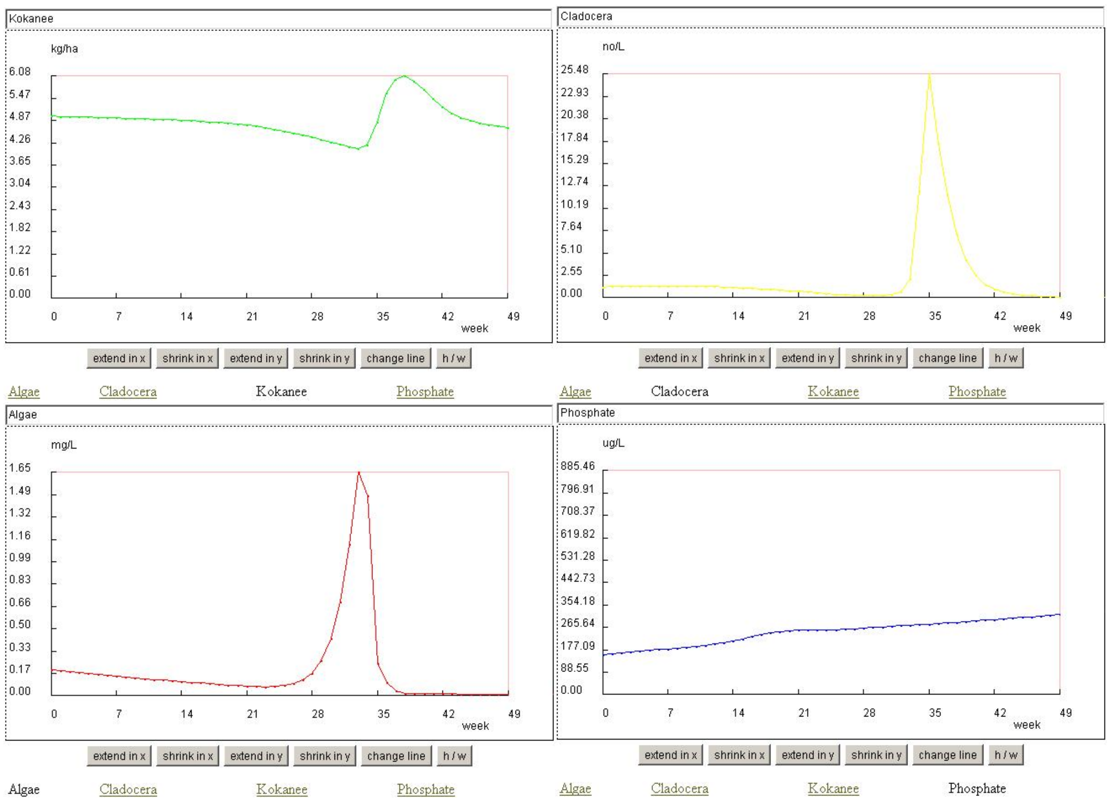


Figure 4.4(a) The first prototype - the Parker model results

Comparing to the original Parker model results (Figure 4.4(b)), it is undoubted that the correctness of the Runge-Kutta calculations and the GUI displaying of the first prototype but phosphate does not decrease around 32 week from the Figure 4.4(a). This unpredicted error could be resulted by any programmed mistake. Fortunately, it does not have significant influence to the general simulated results therefore can be remained. In addition, it further discovers that the requirement of the target SALMO-OO applications. After continuous discussions, the SALMO-OO modeller agrees that the final SALMO-OO GUI should be similar to the one displayed in Figure 4.4(a) but needs to revise the scales in the x-axis to integer. The subsequent task plans to add the SALMO ODE instead of the Parker one.

814 BIOMETRICS, DECEMBER 1968

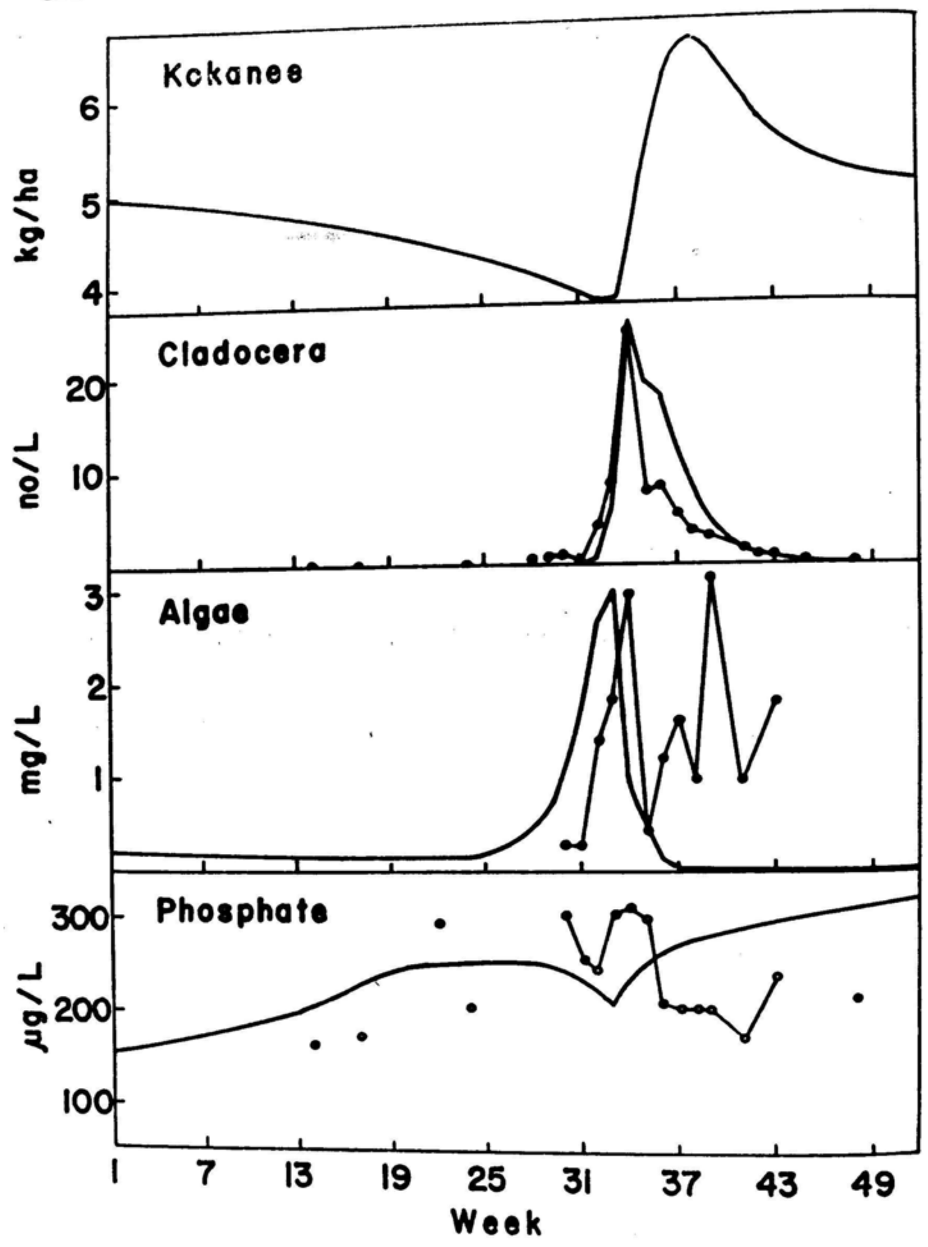


FIGURE 4

SIMULATED AND MEASURED VALUES FOR 1966 OF INORGANIC PHOSPHATE CONCENTRATION, ALGAL CONCENTRATION, CLADOCERAN DENSITY, AND KOKANEE STANDING CROP IN KOOTENAY LAKE

Figure 4.4(b) The original Parker model results ((Parker 1968).

The second prototype implements the basic model version of SALMO as previously implemented by FORTRAN IV in 1980. It aimed to prove that the Java version of the

SALMO could produce the same results as the Fortran version. As a result a basic version of SALMO-OO had been developed with an extended GUI.

The third prototype aimed at the implementation of an advanced version of SALMO-OO. The model had been extended by the ODE for a third functional algal group allowing the simulation of three algal groups rather the two as in the basic version of SALMO-OO. At the same time a library for alternative algal process models was implemented, including algal growth, zooplankton grazing, and zooplankton growth. As far as the user interface is concerned, this advanced version of SALMO-OO offers two options. One is the stand-alone edition, which runs in a single computer with friendly GUI, where users can select any available lake, year, growth model, grazing model, and scenario analysis for lake management in any windows operating environment (e.g. Microsoft Windows 2000). Another is the network edition. The SALMO-OO network edition permits users access to SALMO-OO through the Internet. It covers all of the functionalities that the SALMO-OO stand-alone edition has except the data import function.

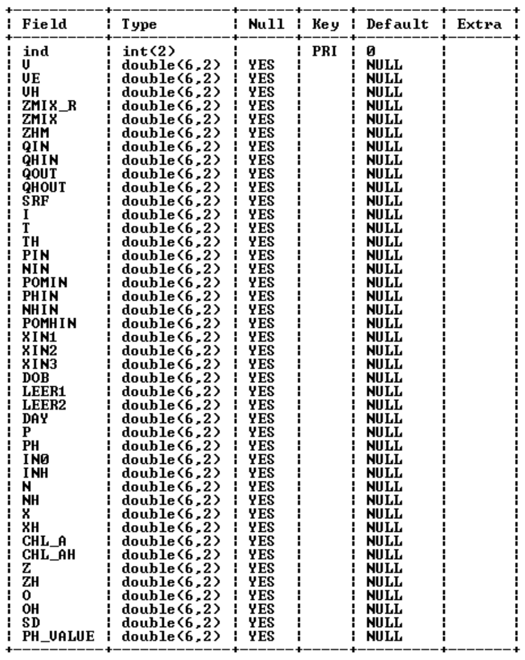
The advanced version of SALMO-OO also allows to perform multiple parameter optimisation by means of genetic algorithms (GA) according to (Back et al. 2000a)) also implemented in Java.. This GA program is designed to optimise SALMO-OO parameters only available for the standalone mode of the advanced version of SALMO-OO. The GA program initialises the size of populations to 100 and allows be evolve 50 generations. As far as the crossover operator is concerned, the Java version of GA program keeps the original design. During cross calculation, this GA program randomly chooses 8 chromosomes among parent populations to perform dual mating, which loop-reproduce 100 child chromosomes (100 groups of SALMO-OO parameters). Subsequently, each generation sends these calculated 100 groups of parameters to SALMO-OO application for fitness evaluation. Finally, the GA program selects the best 100 chromosomes between the parent populations and the child populations by means of rank-based selection (Back et al. 2000b). Notes that we do not enclose any GA program into the SALMO-OO suite because it beyond our perspective in this project. This discussion only shows the possibility and flexibility that adds advanced functionality in the SALMO-OO suite.

The advanced version of SALMO-OO contains the completed sub-outcomes: a database (which is the freshwater lake data tables stored in MySQL RDBMS), a group of class library, web-enabled applications, UML diagrams, documents, and a SALMO-OO API specifications.

4.2 The SALMO-OO Database

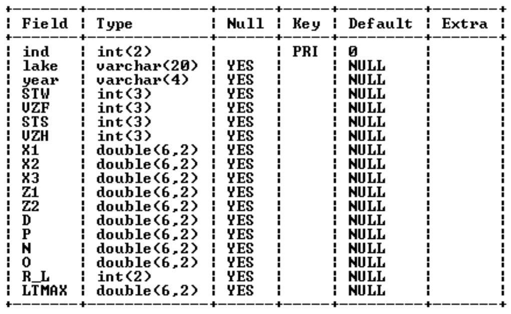
The SALMO-OO database consists of a number of tables that store freshwater lake data. These tables have two types of structures. One is the lake measured data table (data table). A data table owns a name, which consist of three parts: the lake name, the artificial mixing data flag, and the year. These three parts joint by the underscore mark. For example, Barossa_1978 represents Barossa Reservoir measured data in 1978; and Barossa_am_1978 represents Barossa Reservoir artificial mixing data in

1978. Furthermore, each data table has 44 stable columns and 360 stable rows. Its first column represents the day number from 1 to 360 and acts as the primary key in the lake measured data table. The other 43 columns exactly mirror the original structure of the freshwater measured dataset except the lake, year, and day number column. The amount of lake measured data tables depend on how many lake data will be modelled. Figure 4.5(a) displays the detail structure of a data table.



```
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| ind      | int(2)      |      | PRI | 0       |       |
| V        | double(6,2) | YES  |     | NULL    |       |
| VE       | double(6,2) | YES  |     | NULL    |       |
| VH       | double(6,2) | YES  |     | NULL    |       |
| ZMIX_R   | double(6,2) | YES  |     | NULL    |       |
| ZMIX     | double(6,2) | YES  |     | NULL    |       |
| ZHM      | double(6,2) | YES  |     | NULL    |       |
| QIN      | double(6,2) | YES  |     | NULL    |       |
| QHIN     | double(6,2) | YES  |     | NULL    |       |
| QOUT     | double(6,2) | YES  |     | NULL    |       |
| QHOUT    | double(6,2) | YES  |     | NULL    |       |
| SRF      | double(6,2) | YES  |     | NULL    |       |
| I        | double(6,2) | YES  |     | NULL    |       |
| T        | double(6,2) | YES  |     | NULL    |       |
| TH       | double(6,2) | YES  |     | NULL    |       |
| PIN      | double(6,2) | YES  |     | NULL    |       |
| NIN      | double(6,2) | YES  |     | NULL    |       |
| POMIN    | double(6,2) | YES  |     | NULL    |       |
| PHIN     | double(6,2) | YES  |     | NULL    |       |
| NHIN     | double(6,2) | YES  |     | NULL    |       |
| POMHIN   | double(6,2) | YES  |     | NULL    |       |
| XIN1     | double(6,2) | YES  |     | NULL    |       |
| XIN2     | double(6,2) | YES  |     | NULL    |       |
| XIN3     | double(6,2) | YES  |     | NULL    |       |
| DOB      | double(6,2) | YES  |     | NULL    |       |
| LEER1    | double(6,2) | YES  |     | NULL    |       |
| LEER2    | double(6,2) | YES  |     | NULL    |       |
| DAY      | double(6,2) | YES  |     | NULL    |       |
| P        | double(6,2) | YES  |     | NULL    |       |
| PH       | double(6,2) | YES  |     | NULL    |       |
| IN0      | double(6,2) | YES  |     | NULL    |       |
| INH      | double(6,2) | YES  |     | NULL    |       |
| N        | double(6,2) | YES  |     | NULL    |       |
| NH       | double(6,2) | YES  |     | NULL    |       |
| X        | double(6,2) | YES  |     | NULL    |       |
| XH       | double(6,2) | YES  |     | NULL    |       |
| CHL_A    | double(6,2) | YES  |     | NULL    |       |
| CHL_AH   | double(6,2) | YES  |     | NULL    |       |
| Z        | double(6,2) | YES  |     | NULL    |       |
| ZH       | double(6,2) | YES  |     | NULL    |       |
| O        | double(6,2) | YES  |     | NULL    |       |
| OH       | double(6,2) | YES  |     | NULL    |       |
| SD       | double(6,2) | YES  |     | NULL    |       |
| PH_VALUE | double(6,2) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
```

Figure 4.5 (a) Data table structure



```
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| ind   | int(2)      |      | PRI | 0       |       |
| lake  | varchar(20) | YES  |     | NULL    |       |
| year  | varchar(4)  | YES  |     | NULL    |       |
| STW   | int(3)      | YES  |     | NULL    |       |
| VZF   | int(3)      | YES  |     | NULL    |       |
| STS   | int(3)      | YES  |     | NULL    |       |
| VZH   | int(3)      | YES  |     | NULL    |       |
| X1    | double(6,2) | YES  |     | NULL    |       |
| X2    | double(6,2) | YES  |     | NULL    |       |
| X3    | double(6,2) | YES  |     | NULL    |       |
| Z1    | double(6,2) | YES  |     | NULL    |       |
| Z2    | double(6,2) | YES  |     | NULL    |       |
| D     | double(6,2) | YES  |     | NULL    |       |
| P     | double(6,2) | YES  |     | NULL    |       |
| N     | double(6,2) | YES  |     | NULL    |       |
| O     | double(6,2) | YES  |     | NULL    |       |
| R_L   | int(2)      | YES  |     | NULL    |       |
| LTMAX | double(6,2) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

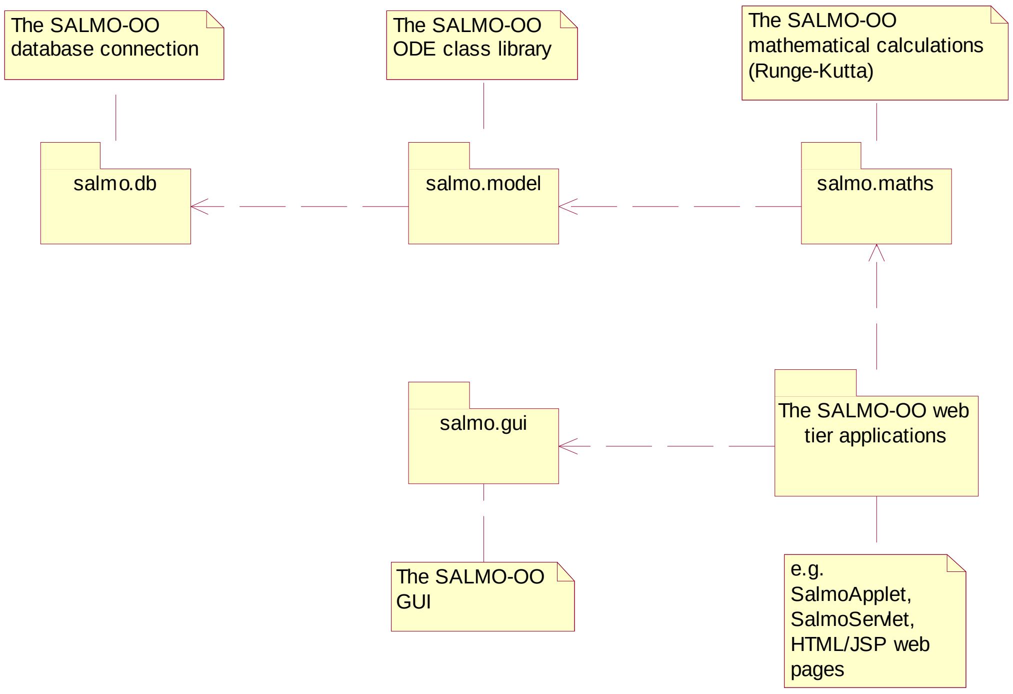
Figure 4.5 (b) Profile table structure

Another is the freshwater lake profile table (profile table). There is only one profile table in this project, which is called SalmoMeta (see Figure 4.5 (b)). The SalmoMeta consists of 18 stable columns and unlimited rows because one row stores a group of parameters for a specific lake in a year, and it is unpredictable how many lakes and different years will be applied in the future. Both the data tables and the profile tables provide input data for SALMO-OO.

SALMO-OO database fails to be well designed in this project. But this weakness plays minor role in the SALMO-OO applications because the SALMO-OO class library is highly independent on any dataset interface. Moreover, the database technique is rapidly varied so that to exhaust a permanent database structure implementation in an easy manner. Therefore, I would personally suggest leave the database issues alone in this SALMO-OO applications.

4.3 The SALMO-OO Class Library

The SALMO-OO class library comprises four packages: the salmo.model, the salmo.maths, the salmo.gui, and the salmo.db package. As its name shows, the salmo.model package plays a core role in the SAMLO-OO class library. Figure 4.6 displays the dependencies between these four packages in the form of UML component view, including the SALMO-OO web tier applications (note: the SALMO-OO web tier applications do not belong to the SALMO-OO class library).



The SALMO-OO database connection
The SALMO-OO ODE class library
The SALMO-OO mathematical calculations (Runge-Kutta)
salmo.db
salmo.model
salmo.maths
salmo.gui
The SALMO-OO web tier applications
The SALMO-OO GUI
e.g. SalmoApplet, SalmoServlet, HTML/JSP web pages

Figure 4.6 The SALMO-OO component view

The salmo.model package represents the core component (ODE) in SALMO-OO. This package consists of nine classes (AlgaeLibrary, Detritus, Light, Nitrate, Oxygen, Phosphate, Phytoplankton, Salmo, and Zooplankton). Each class represents a state variable except for the Light class, the Salmo class, and the AlgaeLibrary Class. The Light class contains the light equations from the previous SALMO model. In SALMO-OO, the light equations are separated from the algal growth equations as an object because of using the object-oriented paradigm. Moreover, we add Salmo class as a value object. The function of the Salmo class is to hold the SALMO-OO parameters and input data. All of the other classes (except the Salmo class) in this package inherit the Salmo class as its subclass in order to directly acquire these data values. In addition, the AlgaeLibrary class realizes a flexible library for a number of optional growth and grazing models. The AlgaeLibrary class indirectly acquires the data values from the Salmo class by inheriting the Phytoplankton class. In other words, if we look at the Salmo class as a father, then the Phytoplankton class is its son; consequently the AlgaeLibrary class is grandson of the Salmo class.

The salmo.maths package is the component that models Runge-Kutta calculations. This package consists of three classes (RungeKutta, SalmoData, and SalmoRK4) and one interface (DerivnFunction). The RungeKutta class and the DerivnFunction interface are provided by Michael Thomas Flanagan (Flanagan 2005), which simulate the logic of the fourth order Runge-Kutta algorithm. The SalmoRK4 class implements the DrivnFunction interface in order to automatically communicate with the RungeKutta class. Subsequently, the SalmoRK4 class executes the ODE calculations by importing the classes in the salmo.model package. Finally, the SalmoData class holds the final calculated results as an output value object.

The salmo.gui package describes the SALMO-OO GUI logic. This package consists of four basic classes (SalmoCanvas, SalmoDiagram, SalmoFrame, and SalmoSelection). The SalmoCanvas class provides a painting canvas for displaying the SALMO-OO output data. The other three classes build a frame for users to select modelling items and observe the output data in the form of a graph. Moreover, these frame classes provide a user interface to print to printer, create output data files, and perform the function of data pre-processing. Figure 4.7 demonstrates an example of the salmo.gui package implementation.

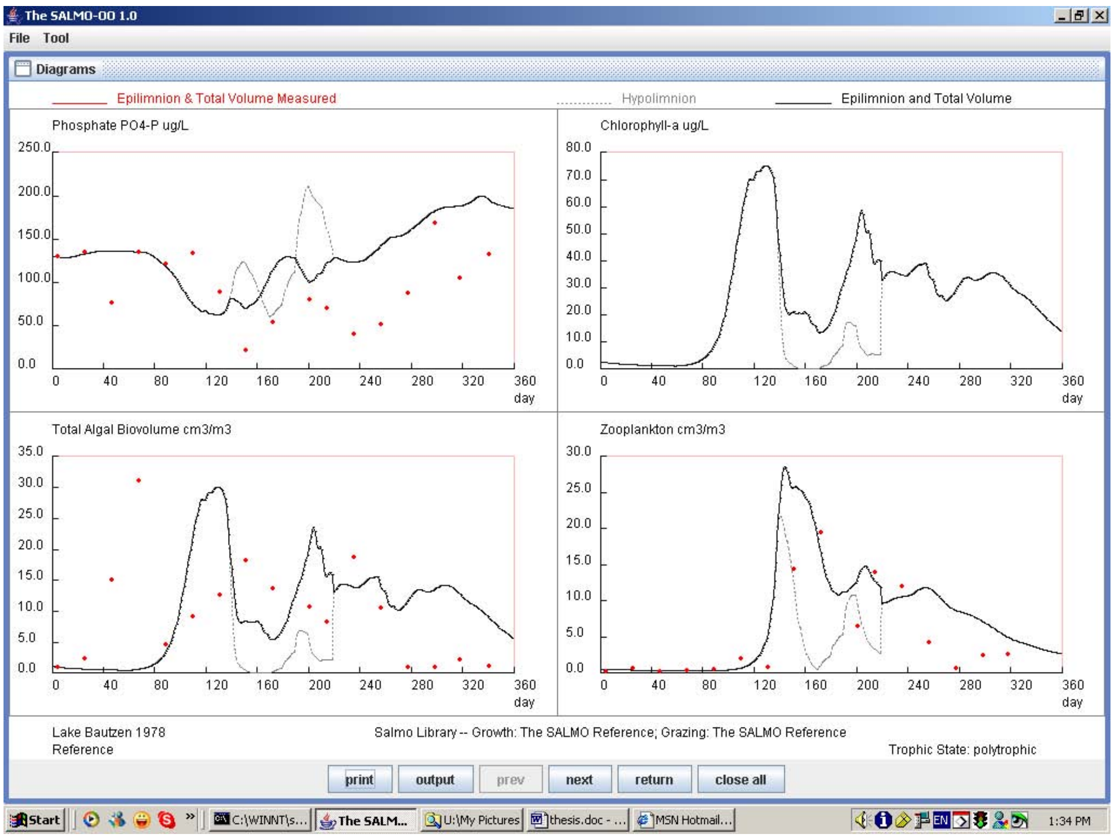


Figure 4.7 Visualisation of validation results for concentrations of phosphate PO4-P, chlorophyll a, total algal biovolume and zooplankton biovolume simulated by SALMO-OO for the Bautzen Reservoir in 1978.

The salmo.db package only contains one SalmoJDBC class. This class realizes the database connection function. Specifically, the SalmoJDBC class needs to cooperate with a source package that is provided by the tools in MySQL version 4 RDBMS.

4.4 Web-Enabled Applications

Web accessibility is available in SALMO-OO network edition by means of JSP, Servlet, and Applet technologies. Although these web tier outcomes use object-oriented paradigm, the applications lie outside the SALMO-OO class library. The current version consists of one HTML file, three JSP files, one Servlet class, and one Applet class. Their relationships are illustrated in Figure 4.8.

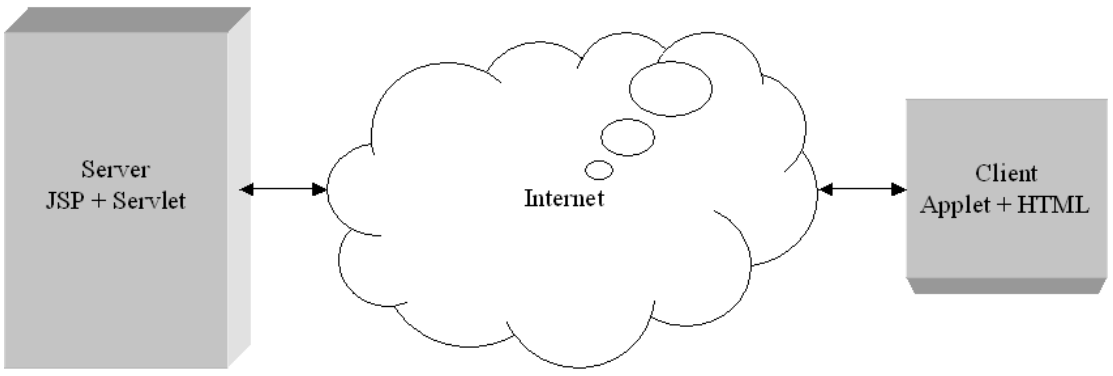


Figure 4.8 Web tier structure for SALMO-OO

On the server side (on the left in Figure 4.8), the JSP and Servlet applications will run in a computer that has installed a web container tool. Firstly, the JSP files are converted into the JSP classes by this web container. These JSP classes are in charge of the presentation logic, which represents the dynamic web pages. SALMO-OO network edition includes the Prepare JSP file, the Confirm JSP file, and the Results JSP file. Secondly, the usage of the SalmoServlet class is to transmit message between JSP classes. Thus, it focuses on the control logic.

On the client side (on the right in Figure 4.8), the HTML, JSP and Applet applications will run in a terminal computer that has installed an Internet Explorer tool. The JSP files are imaged into the JSP web pages then are visibly displayed in this Internet Explorer. These JSP pages record user request services meanwhile convert into Java class on the server side. After receiving the redirection request of the SalmoServlet class, the Results JSP web page finally acquires the SALMO-OO calculated results encapsulated within an object that activates the embedded SalmoApplet class. Subsequently, the SalmoApplet class extracts data from the calculated result object, and then invokes the GUI classes in the slamo.gui package, which eventually displays the graphical calculated results in the web GUI. Any low-level network transactions are transparent to the SALMO-OO developer.

4.5 The SALMO-OO Documents

As one of the most important outcomes, the SALMO-OO documents cover the detail descriptions of the development of SALMO-OO. These documents include a requirement statement document, a specification document, a group of UML diagrams, and one SALMO-OO API specification.

4.5.1 The SALMO-OO Requirement Statement Document

The SALMO-OO requirement statement aims to clarify the requirement of SALMO-OO. This document contains the SALMO-OO description, the purpose to build it, the expectation of performance and function, and the development duration. Formal content is as follows:

SALMO-OO is a software application. Its theoretical supporting comes from freshwater lake modelling and based on the previous Fortran version of the model SALMO. The main objective of SALMO-OO is to investigate how to develop ecosystem models by means of object-oriented technology.

The SALMO-OO target applications should complete the following functions: 1) to implement a group of structured transparent class libraries that simulate the model SALMO ODE; 2) to implement the fourth order Runge-Kutta algorithm for the ODE calculations; 3) to illustrate the daily dynamics of all the state variables in 2D coordinates; 4) to enable SALMO-OO to be accessed in the Internet as well as friendly GUI.

As far as SALMO-OO performance is concerned, flexibility is the highest priority requirement. The SALMO-OO class library must be highly flexible in response to any modification. Moreover, users can run the SALMO-OO class library as their selection such as arranging different lake dataset, scenario analysis, and varying the parameter values. In other words, the SALMO-OO creator and developer can flexibly change anything they want as well as easy access and use by the SALMO-OO users. In addition, high response rate of the SALMO-OO network edition is expected.

As the above-mentioned requirements, it would be estimated the first version of SALMO-OO could be delivered during 12 months.

4.5.2 The SALMO-OO Specification

The SALMO-OO specification contains three parts: naming list, database and class library specification.

Naming list defines names that will be used in developing SALMO-OO. These names include various file names, API's internal element, database table and its field names, and diagram names. Once these names are decided, they will be used throughout the whole development of SALMO-OO as permanent definitions. Table 4.1 displays a fragment of the SALMO-OO naming list.

NAME	CATEGORY	FORMAT	COMMENT
V	measured data	2 decimals	in MS Excel / field name in MySQL tables
VE	measured data	2 decimals	in MS Excel / field name in MySQL tables
VH	measured data	2 decimals	in MS Excel / field name in MySQL tables
Lake_am_Year	.xls name	37 rows, 44 columns	MS Excel as Legacy System
Lake_Year	table name	360 rows, 44 columns	in MySQL database
AlgaeLibrary	class name	.class	Java filename in package salmo.model
Phytoplankton	class name	.class	Java filename in package salmo.model
SalmoCanvas	class name	.class	Java filename in package salmo.gui
SalmoRK4	class name	.class	Java filename in package salmo.maths
prepare	class name	.jsp	Java Server Pages (JSP)

Table 4.1 Examples of the naming list of SALMO-OO

The database specification specifies the ecological data format that is stored in the RDBMS. This project defines that a data table consists 360 records (rows) and 44 fields (columns). The first column is the index as the primary key of the table, which records are incremental integer type number. Other columns hold the double type data with stable two bit decimals. They forbid being given null values and set zero value in the case of lacking data. In addition, it needs to manually copy the value data at season varying point only if the value of the next day is zero. For example, the data value needs to be copied from the last day of summer to the first day of autumn only if the data value of the first day of autumn is zero. The reason why copy data value at season point comes from the previous SALMO model rules. Without copying job, it is possible that the SALMO-OO Runge-Kutta calculations could overflow the computer memory in some exceptions.

The class library specification describes the system boundary of SALMO-OO. It covers the limitation of user transactions, the look and feel of the GUI, the choice of the specific RDBMS, and the choice of the ODE solutions. SALMO-OO final version will provide a few basic user transactions: 1) web accessibility; 2) various algal growth and grazing model library selections; 3) various scenario analysis selections; 4) at least three algal functional group selections; 5) graphically display output results. Secondly, SALMO-OO final version will provide a standard 2D line chart for output results. This line chart will be based on 2D coordinates with the essential units, scales, and legends. The colour of display curve is black, grey, and red, which respectively represent the epilimnion and total volume output data, hypolimnion and scenario output data, and the measured data. Moreover, the line chart displays on the transparent or white background. In addition, these diagrams should display four state variables' output results at one time at the same page, and the users can move pages to previous and next. Figure 10 shows an example. Thirdly, SALMO-OO final version uses the fourth order Runge-Kutta algorithm to calculate the SALMO-OO ODE. Finally, SALMO-OO final version is able to run in single computer and network environment, both of them adopt MySQL version 4 RDBMS. In the case of running in the network environment, Tomcat version 4 will be used to publish the SALMO-OO web pages as an application server.

4.5.3 The SALMO-OO Use-Case Diagrams

The SALMO-OO UML diagrams for use-case not only unambiguously establish the fundamentals to develop SALMO-OO and comply with its creator's intention but also formally clarify the general structure of the target SALMO-OO applications.

During object-oriented analysis phase, three types of actors have been discovered, including the User actor, the System actor, and the State Variable actor (Figure 4.5). Firstly, the User actor represents a person who accesses SALMO-OO via GUI, excluding any developers or administrators. Thus, the definition of the User actor limits a person access authorization to read only SALMO-OO.

Secondly, the System actor means a piece of the SALMO-OO programs or a person who operates these programs. The actions of the System actor involve in database transactions, fetching available modelling selections such as lakes, years, scenario analysis, and algae model libraries, and responding the actions of the User actor.

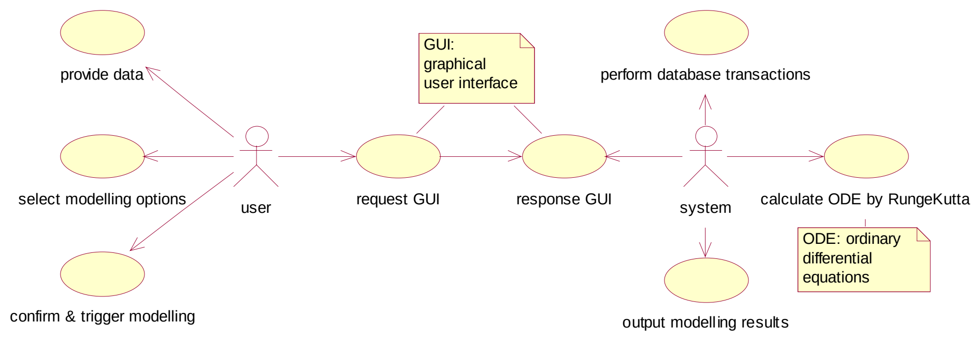


Figure 4.9 UML diagrams for use-case for the interaction between the user and SALMO-OO

A scenario descries the transactions between the User actor and the System actor. The initial step is a user locates the URL via IE in the Internet or open the modelling selection page in a single computer. The SALMO-OO applications display the initial page. Subsequently, the user completes a serial of selections and SALMO-OO responds to these relative selections until triggers Runge-Kutta calculations. Figure 4.10 exhibits this scenario.

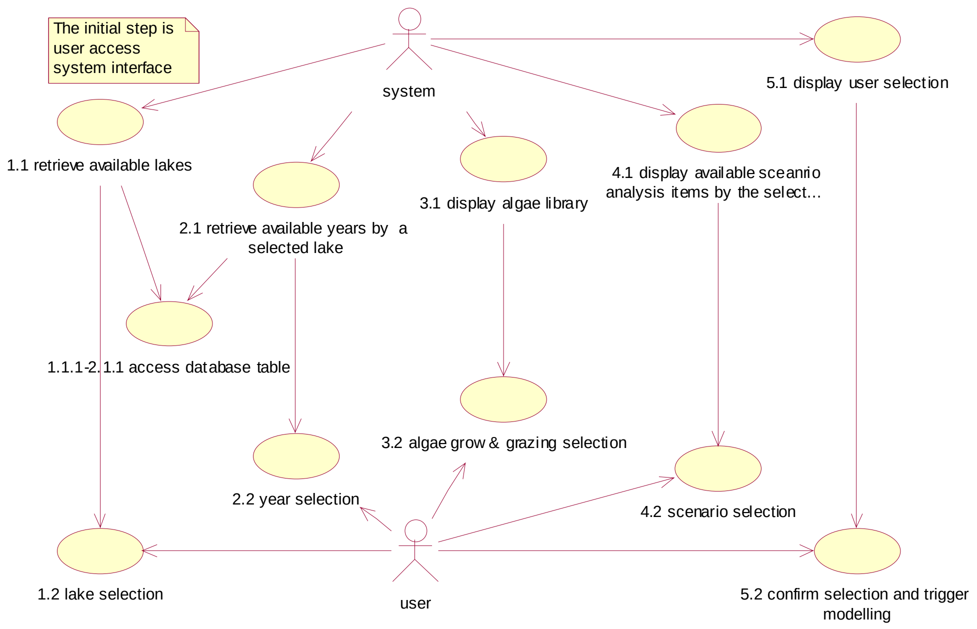

The initial step is user access system interface
system
5.1 display user selection
1.1 retrieve available lakes
2.1 retrieve available years by a selected lake
3.1 display algae library
4.1 display available sceanrio analysis items by the select...
1.1.1-2.1.1 access database table
3.2 algae grow & grazing selection
2.2 year selection
4.2 scenario selection
1.2 lake selection
user
5.2 confirm selection and trigger modelling

Figure 4.10 UML diagrams for use-case for the structuring of SALMO-OO by means of the model library and user selections

Thirdly, the State Variable actor represents the model SALMO state variables. Theoretically, the biological complexity results the sophisticate interactions between these state variables. As far as the detail modelling process is concerned, the complexity of the model SALMO comes out after the User actor triggers modelling. We have to answer at least three questions to figure out the Runge-Kutta calculations, the SALMO ODE, the data import and export, and the relationships among them. The first question asks which process is the subsequence among the Runge-Kutta calculations, the SALMO ODE, and data import after the User actor triggers modelling? Secondly, which process is the last sequence of them? The last question is how to use Runge-Kutta algorithm to calculate the SALMO ODE.

Obviously, the last question has more challenge than the other two. It can be predicted that the future problems will frequently come from the Runge-Kutta and the SALMO ODE calculations, therefore the solution of the last question paly a key role in the following use-case analysis. Fortunately, SALMO-OO does not offer the solutions of the Runge-Kutta algorithm, thus it is not necessary to model it by use-case diagrams. The analyses only focus on the SALMO ODE.

A recommendation to simplify the complex analysis could start from the simpleness. We take advantage of the materials such as the SALMO ODE documents and the model SALMO structure diagrams at this time. The SALMO ODE documents record every specific differential equation of the state variables. In general cases, the documents are organized by each state variable, and each of them is divided into two parts: mixed, epilimnion, and hypolimnion layer (note: the Oxygen state variable adds winter stagnation). Additional instruction of the model SALMO structure diagrams contributes the understanding of the relationships between the state variables and the rate variables. Thus, it is easy to clarify each specific actor and its actions that represent the SALMO ODE. Another advantage of this method is to reconstruct the model SALMO in terms of object-oriented paradigm.

An ecological food web scenario demonstrates the biological activities in the model SALMO. It can be understood from Figure 4.11 that the model SALMO starts from light, which more accurately describes underwater solar radiation. The importing of light and nutrient, including dissolved inorganic phosphate (DIP) and dissolved inorganic nitrate (DIN) provide rational natural conditions for algal growth, the factor of underwater temperature is excluded because it is defined as a part of the algal growth equations in SALMO-OO. The consequence of algal growth provides food for zooplankton growth by grazing. On the other hand, the dead algae and zooplankton become detritus, and detritus contributes phosphorus and nitrogen remineralization. By chemical combination, phosphorus and nitrogen are oxygenised into DIP and DIN respectively. As a cycle, DIP and DIN are consumed by phytoplankton.

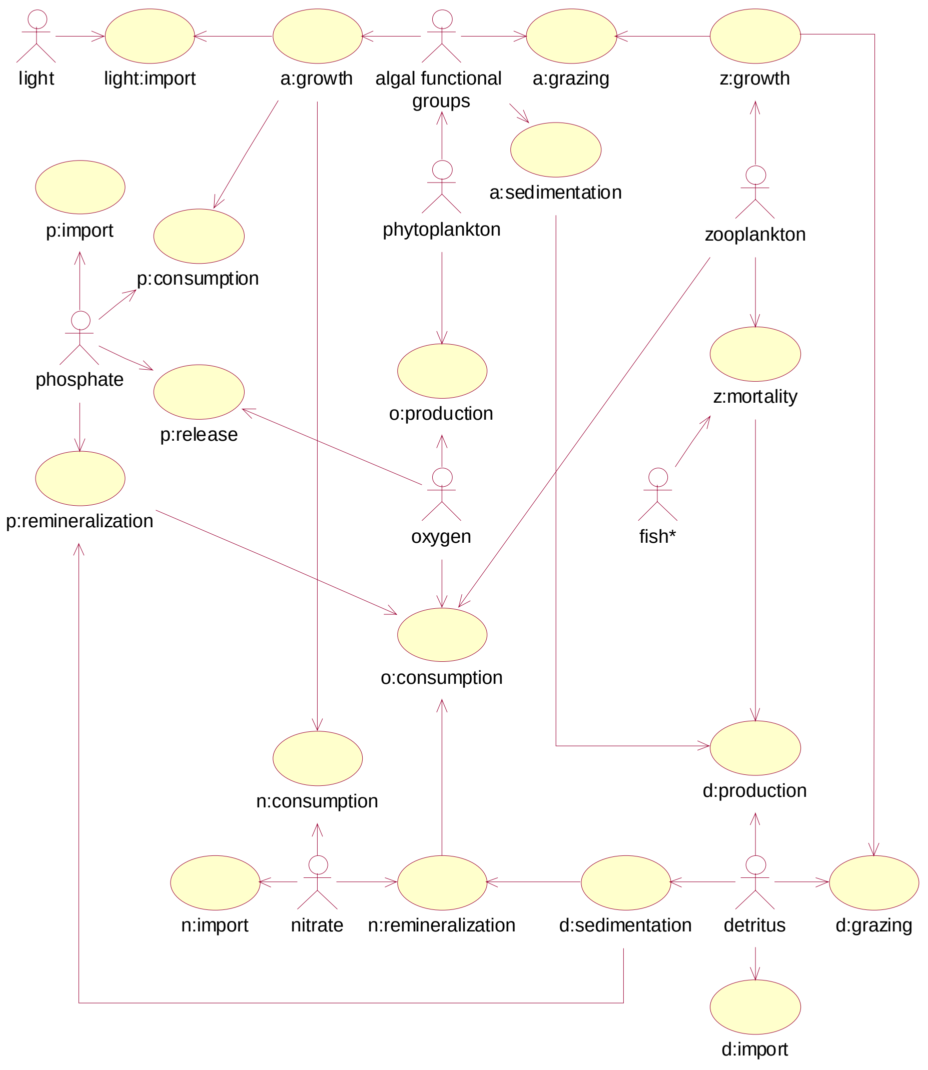


Figure 4.11 UML diagrams for use-case for the structure of the model SALMO

These three UML diagrams for use-case express our understanding of the model SALMO from different perspectives and establish a basic frame for SALMO-OO. Undoubtedly, the confirmation of these UML diagrams for use-case determines whether the next results proceed to our objectives.

4.5.4 The SALMO-OO Class Diagrams

The SALMO-OO UML class diagrams represent the static nature of SALMO-OO. In SALMO-OO, four packages that include twenty-five classes have been implemented.

The salmo.model package is shown in Figure 4.12. A rectangle represents a class, which comprises three parts with vertical layout: one class name lies in the top of the rectangle, zero or more class fields lies in the middle, and zero or more of the class methods lies in the bottom. The solid line with a closed empty arrow represents the inheritance relationships between two classes. It starts from the subclass (e.g. the Light class) and ends with the superclass (e.g. the Salmo class). The rectangle with a folded angle on the up right represents notes. A note can be connected to a class with dashed line.

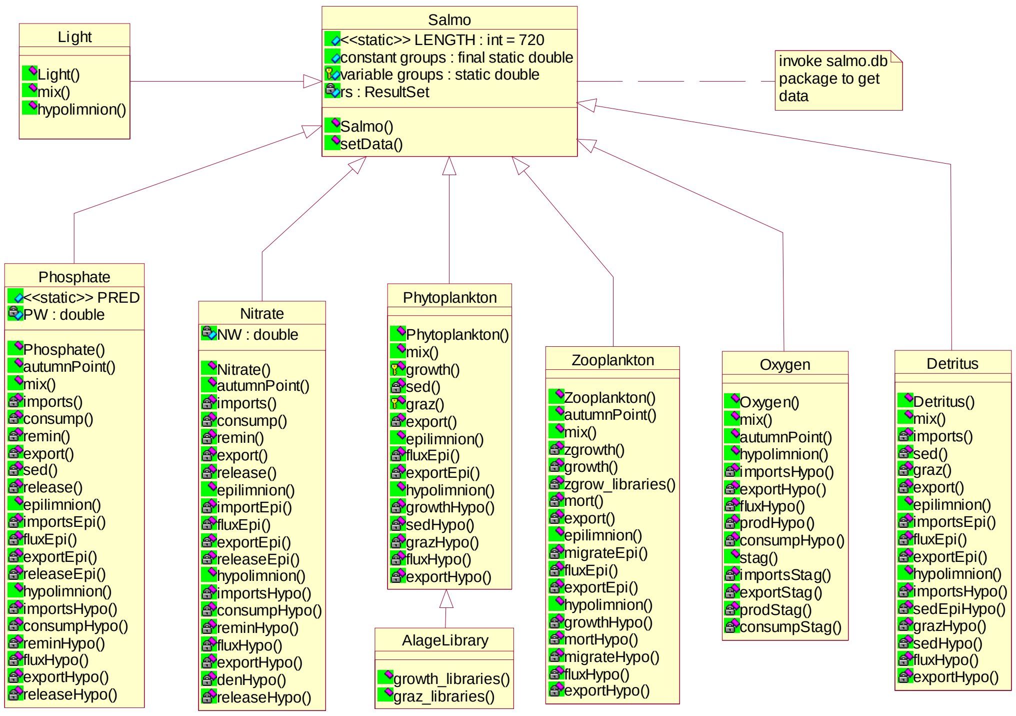


Figure 4.12 UML class diagrams for the model components of SALMO-OO

The salmo.maths package is shown in Figure 4.13. The dashed line with a closed empty arrowhead pointing at an interface (e.g. the DerivnFunction interface) represents the interface inheritance relationships between a class and an interface. The solid line with an open arrow represents the value passing relationships between two classes.

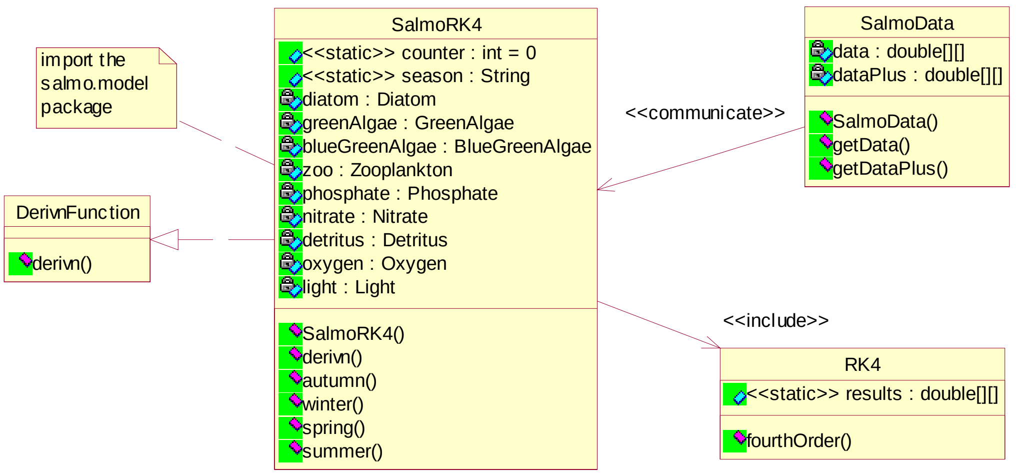


Figure 4.13 UML class diagrams for the mathematical operation of SALMO-OO

The salmo.gui package is shown in Figure 4.14.

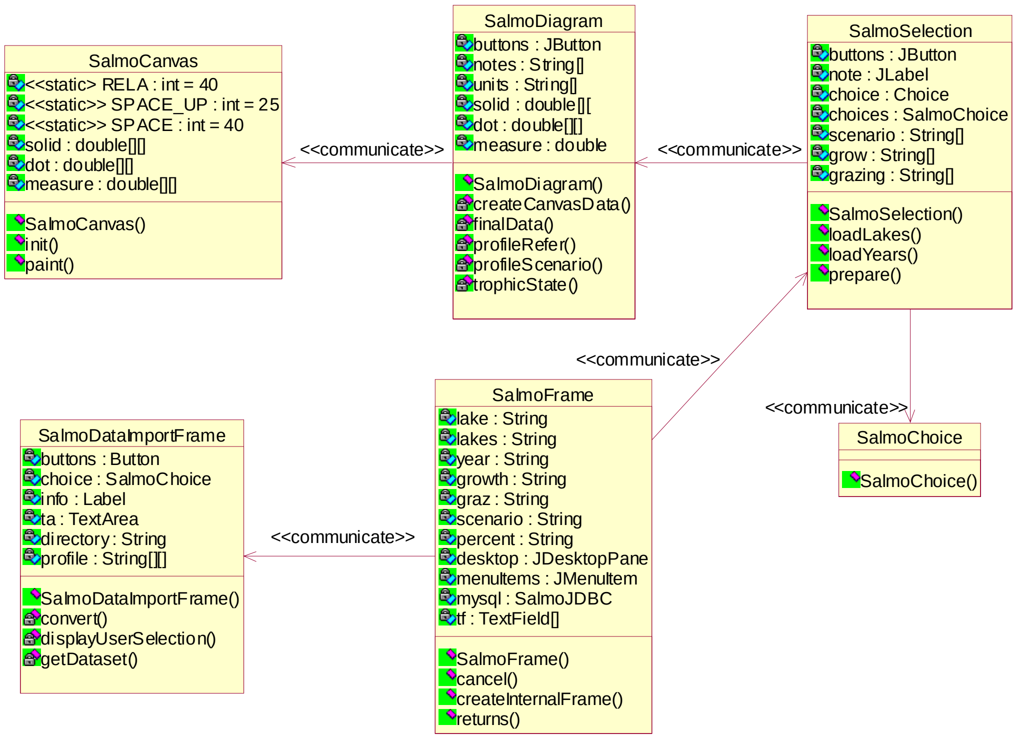


Figure 4.14 UML class diagrams for the graphical user interface of SALMO-OO

The salmo.db package is shown in Figure 4.15.

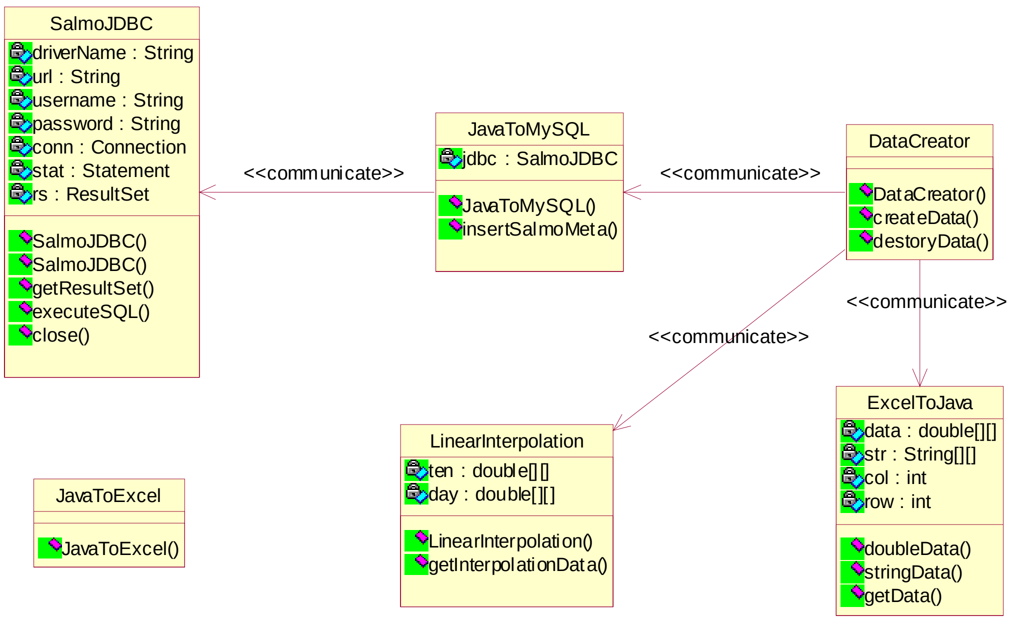


Figure 4.15 UML class diagrams for the lake database of SALMO-OO

4.5.5 The SALMO-OO Sequence Diagrams

The SALMO-OO UML sequence diagrams indicate the dynamic nature of SALMO-OO network edition. From the developer's point of view, this diagram represents the whole SALMO-OO process. This process starts from the User actor via presentation tier transactions, model tier calculations, data tier transactions, then returns by counter order, and finally ends when the User actor exits SALMO-OO. Figure 4.16 illustrates this process (note: this UML sequence diagrams only simulates the SALMO-OO network edition). The rectangle shown in this figure represents an application, which can be an actor, class, program, or a package.

First of all, the User actor requests the InputHTML web page. The details in this step start from the InputHTML web page and then transmit this request to the Servlet class. Then the Servlet class invokes the SalmoJDBC class in the salmo.db package in order to extract the available SALMO-OO data information. Subsequently, the Servlet class output these data information via the InputHTML and display them to the User actor. This is the first process called modelling preparing.

Secondly, the User actor needs to complete a series of tasks before the actual modelling can proceed. Meanwhile, the InputHTML web page displays the available SALMO-OO data information in the form of dropdown web component. These data information encompass various modelling options such as lakes, corresponding years, model libraries, scenario analyses, and parameters. The User actor can select only one item in each of these options, then is intent to submit the selections. The modelling will actually start after the User actor’s confirmation.

The third step is to complete the Runge-Kutta and ODE calculations. Joining the last step, the Servlet class transmits the modelling request and invokes the SalmoData class in the salmo.maths package in order to get the calculated data. Instead of direct calculations, the SalmoData class invoke the RK4, the SalmoRK4, and the ODE classes in the salmo.model package in turn, which these classes have responsibility to perform Runge-Kutta and ODE calculations. During this step, the Salmo class imports the measured data. If this step runs successfully, the SalmoData class stores the final calculated data.

Finally, SALMO-OO outputs the results to the User actor. The detail process involves the data passing between the SalmoServlet class and the SalmoApplet class. An OutputJSP web page lies in the middle of these two classes and aims to transmit the final calculated data value with a JSP Session object. The collaborations of three of them contribute to the task of data passing from the server side to the client side through the Internet. As the client side class, the SalmoApplet class invokes the GUI classes in the salmo.gui package. As a result, the classes in the salmo.gui package finally draw various diagrams that display in a computer screen that enable the User actor to observe the state variable dynamics.

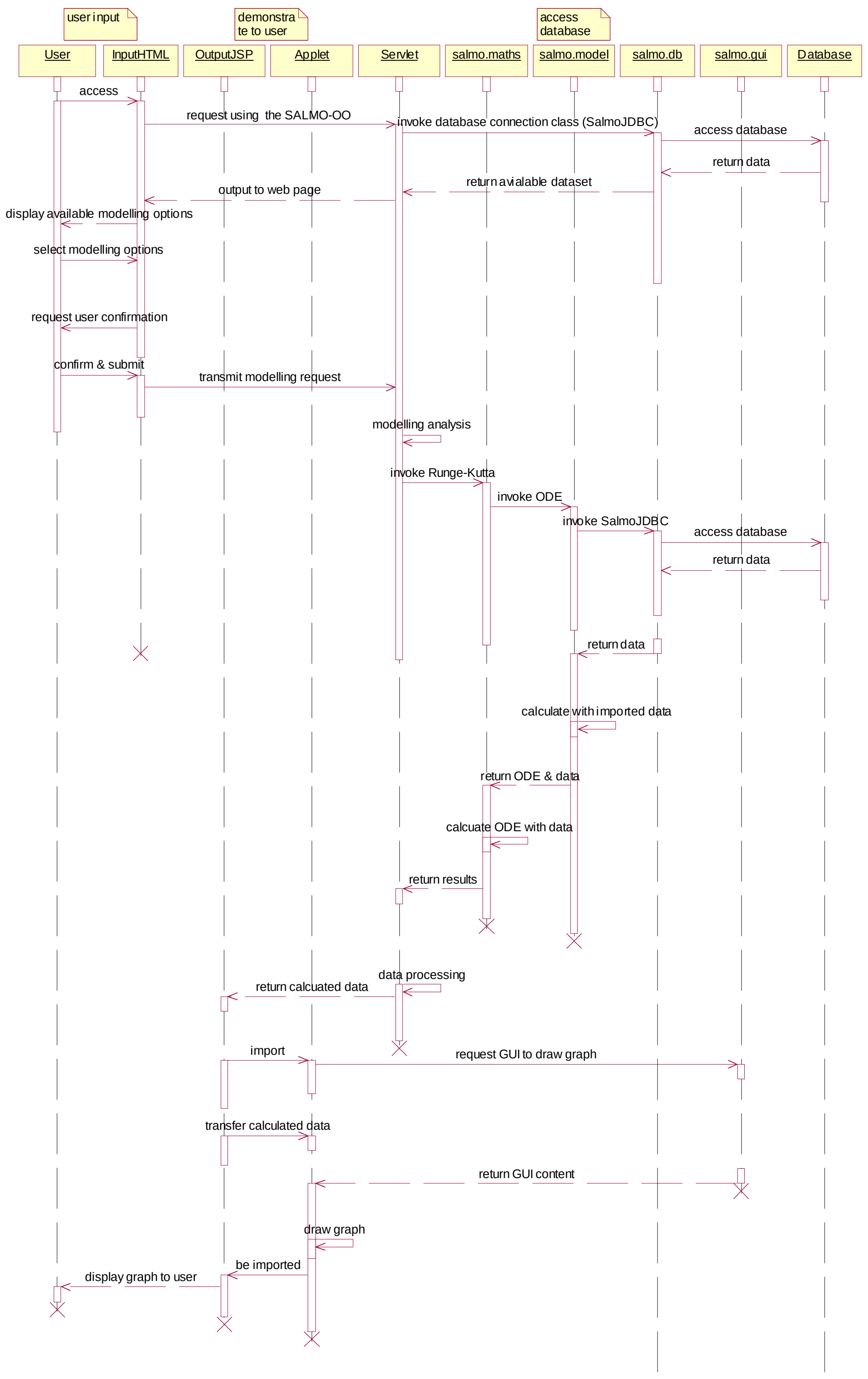


Figure 4.16 UML sequence diagrams for the application of SALMO-OO

4.5.6 The SALMO-OO Communication Diagrams

The SALMO-OO UML communication diagrams aim to model the dynamic nature of SALMO-OO stand-alone edition. Figure 4.13 shows the details. A rectangle in the UML communication diagrams represents an instance of a class, which is an object. A dashed arrow with a number represents a step, which is a method. Figure 4.17 lists the specific SALMO-OO applications as well as their relationships and interactions.

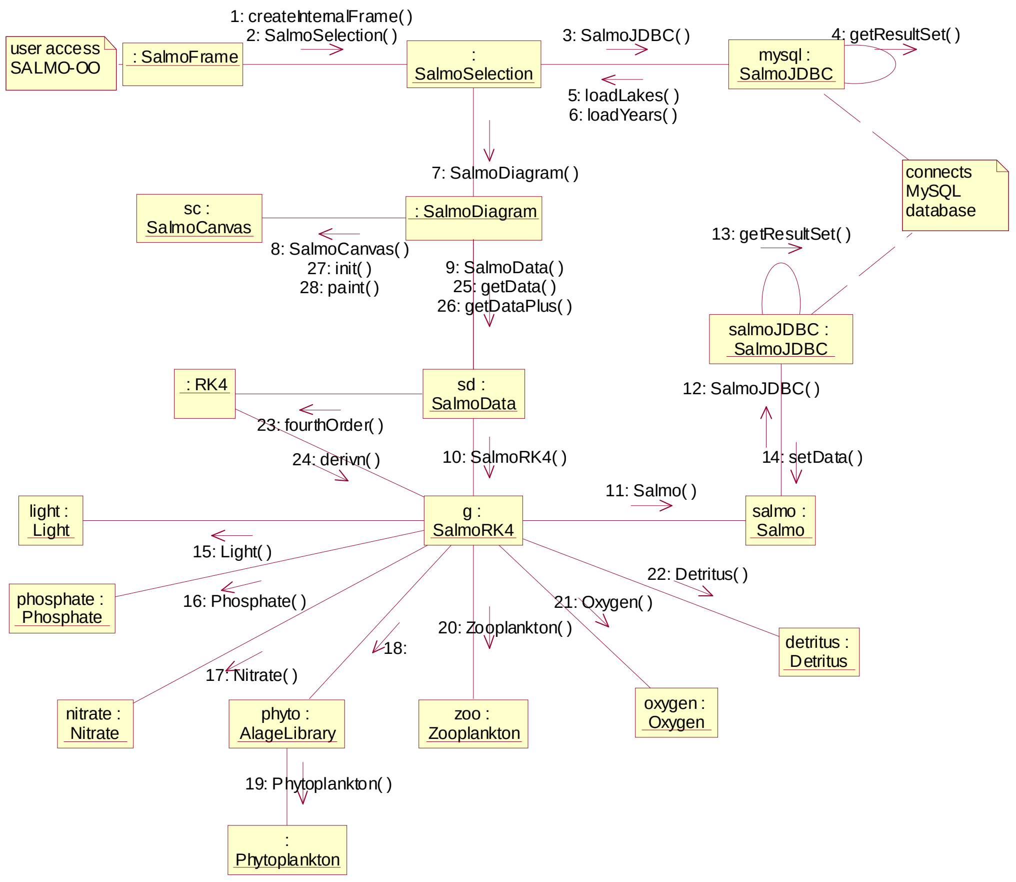


Figure 4.17 UML communication diagrams for the application SALMO-OO

There is more information behind Figure 4.17. It is invaluable to display too many details because the omissions play secondary role in the SALMO-OO communication diagrams. The omissive information involves the process of the distributive seasons, the water volume, and the additional model libraries. It also ignores the relationships between the RK4 class and the SalmoRK4 class. For example, it neither takes into account how the SalmoRK4 class implements the DerivnFunction interface since interfaces do construct any instance, nor indicates how many times that an object of the RK4 class invokes one of the SalmoRK4 class. Obviously, a RK4 object invokes a SalmoRK4 object for 1440 times during 360 differential steps because SALMO-OO uses fourth order Runge-Kutta algorithm.

In addition to the conventional usage, the SALMO-OO UML communication diagrams introduce a new procedure. According to the results in the OOA phase, there are 19-shared variables in the salmo.model package. These variables belong to the objects that represent the state variables. Although the SALMO ODE document records their location in a few equations, it is still difficult to figure out which object is involved in calculating one of these variables. In fact, diagrams can solve this problem. We use UML communication diagrams to represent this logic. Figure 4.18 shows the object rectangle and the note rectangle, which respectively symbolize the objects and the shared variables. It can be shown in Figure 4.18 that a variable needs to be calculated by an object if a dashed line connects them. The Light object does not calculate any of these variables therefore it is isolated. Furthermore, it is clearly shown in Figure 4.14 how many objects need to share a variable’s calculation.

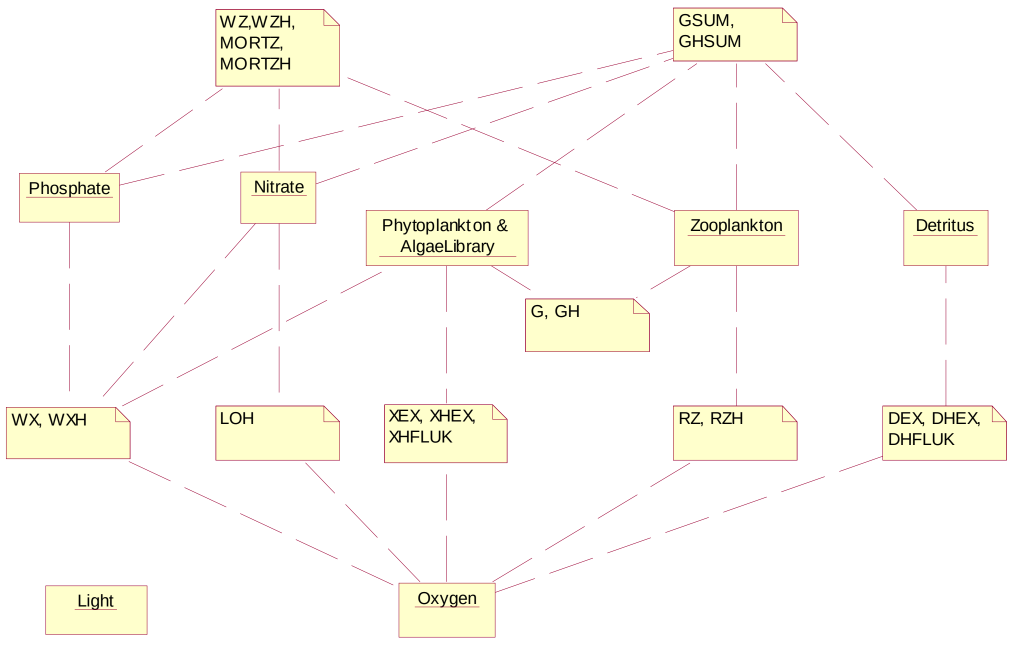


Figure 4.18 A UML communication diagrams for the relationships between the objects and the shard variables

It can be stated that the complete UML communication diagrams are one of the milestone in the SALMO-OO life cycle. Often this is the end of the object-oriented design phase. With the guide of these UML diagrams, Java programming language can implement the SALMO-OO class library in a short time. In general sense, a well-designed software structure will free the person who types source code to a large degree. However, object-oriented programming strongly relies on a programmer's experience and skill.

At this stage, a conclusion can be drawn for both the SALMO-OO UML sequence diagrams and SALMO-OO communication diagrams. On one hand, the purpose of the

SALMO-OO UML sequence diagrams is to figure out the flow logic between the SALMO-OO applications on the top level. As normal understanding process, it is more important for us to firstly understand how the message flows between the top-level components than the low-level classes. Thus, the SALMO-OO UML sequence diagrams focus on the general message flow in the network edition rather the details in the stand-alone edition because the network edition contains a large amount of complex user transactions. On the other hand, the SALMO-OO UML communication diagrams deal with the detail message flow as a compensative solution. These details dominate the flow logic in SALMO-OO stand-alone edition. Therefore, the collaboration of these two diagrams clarifies the interactions between the SALMO-OO applications in the OOD phase.

4.5.7 The SALMO-OO UML diagrams for deployment

The target SALMO-OO applications need deployment to be available. The components of deployment encompass hardware devices, operating systems, SALMO-OO suites, and various supporting software.

SALMO-OO involves two deployment strategies. One is the stand-alone edition deployment. The stand-alone edition requires the SALMO-OO suites to be installed in a single computer but does not involve any transactions across the computer network. A single computer can be any computer that has ability to run the SALMO-OO suites. For example, this single computer can be a desktop, laptop, or mainframe regardless of connecting to the Internet, but it has to meet the minimum configuration. Actually, most of the computers produced after 2000 are able to run the SALMO-OO suites.

Figure 4.19 uses UML diagrams for deployment to show the details. It can be seen from this figure that all the components are deployed in one computer device.

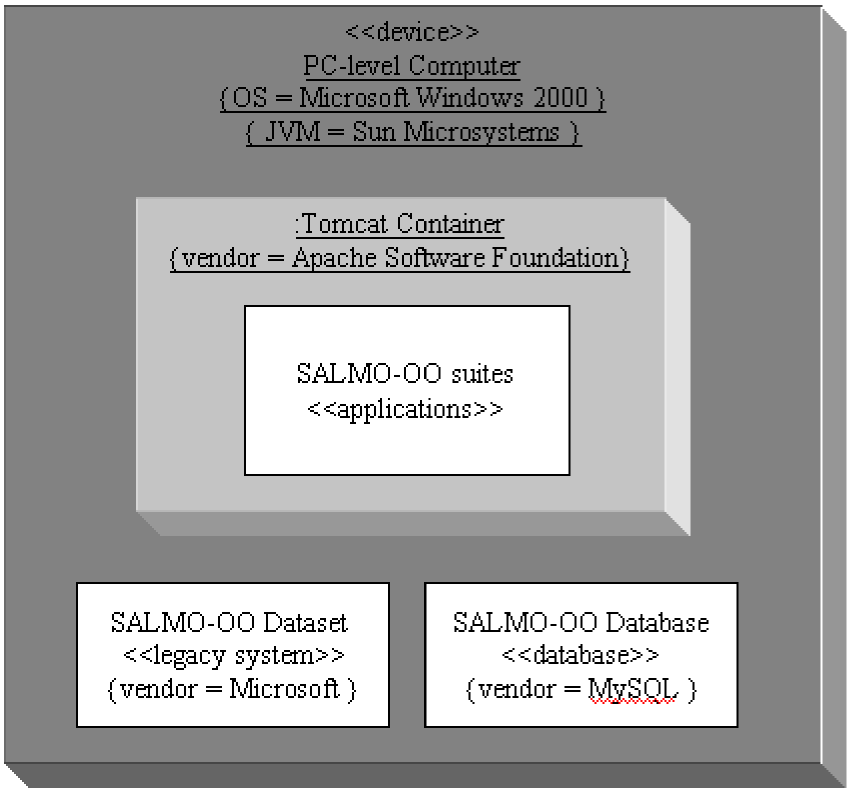


Figure 4.19 UML diagrams for deployment for the stand-alone version of SALMO-OO

Another is the network version deployment. The network version requires the SALMO-OO suites to be installed in the computers that connect to the Internet. The specific deployment strongly depends on the available network structure. Generally, the SALMO-OO suites distribute the class library and database into two hardware devices. Moreover, it requires the client terminals to install Internet Explorer embedded Java Runtime Environment that enable to display Applet. Figure 4.20 shows these configurations.

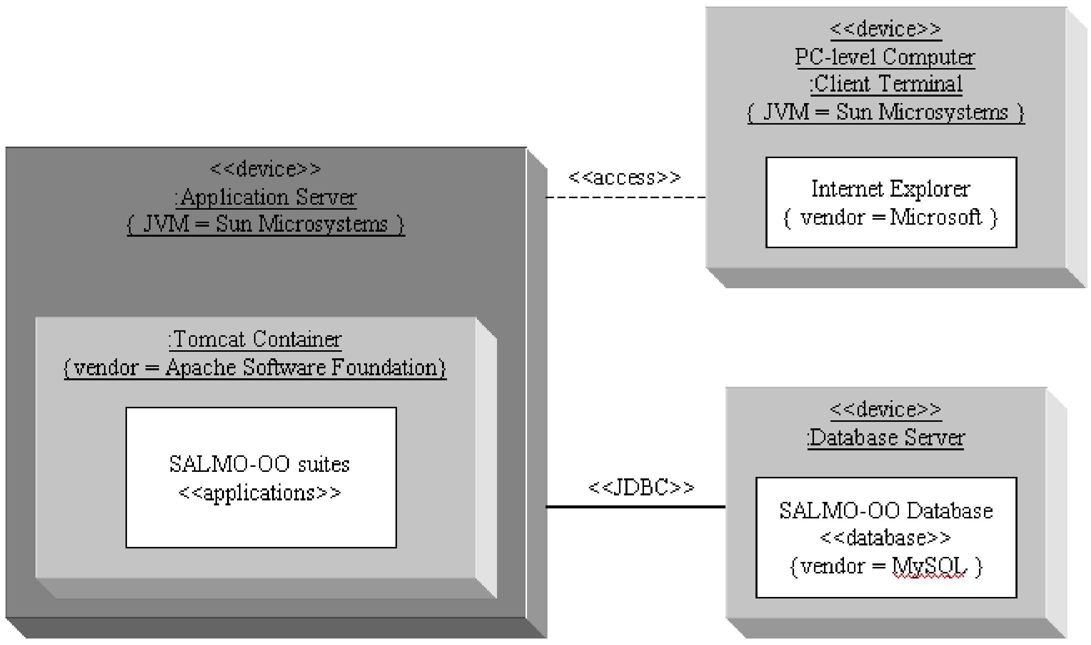


Figure 4.20 UML diagrams for deployment for network version of SALMO-OO

In addition to these UML diagrams for deployment, it is necessary to display further deployment of the SALMO-OO suites in the Tomcat container. Figure 4.21 shows the hierarchy in the form of tree graph. A rectangle with underline text represents a directory, and a rectangle with plain text represent one or more files.

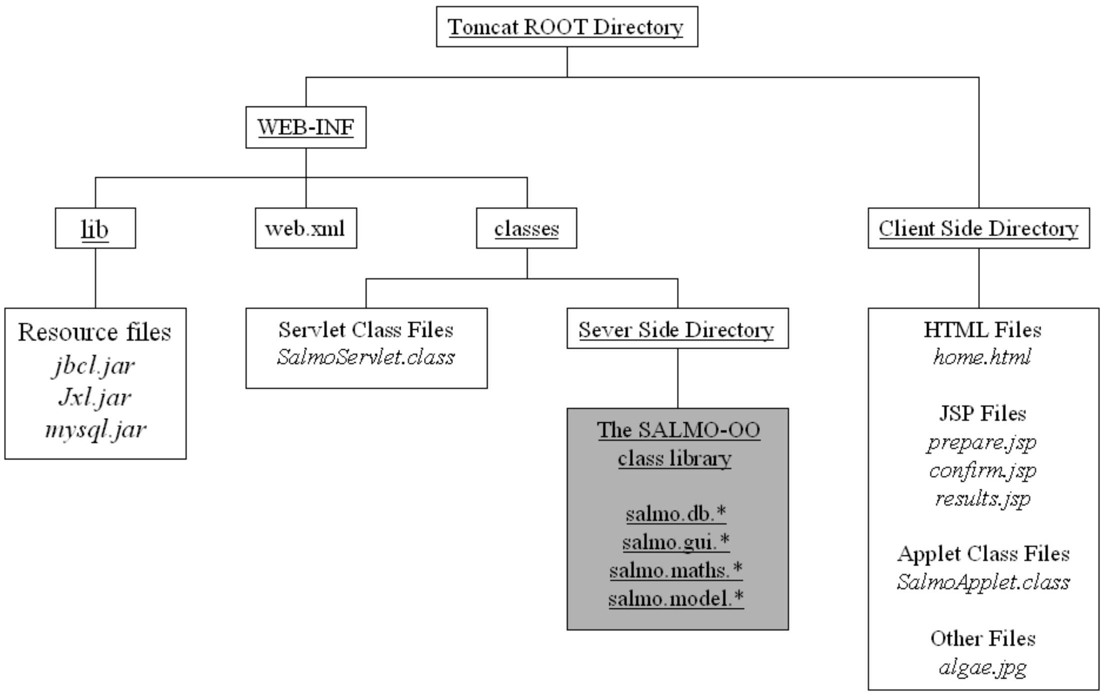


Figure 4.21 Suites file hierarchy of SALMO-OO

Tomcat 4 and JRE 1.5 are required for both the stand-alone and network version of SALMO-OO.

4.6 The SALMO-OO API Specification

The SALMO-OO API specification provides the detail instructions for the SALMO-OO class library. As an important document, this API specification reflects the structure of the core SALMO-OO applications, including packages, classes, fields, and methods. It is organized in the form of web pages that comply with the standard of Java code conventions (Sun Microsystems 1994-2005b).

Chapter 5

Discussion

5.1 SALMO-OO can be implemented by Object-Oriented programming using Java

SALMO-OO achieves the same results as the original Fortran IV version of SALMO. Figure 5.1 illustrates the simulation results of the Fortran IV version of SALMO for phosphate, phytoplankton and zooplankton by means of data from Lake Stechlin 1975, Saidenbach Reservoir 1975, and Bautzen Reservoir 1978.

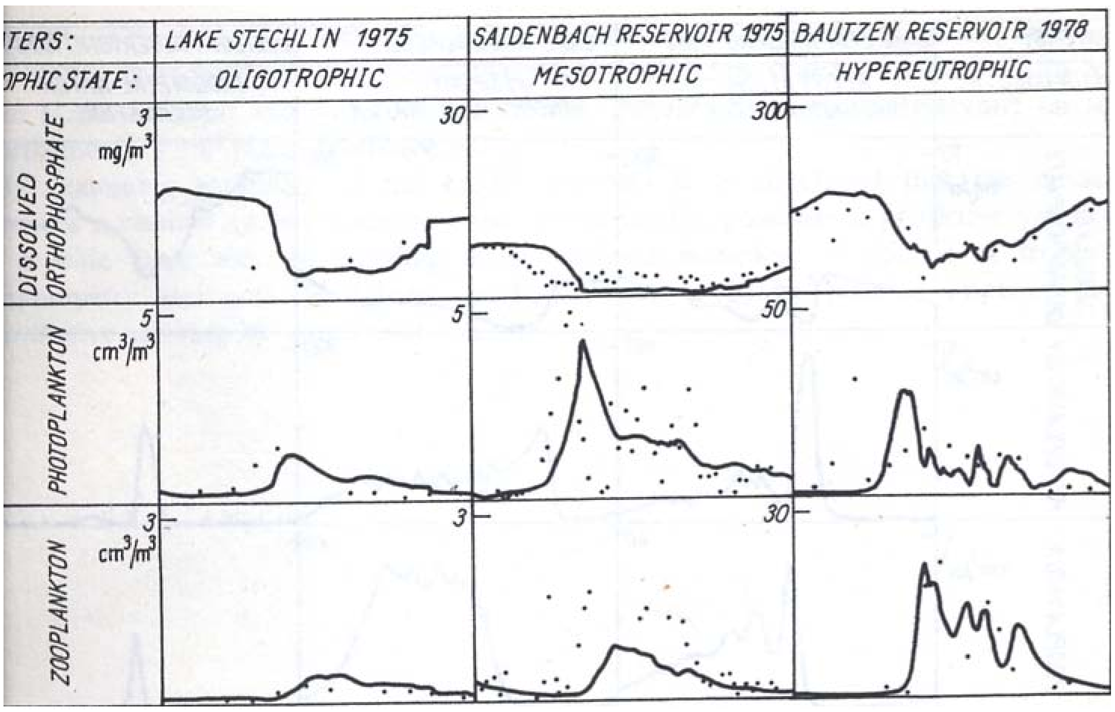


Figure 5.1 Simulation results by the FORTRAN IV version of SALMO for the Lake Stechlin 1975, Saidenbach Reservoir 1975 and Bautzen Reservoir 1978

The second prototype system exactly reconstructs SALMO but is implemented in Java programming language. The new results add Chlorophyll-a state variable and simulate epilimnion and hypolimnion together, excluding Lake Stechlin 1975.

Figure 5.2 shows the result of the second prototype system that simulates Saidenbach Reservoir 1975.

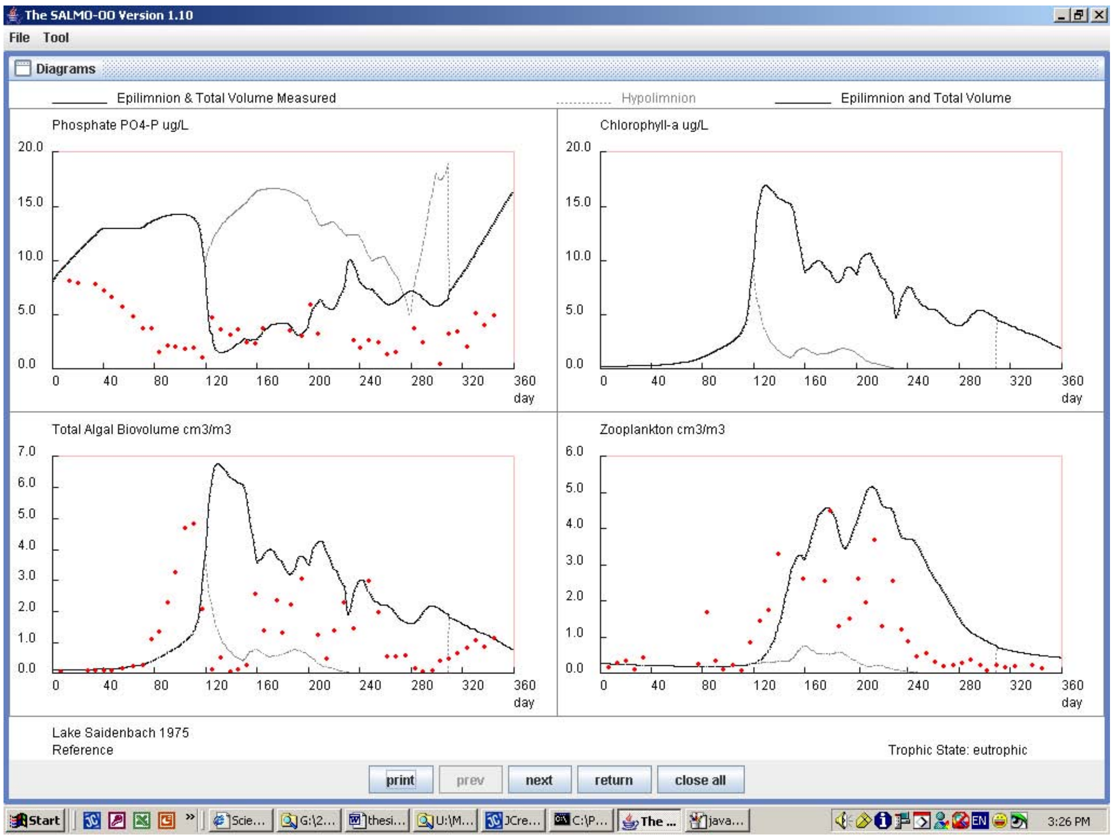


Figure 5.2 Simulation results by SALMO-OO for Saidenbach Reservoir 1975

Comparing to the Fortran result of Saidenbach Reservoir 1975 in the middle of Figure 5.1, it is obvious that Figure 5.2 has higher daily dynamics than Figure 5.1 in DIP, phytoplankton, and zooplankton. This difference comes from using wrong parameters.

After correcting the mistakes, the result of Bautzen Reservoir 1975 is far similar to the Fortran one (Figure 5.3).

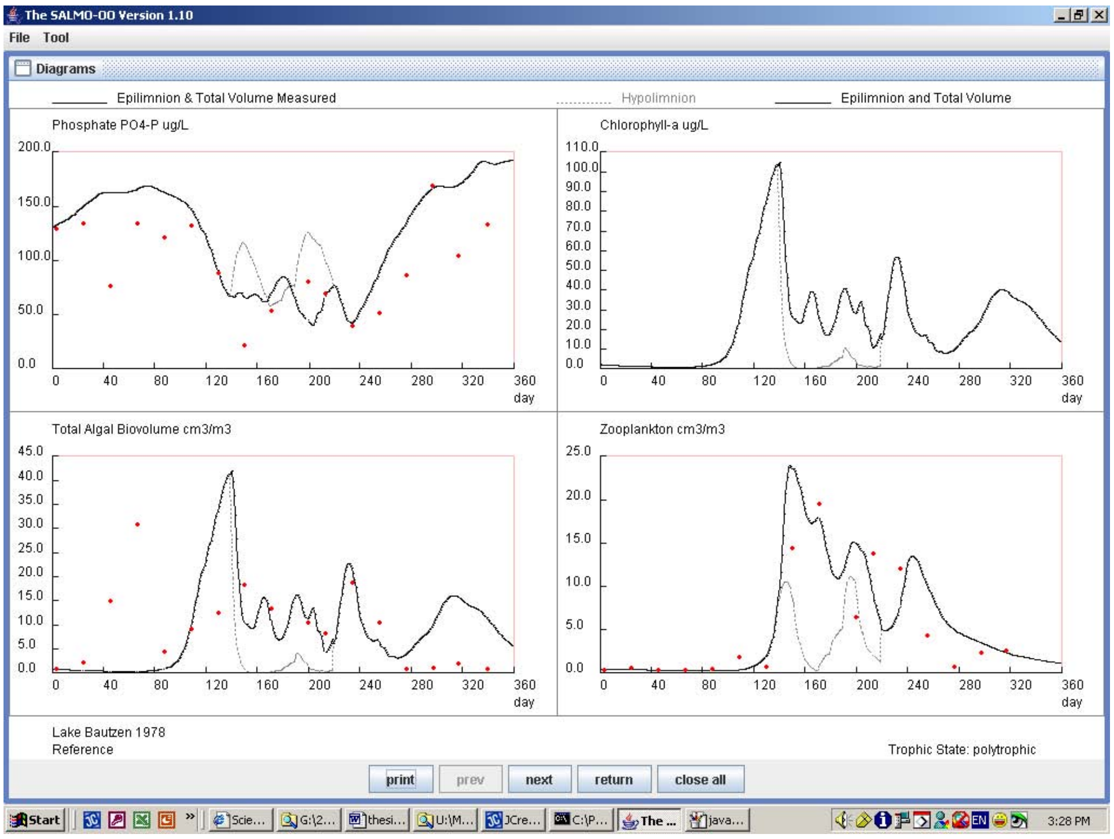


Figure 5.3 Simulation results by SALMO-OO for Bautzen Reservoir 1978

It is clear that using object-oriented technology can produce the same model as before. However, the second prototype system is not good enough to show the strength of object-oriented technology. Our project further enforces the model SALMO in the third prototype system, which is the final version of SALMO-OO. It achieves more accurate results than those previous ones, and optimises the structures of the SALMO-OO class library. These improvements realize the possibility to add an extra algal functional group; therefore SALMO-OO can simulate Diatoms, Green Algae, and Blue Green Algae at the same time. Moreover, SALMO-OO offers a new function that permits users to adjust the primary modelling parameters. Thus, ecologists can flexibly select the SALMO parameters to pursue any perspective result. Fortunately, all these processes are happened in an easy manner.

The SALMO-OO achievements benefit from the object-oriented technology. In others words, the model SALMO is standardised by means of object-oriented modelling such as UML modelling. However, the Fortran version of SALMO follows the structural paradigm, which uses flow chart to simulate the modelling process. Although the Fortran one has a well-design structure (Figure 1 and Figure 2 in Section 1.3.1), it did not describe it in a simple modelling language, which results in the complexity of SALMO. As a result, neither modellers can directly clarify the structure, nor programming languages can easily implement the relative applications.

On the other hand, the SALMO-OO class library solves the problems by means of object-oriented modelling. As far as the performance is concerned, the realizations of flexibility, reusability, and platform independence enable the SALMO-OO class library to be accessed more freely than before. The flexible SALMO-OO class library takes advantage of object-oriented analysis and design. Specifically, UML modelling achieves the aims. SALMO-OO provides transparent program structure against its complexity. The previous SALMO model is broken up into pieces that represented by objects in the OOA phase. The merit of object orientation provides a beneficial way to simplify the complexity of the model SALMO because everything originates from an atomic particle. Theoretically, an atomic particle is the minimum understandable unit in the human mind. In the context of object-oriented paradigm, this atomic particle means object. Subsequently, OOD discovers the relationships and interactions between these objects and represents the details by various UML diagrams. These two processes finally model the SALMO-OO requirement, structure, and logic in a simple way. Moreover, the well-designed structure benefits the reusability of the class library. The advantage of reusability not only reduces a large amount of source code but also simplify the implementation of target SALMO-OO. In addition, the SALMO-OO class library is independent to the runtime environment, including hardware devices, operating systems, and databases. Undoubtedly, Java technology plays a key role in the platform independence. In general, UML modelling provides one of the most representative methods standardise the model SALMO and data structure, which delivers the aims of data and model sharing by standardised model and data structures.

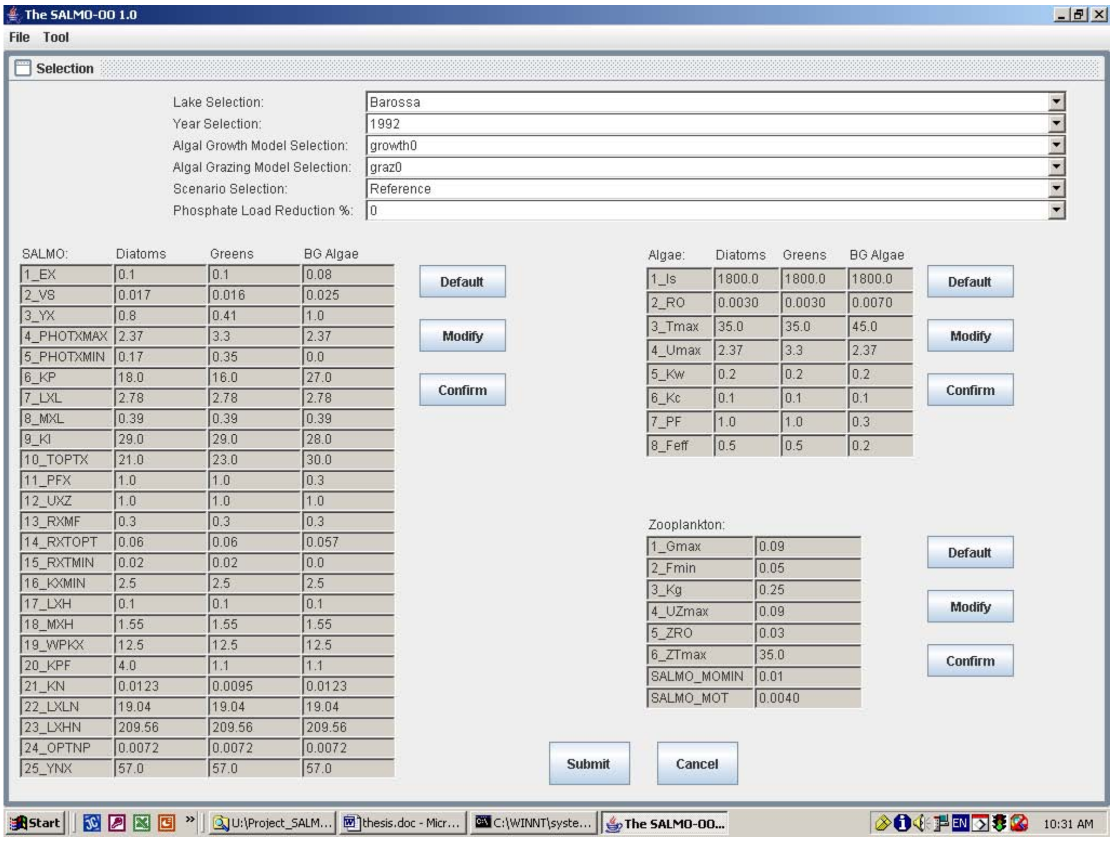
Since SALMO-OO can be implemented by object-oriented technology, it also is capable to pursue more advanced applications such as web accessibility, even to be built accompanying with other application-level software systems such as Graphical Information System (GIS). The latter is supposed to upgrade SALMO-OO on the spatial dimension. Therefore, using object-oriented technology solves the key of SALMO-OO as well as aims to future considerations.

5.2 Users can access SALMO-OO via Internet as well as friendly GUI

Web accessibility and friendly GUI is the second achievement of SALMO-OO.

Although SALMO-OO considers both stand-alone edition and network edition, the core class library is exactly same between two editions. The difference between them only involves part of GUI components, but does not has any dissimilar operations for users. To clarify the visual effect, both editions demonstrate an example.

First, SALMO-OO stand-alone edition GUI is shown in Figure 5.4.



Lake Selection: Barossa
Year Selection: 1992
Algal Growth Model Selection: growth0
Algal Grazing Model Selection: graz0
Scenario Selection: Reference
Phosphate Load Reduction %: 0

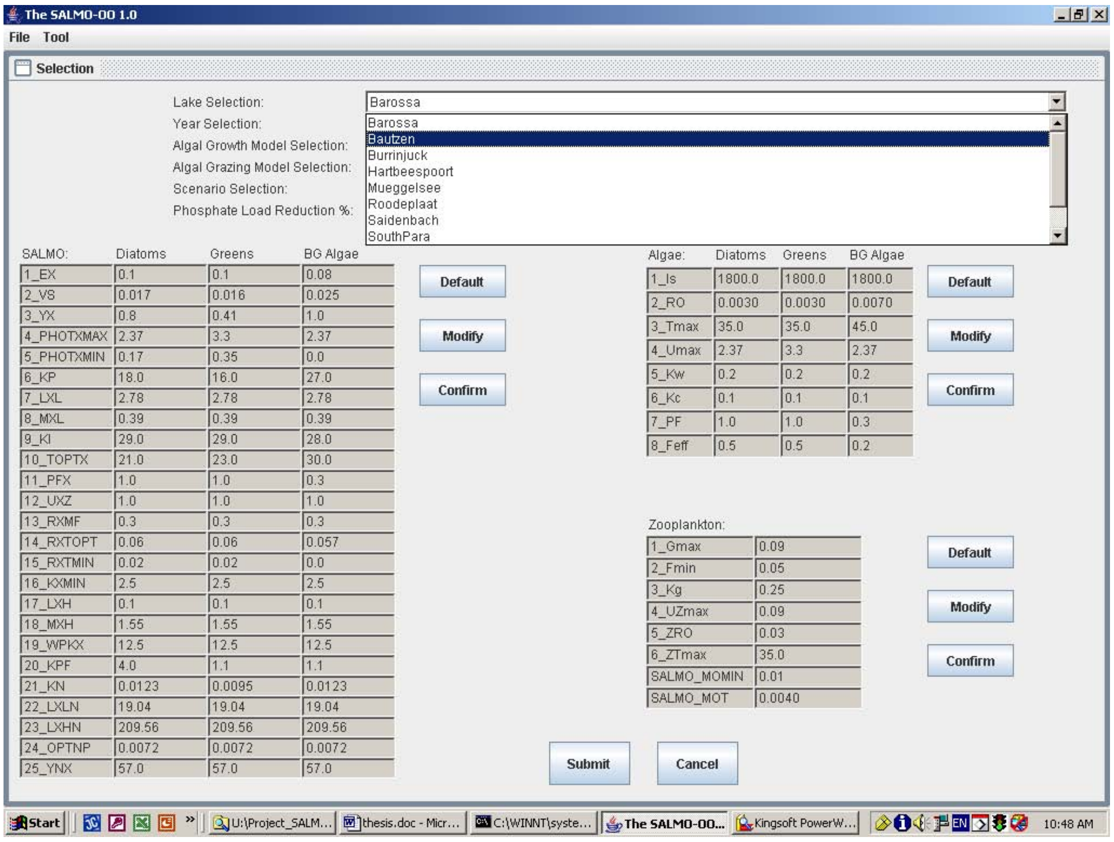
SALMO:	Diatoms	Greens	BG Algae
1_EX	0.1	0.1	0.08
2_VS	0.017	0.016	0.025
3_YX	0.8	0.41	1.0
4_PHOTXMAX	2.37	3.3	2.37
5_PHOTXMIN	0.17	0.35	0.0
6_KP	18.0	16.0	27.0
7_LXL	2.78	2.78	2.78
8_MXL	0.39	0.39	0.39
9_KI	29.0	29.0	28.0
10_TOPTX	21.0	23.0	30.0
11_PFX	1.0	1.0	0.3
12_UXZ	1.0	1.0	1.0
13_RXMF	0.3	0.3	0.3
14_RXTOPT	0.06	0.06	0.057
15_RXTMIN	0.02	0.02	0.0
16_KXMIN	2.5	2.5	2.5
17_LXH	0.1	0.1	0.1
18_MXH	1.55	1.55	1.55
19_WPKX	12.5	12.5	12.5
20_KPF	4.0	1.1	1.1
21_KN	0.0123	0.0095	0.0123
22_LXLN	19.04	19.04	19.04
23_LXHN	209.56	209.56	209.56
24_OPTNP	0.0072	0.0072	0.0072
25_YNX	57.0	57.0	57.0

Algae:	Diatoms	Greens	BG Algae
1_Is	1800.0	1800.0	1800.0
2_RO	0.0030	0.0030	0.0070
3_Tmax	35.0	35.0	45.0
4_Umax	2.37	3.3	2.37
5_Kw	0.2	0.2	0.2
6_Kc	0.1	0.1	0.1
7_PF	1.0	1.0	0.3
8_Feff	0.5	0.5	0.2

Zooplankton:	
1_Gmax	0.09
2_Fmin	0.05
3_Kg	0.25
4_UZmax	0.09
5_ZRO	0.03
6_ZTmax	35.0
SALMO_MOMIN	0.01
SALMO_MOT	0.0040

Figure 5.4 Initial GUI of the stand-alone version of SALMO-OO

Figure 5.4 consists of three parts. The first part is the SALMO-OO modelling options, which lies in the top of Figure 5.4. SALMO-OO provides lake, year, algal growth, algal grazing, and scenario selections. User can freely chose any combination among them. The second part is the SALMO-OO modelling parameters, which lies in the middle of Figure 5.4. Currently, SALMO-OO provides the whole algal parameters and part of zooplankton parameters, including additional model library parameters for algal growth and algal grazing. The third part is the GUI functional area, which lies in the bottom of Figure 5.4. Users can submit modelling request by press the Submit button, or cancel it. For example, a user clicks the Lake Selection option, the system responses to list all the available lakes or reservoirs (Figure 5.5). Responsively, the user chooses the Bautzen item and the system vary it into blue background.



Lake Selection: Barossa
Year Selection:
Algal Growth Model Selection:
Algal Grazing Model Selection:
Scenario Selection:
Phosphate Load Reduction %:

Barossa
Bautzen
Burrinjuck
Hartbeespoort
Mueggelsee
Roodeplaat
Saidenbach
SouthPara

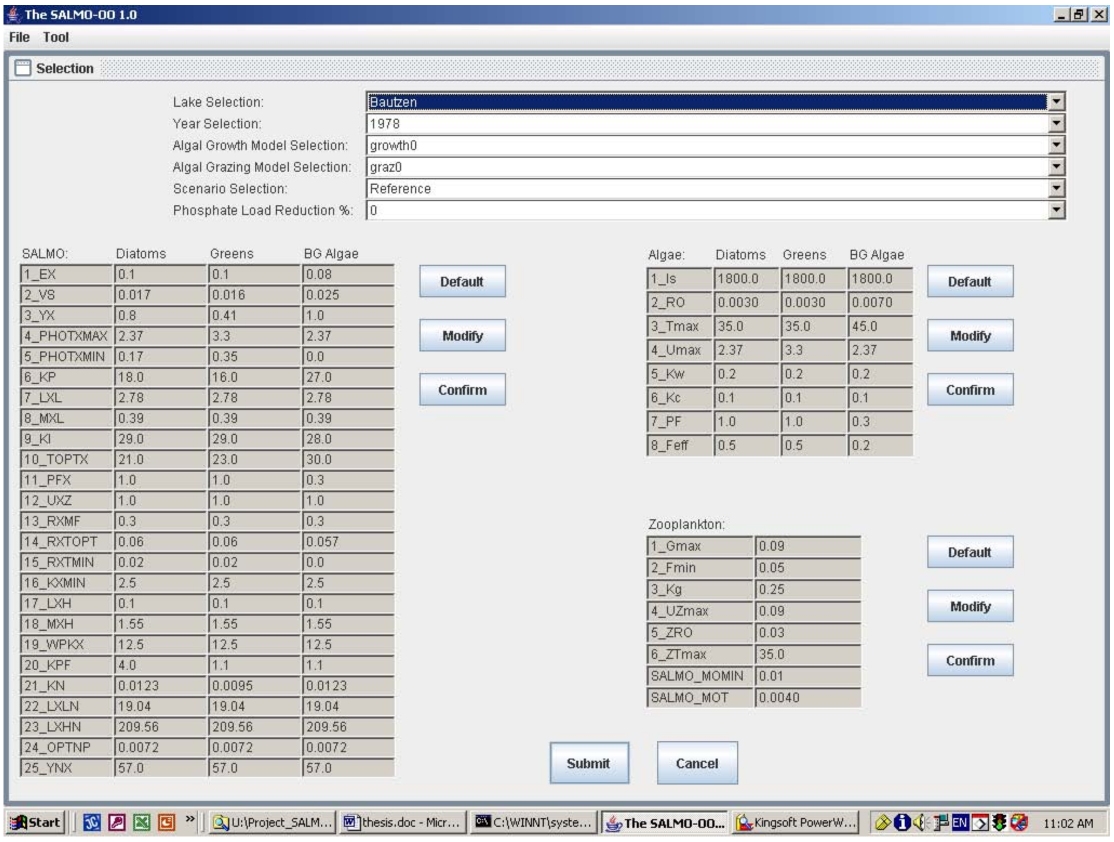
SALMO:	Diatoms	Greens	BG Algae
1_EX	0.1	0.1	0.08
2_VS	0.017	0.016	0.025
3_YX	0.8	0.41	1.0
4_PHOTXMAX	2.37	3.3	2.37
5_PHOTXMIN	0.17	0.35	0.0
6_KP	18.0	16.0	27.0
7_LXL	2.78	2.78	2.78
8_MXL	0.39	0.39	0.39
9_KI	29.0	29.0	28.0
10_TOPTX	21.0	23.0	30.0
11_PFX	1.0	1.0	0.3
12_UXZ	1.0	1.0	1.0
13_RXMF	0.3	0.3	0.3
14_RXTOPT	0.06	0.06	0.057
15_RXTMIN	0.02	0.02	0.0
16_KXMIN	2.5	2.5	2.5
17_LXH	0.1	0.1	0.1
18_MXH	1.55	1.55	1.55
19_WPKX	12.5	12.5	12.5
20_KPF	4.0	1.1	1.1
21_KN	0.0123	0.0095	0.0123
22_LXLN	19.04	19.04	19.04
23_LXHN	209.56	209.56	209.56
24_OPTNP	0.0072	0.0072	0.0072
25_YNX	57.0	57.0	57.0

Algae:	Diatoms	Greens	BG Algae
1_Is	1800.0	1800.0	1800.0
2_RO	0.0030	0.0030	0.0070
3_Tmax	35.0	35.0	45.0
4_Umax	2.37	3.3	2.37
5_Kw	0.2	0.2	0.2
6_Kc	0.1	0.1	0.1
7_PF	1.0	1.0	0.3
8_Feff	0.5	0.5	0.2

Zooplankton:	
1_Gmax	0.09
2_Fmin	0.05
3_Kg	0.25
4_UZmax	0.09
5_ZRO	0.03
6_ZTmax	35.0
SALMO_MOMIN	0.01
SALMO_MOT	0.0040

Figure 5.5 Example for the selection of lake ‘Bautzen’ by means of the GUI stand-alone version of SALMO-OO

After release mouse, the Bautzen item displays in the Lake Selection option instead of the previous default one. The Year Selection option automatically displays the available years that respond to one selected lake (Figure 5.6). In this case, SALMO-OO only offers one year 1978 regrading to the Bautzen Reservoir. Subsequently, the user press the Submit button with other default options, which are the original SALMO-OO algal growth and grazing model without scenario analysis, therefore the system skip to the result GUI (Figure 5.7).



Lake Selection:	Bautzen
Year Selection:	1978
Algal Growth Model Selection:	growth0
Algal Grazing Model Selection:	graz0
Scenario Selection:	Reference
Phosphate Load Reduction %:	0

SALMO:	Diatoms	Greens	BG Algae
1_EX	0.1	0.1	0.08
2_VS	0.017	0.016	0.025
3_YX	0.8	0.41	1.0
4_PHOTXMAX	2.37	3.3	2.37
5_PHOTXMIN	0.17	0.35	0.0
6_KP	18.0	16.0	27.0
7_LXL	2.78	2.78	2.78
8_MXL	0.39	0.39	0.39
9_KI	29.0	29.0	28.0
10_TOPTX	21.0	23.0	30.0
11_PFX	1.0	1.0	0.3
12_UXZ	1.0	1.0	1.0
13_RXMF	0.3	0.3	0.3
14_RXTOPT	0.06	0.06	0.057
15_RXTMIN	0.02	0.02	0.0
16_KXMIN	2.5	2.5	2.5
17_LXH	0.1	0.1	0.1
18_MXH	1.55	1.55	1.55
19_WPKX	12.5	12.5	12.5
20_KPF	4.0	1.1	1.1
21_KN	0.0123	0.0095	0.0123
22_LXLN	19.04	19.04	19.04
23_LXHN	209.56	209.56	209.56
24_OPTNP	0.0072	0.0072	0.0072
25_YNX	57.0	57.0	57.0

Algae:	Diatoms	Greens	BG Algae
1_Is	1800.0	1800.0	1800.0
2_RO	0.0030	0.0030	0.0070
3_Tmax	35.0	35.0	45.0
4_Umax	2.37	3.3	2.37
5_Kw	0.2	0.2	0.2
6_Kc	0.1	0.1	0.1
7_PF	1.0	1.0	0.3
8_Feff	0.5	0.5	0.2

Zooplankton:	
1_Gmax	0.09
2_Fmin	0.05
3_Kg	0.25
4_UZmax	0.09
5_ZRO	0.03
6_ZTmax	35.0
SALMO_MOMIN	0.01
SALMO_MOT	0.0040

Figure 5.6 Example for the selection of year ‘1978’ of lake ‘Bautzen’ by means of the GUI stand-alone version of SALMO-OO

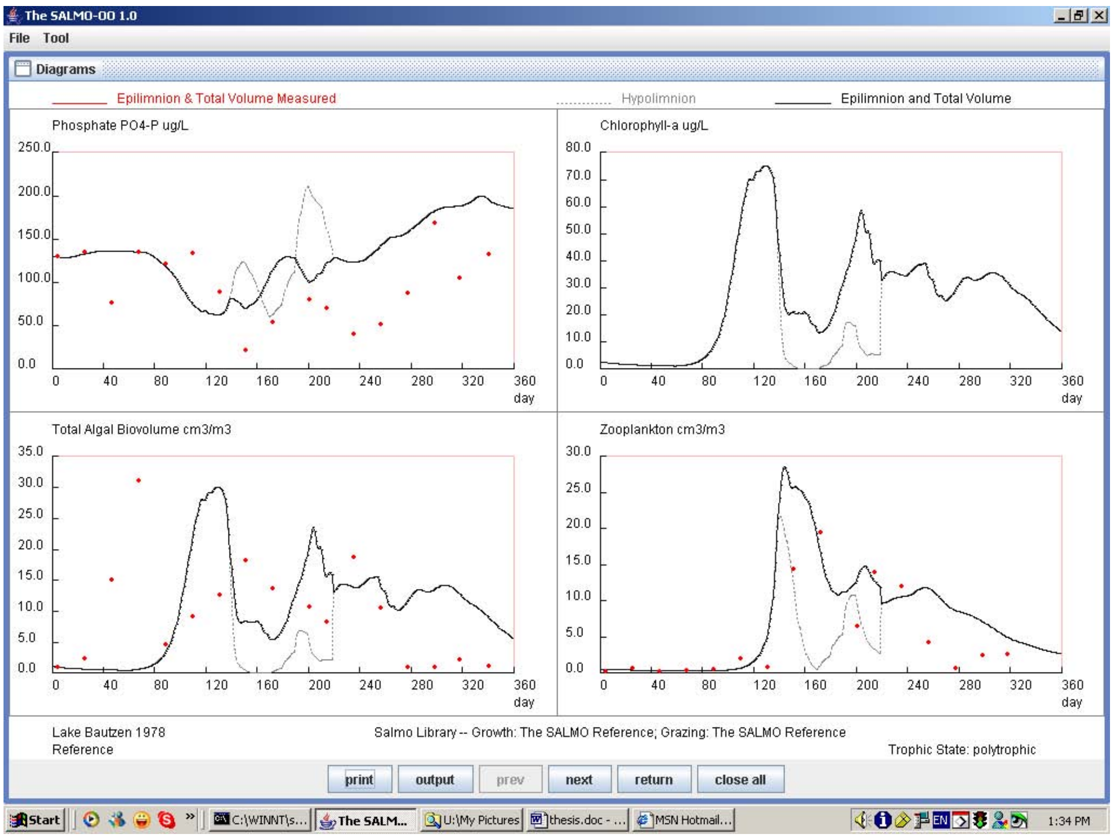

The SALMO-OO 1.0
File Tool
Diagrams
Epilimnion & Total Volume Measured
Hypolimnion
Epilimnion and Total Volume
Phosphate PO4-P ug/L
Chlorophyll-a ug/L
Total Algal Biovolume cm3/m3
Zooplankton cm3/m3
day
Lake Bautzen 1978
Reference
Salmo Library -- Growth: The SALMO Reference; Grazing: The SALMO Reference
Trophic State: polytrophic
print
output
prev
next
return
close all

Figure 5.7 Visualisation of validation results for concentrations of phosphate PO4-P, chlorophyll a, total algal biovolume and zooplankton biovolume simulated for the Bautzen Reservoir in 1978 by the stand-alone version of SALMO-OO (page No. 1)

Figure 5.7 displays the simulated data of DIP, Chlorophyll-a, total algal biomass, and zooplankton. The user can look through other state variables by press the Next button in the bottom of Figure 5.7. For instance, the user wants to display DIP, DIN, oxygen, and detritus, which are illustrated in Figure 5.8.

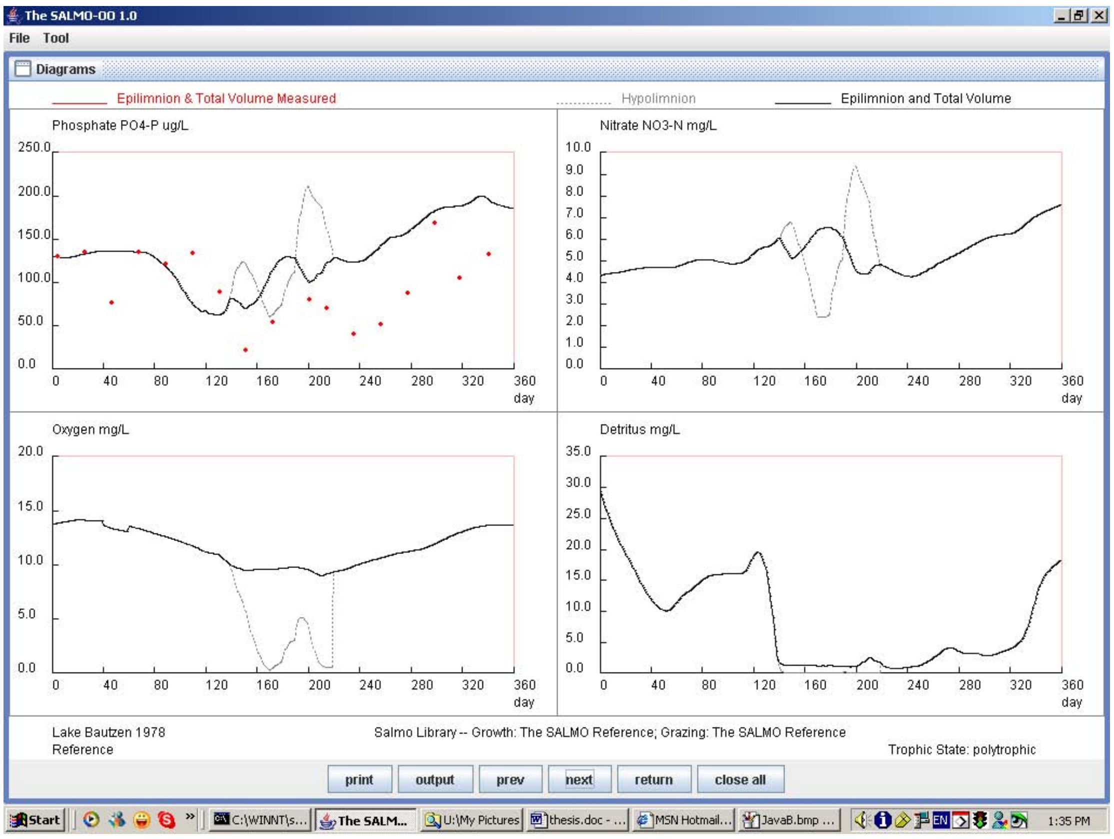


Figure 5.8 Visualisation of validation results for concentrations of phosphate PO4-P, nitrate NO3-N, dissolved oxygen and detritus simulated for the Bautzen Reservoir in 1978 by the stand-alone version of SALMO-OO (page No. 2)

Similarly, Figure 5.9 displays total algal biomass and three functional groups respectively.

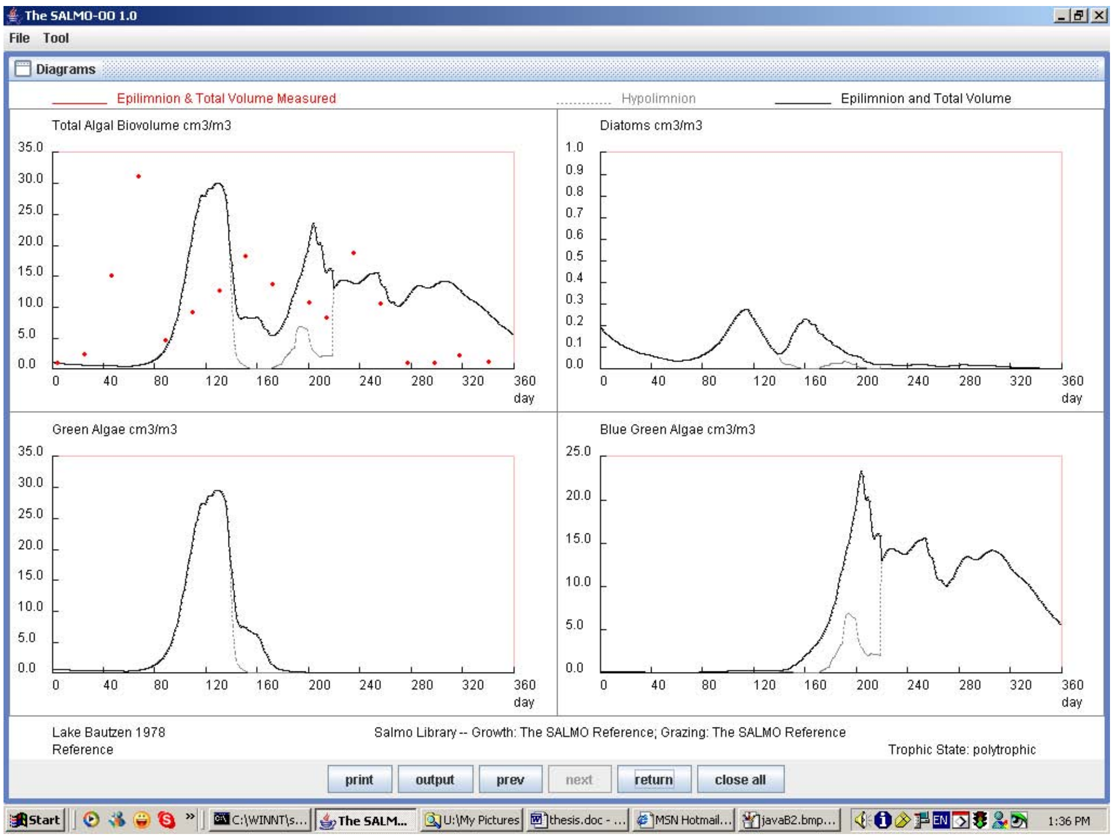
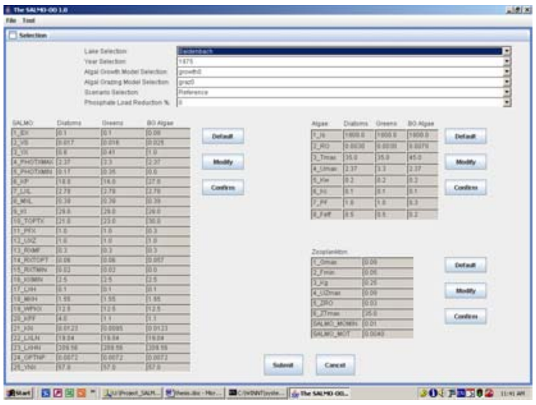

The SALMO-OO 1.0
File Tool
Diagrams
Epilimnion & Total Volume Measured
Hypolimnion
Epilimnion and Total Volume
Total Algal Biovolume cm3/m3
Diatoms cm3/m3
Green Algae cm3/m3
Blue Green Algae cm3/m3
day
Lake Bautzen 1978
Reference
Salmo Library -- Growth: The SALMO Reference; Grazing: The SALMO Reference
Trophic State: polytrophic
print
output
prev
next
return
close all

Figure 5.9 Visualisation of validation results for the concentrations of total algal biomass, diatom biuomass, green algae biomass and blue-green algae biomass simulated for the Bautzen Reservoir in 1978 by the stand-alone version of SALMO-OO (page No. 3)

The user usually wants to look through another lake at the result GUI. For instance, the user is intent to explore Saidenbach Reservoir. It can be done by click the Return button that closed to the Next button. The system will display the initial GUI in Figure 5.4. By performing the similar operation, Figure 5.10 suite displays the simulated data respectively of Saidenbach Reservoir 1975.



Lake Selection	Saidenbach
Year Selection	1975
Algal Growth Model Selection	growth0
Algal Grazing Model Selection	graz0
Scenario Selection	Reference
Phosphate Load Reduction %	0

SALMO	Diatoms	Greens	BG Algae
1_EX	0.1	0.1	0.08
2_VS	0.017	0.016	0.025
3_YX	0.6	0.41	1.0
4_PHOTXMAX	2.37	3.3	2.37
5_PHOTXMIN	0.17	0.35	0.6
6_KP	18.0	16.0	27.0
7_LXL	2.78	2.78	2.78
8_MXL	0.39	0.39	0.39
9_KI	29.0	29.0	29.0
10_TOPTX	21.0	23.0	30.0
11_PRX	1.0	1.0	0.3
12_UXZ	1.0	1.0	1.0
13_RXMF	0.3	0.3	0.3
14_RXTOPT	0.08	0.06	0.057
15_RXTMIN	0.02	0.02	0.0
16_KXMIN	2.5	2.5	2.5
17_LXH	0.1	0.1	0.1
18_MXH	1.55	1.55	1.55
19_WPXX	12.5	12.5	12.5
20_KFF	4.0	1.1	1.1
21_KN	0.0123	0.0085	0.0123
22_LXLN	19.04	19.04	19.04
23_LXHN	209.56	209.56	209.56
24_OPTNP	0.0072	0.0072	0.0072
25_YNX	57.0	57.0	57.0

Algae	Diatoms	Greens	BG Algae
1_Is	1600.0	1600.0	1600.0
2_RO	0.0030	0.0030	0.0070
3_Tmax	35.0	35.0	45.0
4_Umax	2.37	3.3	2.37
5_Kw	0.2	0.2	0.2
6_Ki	0.1	0.1	0.1
7_PF	1.0	1.0	0.3
8_Feff	0.5	0.5	0.2

Zooplankton	
1_Gmax	0.09
2_Fmin	0.05
3_Kg	0.25
4_UZmax	0.09
5_ZRO	0.03
6_ZTmax	35.0
SALMO_MOMIN	0.01
SALMO_MOT	0.0040

Figure 5.10(a) Modelling selection page

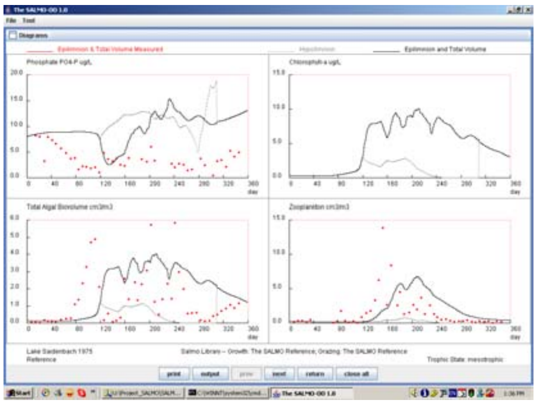
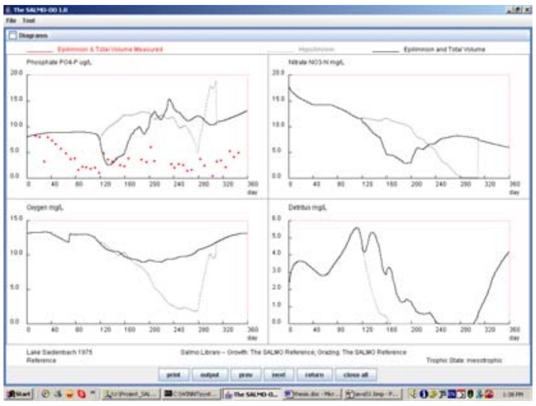

The SALMO-OO 1.0
File Tool
Diagrams
Epilimnion & Total Volume Measured
Hypolimnion
Epilimnion and Total Volume
Phosphate PO4-P ug/L
Chlorophyll-a ug/L
Total Algal Biovolume cm3/m3
Zooplankton cm3/m3
day
Lake Saidenbach 1975
Reference
Salmo Library – Growth: The SALMO Reference, Grazing: The SALMO Reference
Trophic State: mesotrophic
print
output
prev
next
return
close all

Figure 5.10(b) 1st result page



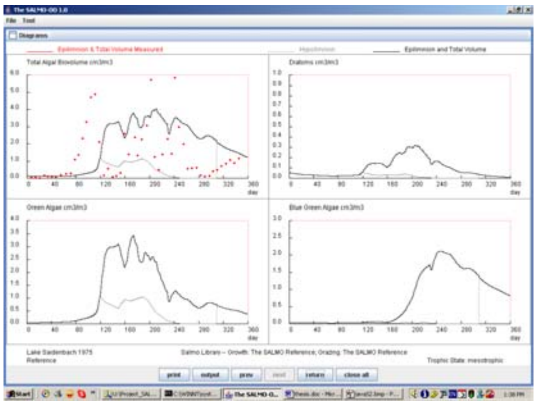
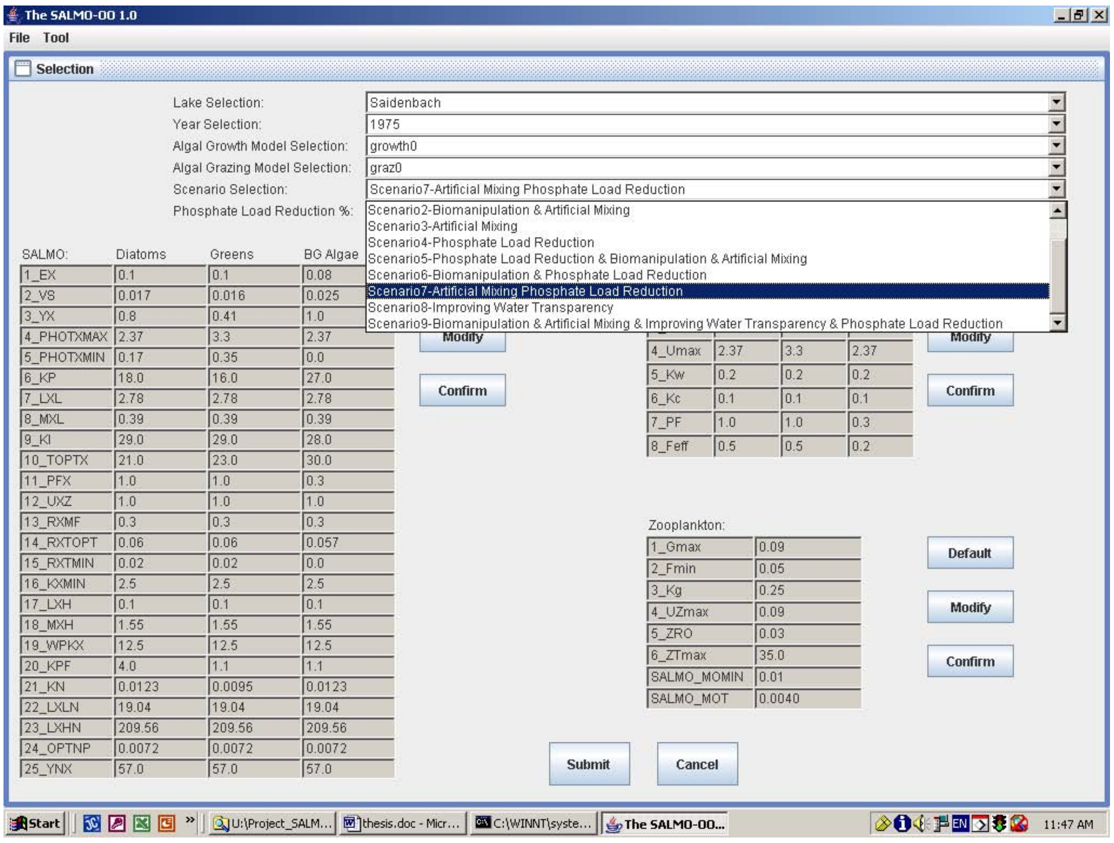

The SALMO-OO 1.0
File Tool
Diagrams
Epilimnion & Total Volume Measured
Hypolimnion
Epilimnion and Total Volume
Total Algal Biovolume cm3/m3
Diatoms cm3/m3
Green Algae cm3/m3
Blue Green Algae cm3/m3
day
Lake Saidenbach 1975
Reference
Salmo Library – Growth: The SALMO Reference, Grazing: The SALMO Reference
Trophic State: mesotrophic
print
output
prev
next
return
close all

Figure 5.10(c) 2nd result page

Figure 5.10(d) 3rd result page

Sometimes, the user adopts various scenario analyses in order to investigate the simulated results of lake management. For example, it is often helpful for freshwater management personnel to reduce DIP loads and artificial mixing to control the total algal biomass. SALMO-OO enables this operation to be easily performed. Figure 5.11 illustrates a user click the Scenario Selection option, which is the Scenario7-Artificial Mixing and Phosphate Load Reduction item.



Lake Selection: Saidenbach
Year Selection: 1975
Algal Growth Model Selection: growth0
Algal Grazing Model Selection: graz0
Scenario Selection: Scenario7-Artificial Mixing Phosphate Load Reduction
Phosphate Load Reduction %:

Scenario2-Biomanipulation & Artificial Mixing
Scenario3-Artificial Mixing
Scenario4-Phosphate Load Reduction
Scenario5-Phosphate Load Reduction & Biomanipulation & Artificial Mixing
Scenario6-Biomanipulation & Phosphate Load Reduction
Scenario7-Artificial Mixing Phosphate Load Reduction
Scenario8-Improving Water Transparency
Scenario9-Biomanipulation & Artificial Mixing & Improving Water Transparency & Phosphate Load Reduction

SALMO:	Diatoms	Greens	BG Algae
1_EX	0.1	0.1	0.08
2_VS	0.017	0.016	0.025
3_YX	0.8	0.41	1.0
4_PHOTXMAX	2.37	3.3	2.37
5_PHOTXMIN	0.17	0.35	0.0
6_KP	18.0	16.0	27.0
7_LXL	2.78	2.78	2.78
8_MXL	0.39	0.39	0.39
9_KI	29.0	29.0	28.0
10_TOPTX	21.0	23.0	30.0
11_PFX	1.0	1.0	0.3
12_UXZ	1.0	1.0	1.0
13_RXMF	0.3	0.3	0.3
14_RXTOPT	0.06	0.06	0.057
15_RXTMIN	0.02	0.02	0.0
16_KXMIN	2.5	2.5	2.5
17_LXH	0.1	0.1	0.1
18_MXH	1.55	1.55	1.55
19_WPKX	12.5	12.5	12.5
20_KPF	4.0	1.1	1.1
21_KN	0.0123	0.0095	0.0123
22_LXLN	19.04	19.04	19.04
23_LXHN	209.56	209.56	209.56
24_OPTNP	0.0072	0.0072	0.0072
25_YNX	57.0	57.0	57.0

4_Umax	2.37	3.3	2.37
5_Kw	0.2	0.2	0.2
6_Kc	0.1	0.1	0.1
7_PF	1.0	1.0	0.3
8_Feff	0.5	0.5	0.2

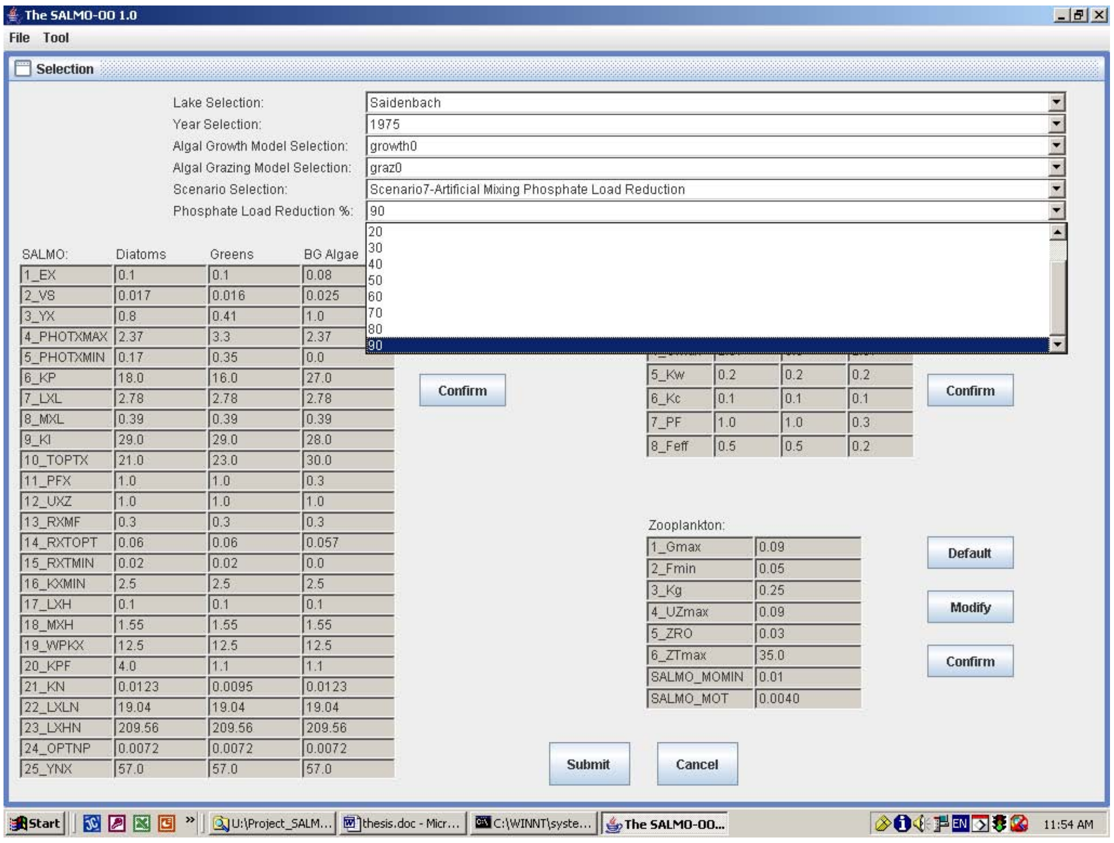
Zooplankton:

1_Gmax	0.09
2_Fmin	0.05
3_Kg	0.25
4_UZmax	0.09
5_ZRO	0.03
6_ZTmax	35.0
SALMO_MOMIN	0.01
SALMO_MOT	0.0040

Start | U:\Project_SALM... | thesis.doc - Micr... | C:\WINNT\syste... | The SALMO-OO... | 11:47 AM

Figure 5.11(a) Selection of the scenario ‘artificial mixing and phosphate load reduction’ for the Saidenbach Reservoir in 1975 by means of the GUI of the stand-alone version of SALMO-OO

After choose the Scenario7, the user needs to choose how much reduction of DIP from the Phosphate Load Reduction % option. In this case, the user clicks the ‘90’ item (Figure 5.11(b)).



Lake Selection: Saidenbach
Year Selection: 1975
Algal Growth Model Selection: growth0
Algal Grazing Model Selection: graz0
Scenario Selection: Scenario7-Artificial Mixing Phosphate Load Reduction
Phosphate Load Reduction %: 90

SALMO:	Diatoms	Greens	BG Algae
1_EX	0.1	0.1	0.08
2_VS	0.017	0.016	0.025
3_YX	0.8	0.41	1.0
4_PHOTXMAX	2.37	3.3	2.37
5_PHOTXMIN	0.17	0.35	0.0
6_KP	18.0	16.0	27.0
7_LXL	2.78	2.78	2.78
8_MXL	0.39	0.39	0.39
9_KI	29.0	29.0	28.0
10_TOPTX	21.0	23.0	30.0
11_PFX	1.0	1.0	0.3
12_UXZ	1.0	1.0	1.0
13_RXMF	0.3	0.3	0.3
14_RXTOPT	0.06	0.06	0.057
15_RXTMIN	0.02	0.02	0.0
16_KXMIN	2.5	2.5	2.5
17_LXH	0.1	0.1	0.1
18_MXH	1.55	1.55	1.55
19_WPKX	12.5	12.5	12.5
20_KPF	4.0	1.1	1.1
21_KN	0.0123	0.0095	0.0123
22_LXLN	19.04	19.04	19.04
23_LXHN	209.56	209.56	209.56
24_OPTNP	0.0072	0.0072	0.0072
25_YNX	57.0	57.0	57.0

5_Kw	0.2	0.2	0.2
6_Kc	0.1	0.1	0.1
7_PF	1.0	1.0	0.3
8_Feff	0.5	0.5	0.2

Zooplankton:

1_Gmax	0.09
2_Fmin	0.05
3_Kg	0.25
4_UZmax	0.09
5_ZRO	0.03
6_ZTmax	35.0
SALMO_MOMIN	0.01
SALMO_MOT	0.0040

Figure 5.11(b) Specification of the phosphate load reduction by 90% for the selected the scenario ‘artificial mixing and phosphate load reduction’ for the Saidenbach Reservoir in 1975 by means of the GUI of the stand-alone version of SALMO-OO

Responsively, the SALMO-OO invokes the artificial mixing dataset associating with DIP load reduction 90%, finally drawing two different curves to represent these simulated results. Figure 5.12 shows the scenario simulation data with dash line differs the original SALMO one with solid line. It is obvious that the total algal biomass is decreased because of DIP reduction.

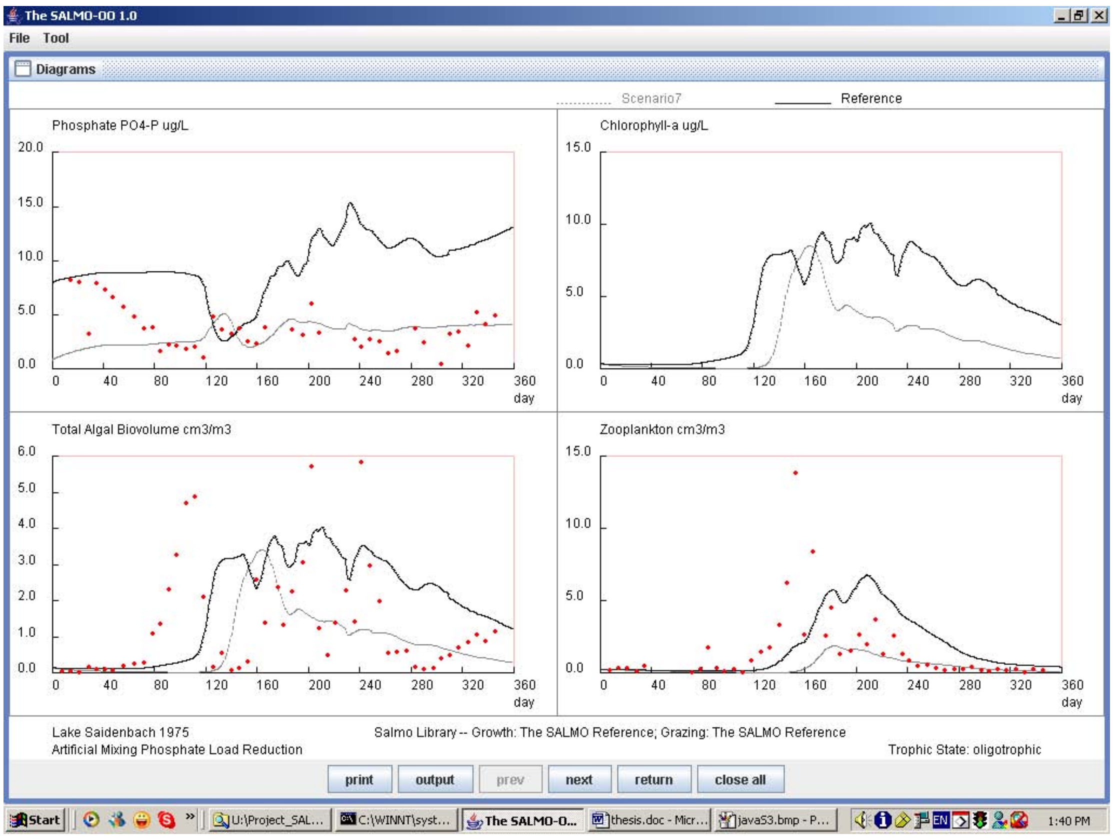
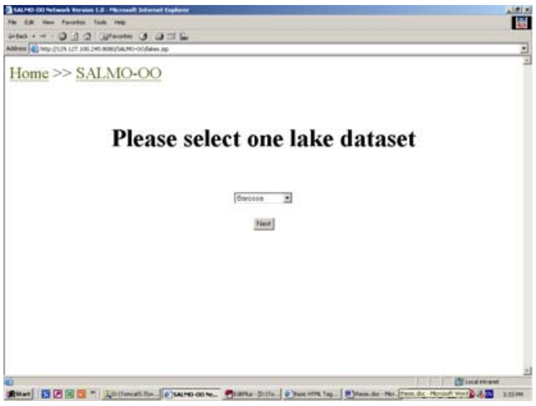

The SALMO-OO 1.0
File Tool
Diagrams
Scenario7
Reference
Phosphate PO4-P ug/L
Chlorophyll-a ug/L
Total Algal Biovolume cm3/m3
Zooplankton cm3/m3
day
Lake Saidenbach 1975
Artificial Mixing Phosphate Load Reduction
Salmo Library -- Growth: The SALMO Reference; Grazing: The SALMO Reference
Trophic State: oligotrophic
print
output
prev
next
return
close all

Figure 5.12 Simulation result of the scenario ‘artificial mixing and phosphate load reduction’ for the Saidenbach Reservoir in 1975 visualised by the GUI of the stand-alone version of SALMO-OO

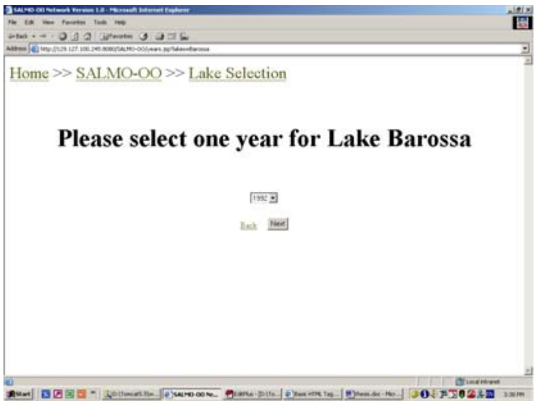
SALMO-OO network edition delivers almost similar visual effect to its stand-alone edition, but disable the functions of the parameter adjustments or the algal growth and grazing model library selections. It is unnecessary for remote users to assess our proceeding project. Although some remote users expect doing so, it is actually hard to keep the robust and correctness of the whole algal model library because varied parameters and algal models have not been verified. In some worst case, some arbitrary parameter and algal model selections could cause system collapse. Thus, SALMO-OO network edition GUI is different from its stand-alone one somewhat.

Figure 5.13(a) shows the initial web page (lake selection). A remote user must choose one lake dataset to skip to the next web page (year selection Figure 5.13(b)). The following web page is scenario selection (Figure 5.13(c)). Finally, the first result web page is displayed in Figure 5.13(d) after the remote user press the Submit button. Also the remote user can look through other results by the Next button (Figure 5.13(e) and Figure 5.13(f)), but web page does not skip.



Home >> SALMO-OO

Please select one lake dataset

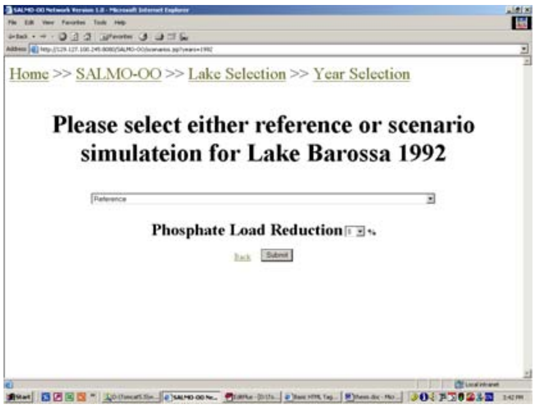


Home >> SALMO-OO >> Lake Selection

Please select one year for Lake Barossa

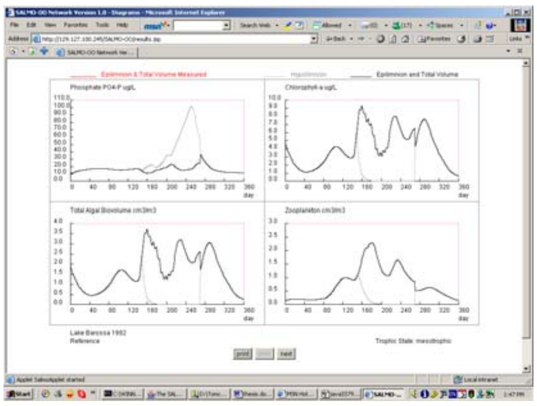
Figure 5.13(a) Lake selection web page

Figure 5.13(b) Year selection web page



Home >> SALMO-OO >> Lake Selection >> Year Selection

Please select either reference or scenario simulateion for Lake Barossa 1992



Epilimnion & Total Volume Measured
Hypolimnion
Epilimnion and Total Volume
Phosphate PO4-P ug/L
Chlorophyll-a ug/L
Total Algal Biovolume cm3/m3
Zooplankton cm3/m3
day
Lake Barossa 1992
Reference
Trophic State: mesotrophic
print
next

Figure 5.13(c) Scenario selection web page

Figure 5.13(d) 1st result web page

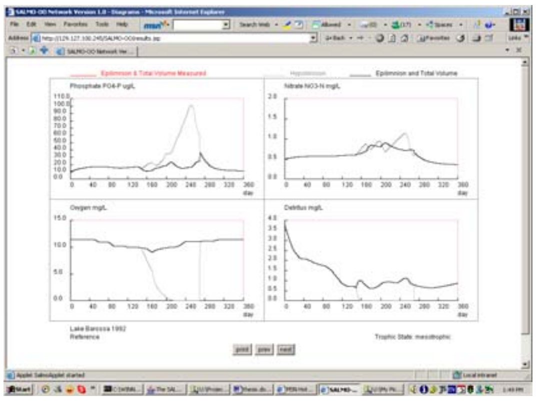


Figure 5.13(e) 2nd result web page

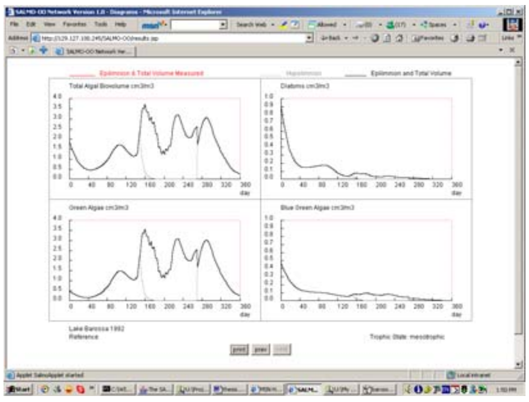

Epilimnion & Total Volume Measured
Hypolimnion
Epilimnion and Total Volume
Total Algal Biovolume cm3/m3
Diatoms cm3/m3
Green Algae cm3/m3
Blue Green Algae cm3/m3
Lake Barossa 1992
Reference
Trophic State: mesotrophic
print
prev
next

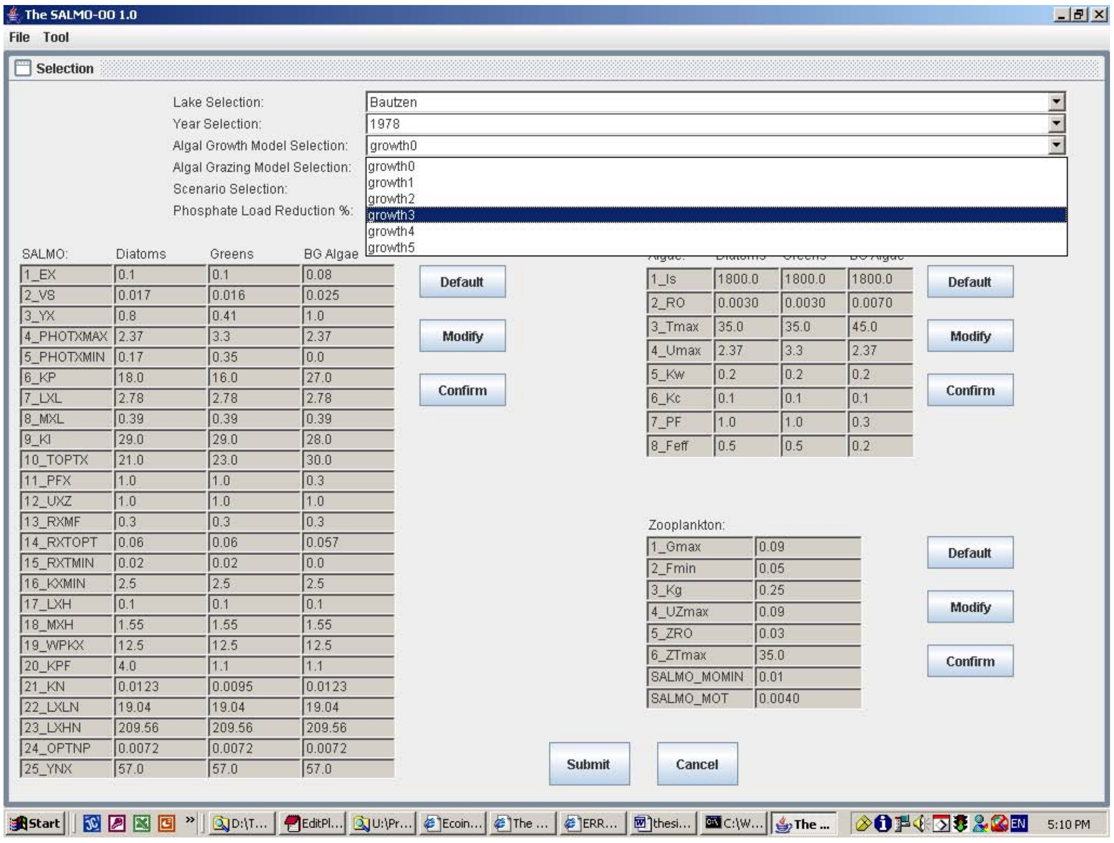
Figure 5.13(f) 3rd result web page

It can be seen from these figures that SALMO-OO provides friendly GUI as well as web-based access. In the stand-alone edition, friendly GUI implementation mainly depends on Java Abstract Window Toolkit (AWT) and Java Swing component, and network edition adds HTML and JSP to be enabled web presentation.

5.3 Object-oriented technology allows for developing an algal model library

SALMO-OO easily implements an algal model library by using object-oriented technology. This algal model library contains a number of algal growth and grazing models, which encapsulated into the AlgaeLibrary class in the salmo.model package. Technologically, the AlgaeLibrary class extends the Phytoplankton class, the various growth and grazing models act as methods in the AlgaeLibrary class. Java programming language provides polymorphism for an AlgaeLibrary object invokes one specific method between the superclass (Phytoplankton) and the subclass (AlgaeLibrary). Therefore, the alternative equations can replace the ones in SALMO-OO; eventually investigate which one improves SALMO as ecological expectations.

As far as users are concerned, these processes are transparent. The model library option is similar to the lake or year option in GUI. Figure 5.14 illustrates how a user chooses a growth model associating with Bautzen Reservoir 1978. Obviously, it does not require users to put any extra efforts.



Lake Selection:	Bautzen
Year Selection:	1978
Algal Growth Model Selection:	growth0
Algal Grazing Model Selection:	
Scenario Selection:	
Phosphate Load Reduction %:	

growth0
growth1
growth2
growth3
growth4
growth5

SALMO:	Diatoms	Greens	BG Algae
1_EX	0.1	0.1	0.08
2_VS	0.017	0.016	0.025
3_YX	0.8	0.41	1.0
4_PHOTXMAX	2.37	3.3	2.37
5_PHOTXMIN	0.17	0.35	0.0
6_KP	18.0	16.0	27.0
7_LXL	2.78	2.78	2.78
8_MXL	0.39	0.39	0.39
9_KI	29.0	29.0	28.0
10_TOPTX	21.0	23.0	30.0
11_PFX	1.0	1.0	0.3
12_UXZ	1.0	1.0	1.0
13_RXMF	0.3	0.3	0.3
14_RXTOPT	0.06	0.06	0.057
15_RXTMIN	0.02	0.02	0.0
16_KXMIN	2.5	2.5	2.5
17_LXH	0.1	0.1	0.1
18_MXH	1.55	1.55	1.55
19_WPKX	12.5	12.5	12.5
20_KPF	4.0	1.1	1.1
21_KN	0.0123	0.0095	0.0123
22_LXLN	19.04	19.04	19.04
23_LXHN	209.56	209.56	209.56
24_OPTNP	0.0072	0.0072	0.0072
25_YNX	57.0	57.0	57.0

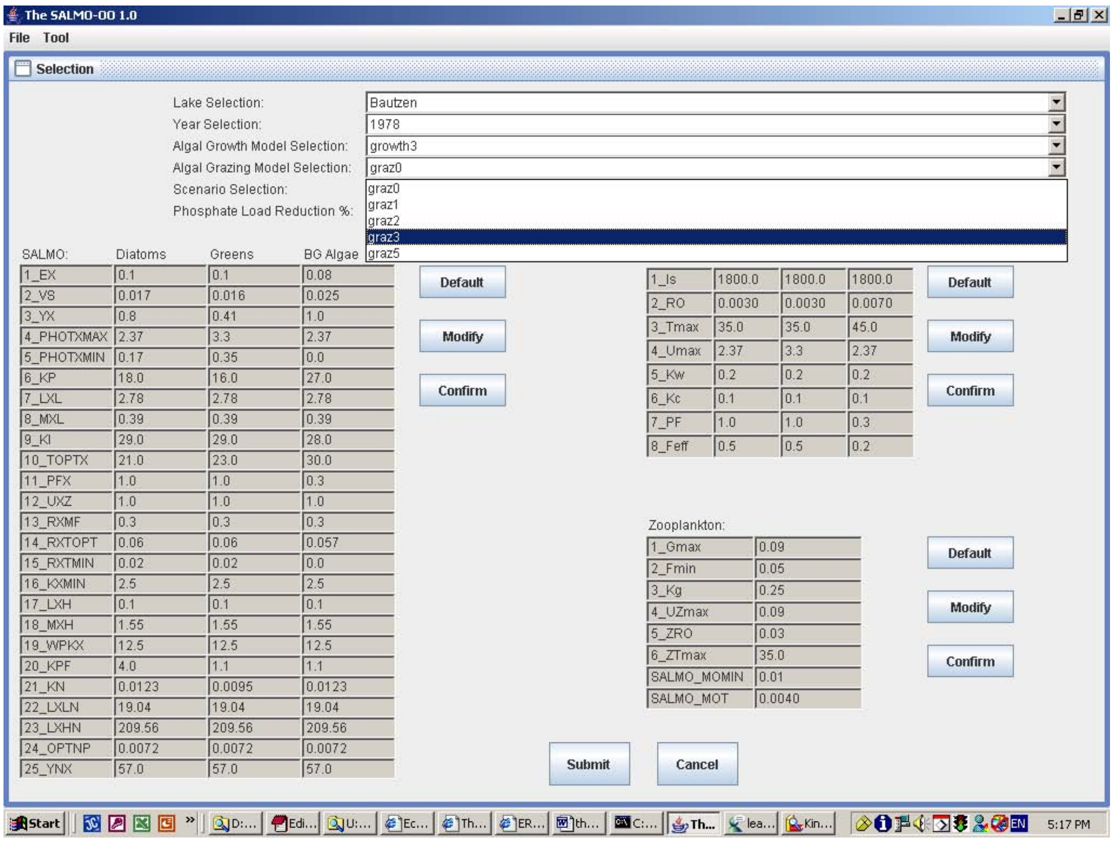
1_Is	1800.0	1800.0	1800.0
2_RO	0.0030	0.0030	0.0070
3_Tmax	35.0	35.0	45.0
4_Umax	2.37	3.3	2.37
5_Kw	0.2	0.2	0.2
6_Kc	0.1	0.1	0.1
7_PF	1.0	1.0	0.3
8_Feff	0.5	0.5	0.2

Zooplankton:

1_Gmax	0.09
2_Fmin	0.05
3_Kg	0.25
4_UZmax	0.09
5_ZRO	0.03
6_ZTmax	35.0
SALMO_MOMIN	0.01
SALMO_MOT	0.0040

Figure 5.14 Selection of the algal growth model 3 from the model library for the simulation of the Bautzen Reservoir in 1978 by means of the GUI of the stand-alone version of SALMO-OO

Usually, a user may observe accompanying grazing model instead of the original SALMO grazing model. Similar operation needs to be done for this user (Figure 5.13(b)).



Lake Selection: Bautzen
Year Selection: 1978
Algal Growth Model Selection: growth3
Algal Grazing Model Selection: graz0
Scenario Selection:
Phosphate Load Reduction %:

graz0
graz1
graz2
graz3
graz5

SALMO:	Diatoms	Greens	BG Algae
1_EX	0.1	0.1	0.08
2_VS	0.017	0.016	0.025
3_YX	0.8	0.41	1.0
4_PHOTXMAX	2.37	3.3	2.37
5_PHOTXMIN	0.17	0.35	0.0
6_KP	18.0	16.0	27.0
7_LXL	2.78	2.78	2.78
8_MXL	0.39	0.39	0.39
9_KI	29.0	29.0	28.0
10_TOPTX	21.0	23.0	30.0
11_PFX	1.0	1.0	0.3
12_UXZ	1.0	1.0	1.0
13_RXMF	0.3	0.3	0.3
14_RXTOPT	0.06	0.06	0.057
15_RXTMIN	0.02	0.02	0.0
16_KXMIN	2.5	2.5	2.5
17_LXH	0.1	0.1	0.1
18_MXH	1.55	1.55	1.55
19_WPKX	12.5	12.5	12.5
20_KPF	4.0	1.1	1.1
21_KN	0.0123	0.0095	0.0123
22_LXLN	19.04	19.04	19.04
23_LXHN	209.56	209.56	209.56
24_OPTNP	0.0072	0.0072	0.0072
25_YNX	57.0	57.0	57.0

1_Is	1800.0	1800.0	1800.0
2_RO	0.0030	0.0030	0.0070
3_Tmax	35.0	35.0	45.0
4_Umax	2.37	3.3	2.37
5_Kw	0.2	0.2	0.2
6_Kc	0.1	0.1	0.1
7_PF	1.0	1.0	0.3
8_Feff	0.5	0.5	0.2

Zooplankton:

1_Gmax	0.09
2_Fmin	0.05
3_Kg	0.25
4_UZmax	0.09
5_ZRO	0.03
6_ZTmax	35.0
SALMO_MOMIN	0.01
SALMO_MOT	0.0040

Figure 5.15 Selection of the algal grazing model 3 from the model library for the simulation of the Bautzen Reservoir in 1978 by means of the GUI of the stand-alone version of SALMO-OO

In the meanwhile, the user can freely adjust various modelling parameters to pursue expected simulation results. By using these functions, my colleague has made further progress on SALMO improvements. Figure 5.16 lists the comparisons among a group of diagrams that represent various simulation results by model library selections and parameter adjustments. Undoubtedly, these improvements benefit from using object-oriented technology.

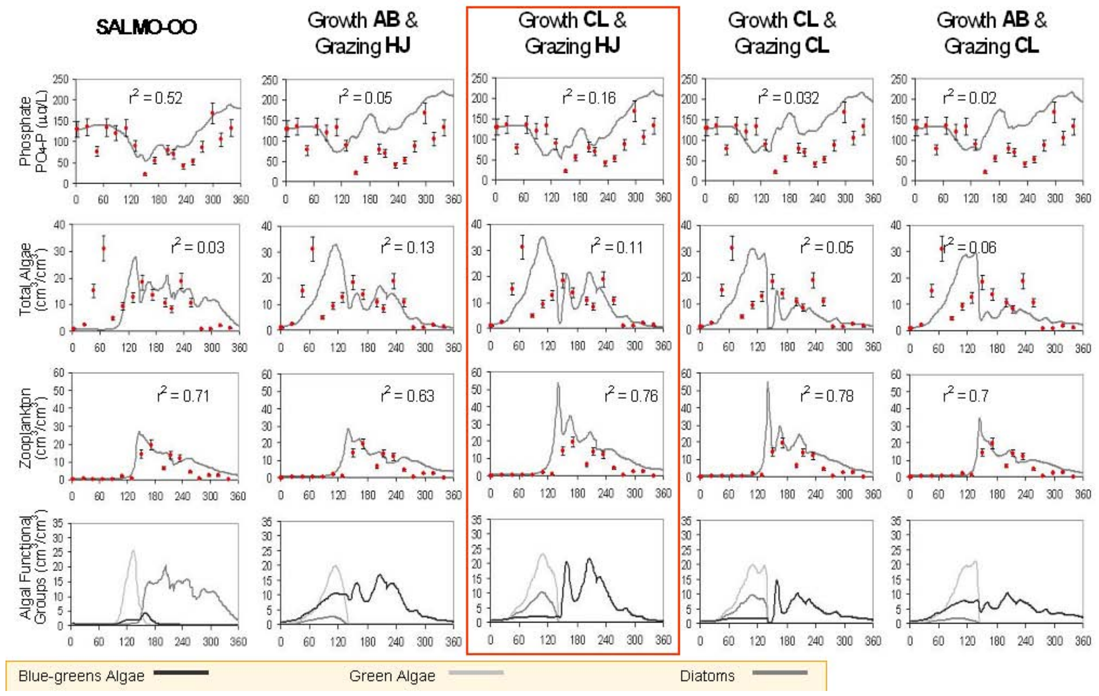


Figure 5.16 Illustration of the simulation results for the Bautzen Reservoir in 1978 by different model structures of SALMO-OO selected from the model library of the stand-alone version of SALMO-OO (from Cetin, Zhang and Recknagel 2005)

5.4 The implication of the SALMO-OO documents

The SALMO-OO documents are to abstractly model SALMO-OO. These documents consist of three basic categories: descriptions, simulations, and instructions. The category of descriptions comprises the SALMO-OO requirement statement and specification document, which clarify the fundaments of SALMO-OO and specify the limitations of SALMO-OO. The second category simulates SALMO-OO by means of UML modelling, including the SALMO-OO use-case diagrams, class diagrams, sequence diagrams, communication diagrams, and UML diagrams for deployment . The third category is the SALMO-OO API specification, which instructs how to use the SALMO-OO class library. The strength of the SALMO-OO documents plays a key role in the process of simplifying the complexity of SALMO-OO. Moreover, these documents provide an understandable platform for anyone who involves SALMO-OO to easily mutual communication. Therefore, the SALMO-OO documents are the primary outcomes of this project.

Chapter 6

Conclusions

Through this project, we draw two conclusions: lake ecosystem modelling can use object-oriented technology and lake ecosystem modelling can benefit from object-oriented software engineering.

We have achieved the key requirements of SALMO-OO. Now, the SALMO-OO creator and developer can arbitrarily vary and extend their logics. The flexibility of the class library permits them to perform any reasonable operations without destruction. Moreover, the friendly GUI enables the model users access SALMO-OO in a comfortable situation. Also they can observe the state variable dynamics with any combinative modelling options.

As far as the methods are concerned, the development of SALMO-OO takes advantages of up-to-date information technologies. With the guidance of object-oriented software engineering, SALMO-OO realizes all the objectives in the proposal. Furthermore, the developing duration and workload has been overestimated because of using some Extreme Programming methods. Finally, SALMO-OO delivers some useful documents to describe the whole developing process. These documents not only clarify SALMO-OO development but also explain SALMO-OO from different perspectives. Therefore, SALMO-OO is far understandable for both ecologists and computer scientists.

It is clear that object orientation has more ability to build the flexible lake ecosystem model class library than some previous means. Lake ecosystem modelling can make achievements by means of object-oriented technology.

Chapter 7

Recommendations

From the scientific perspective, the SALMO-OO class library considers to be improved to be compatible to the future applications. Specifically, a generic freshwater class library contains various variables that involve in freshwater simulation is needed. This expected achievement would highly contribute to the society of water research and management. The benefits are the whole society can reuse the programming code as well as easily deploy this standard class library in various water dependent applications.

The achieved SALMO-OO needs maintenance. The methodology of modern software engineering concludes the maintenance task lives in all the phases of software life cycles. Theoretically, it sounds that we have completed the SALMO-OO maintenance since this project ended. However, it is possible to vary SALMO-OO in the future such as correcting faults, optimising structure, extending algal functional groups or model libraries, changing run environment, even transferring to other computer systems. Therefore, it can be recommended leaving the maintenance of SALMO-OO as the future works.

Maintenance will improve SALMO-OO in the future. However, our conclusions do not prove other object-oriented ecological models are inability to achieve our objectives. Information technology is upgrading everyday, ecological models accompany enforcement once if they adopt novel techniques. It is impossible to trace and look through their improvements in real time. In the world wide, the development of Information technology is far faster than using them in a real application. Thus, SALMO-OO has to keep up with the development in order to make future achievements.

Reference:

Abadi, M., and L. Cardelli. 1998. A Theory of Objects. Springer, New York.

Acock, B., and V. R. Reddy. 1997. Designing an object-oriented structure for crop models. Ecological Modelling **94**:33-44.

Apache Jakarta Project. 1999-2005. *in*. Apache Jakarta Project.

Back, T., D. B. Fogel, and Z. Michalewicz. 2000a. Evolutionary Computation 1 Basic Algorithms and Operators. IOP Publishing Ltd.

Back, T., D. B. Fogel, and Z. Michalewicz. 2000b. Evolutionary Computation 1 Basic Algorithms and Operators. IOP Publishing Ltd.

Beck, K. 1999. Extreme Programming Explained: Embrace Change. Addison-Wesley.

Brenner, N. 2005. VISUAL BASIC .NET – ONE TEACHER'S EXPERIENCE. Consortium for Computing Sciences in Colleges **21**:89 - 94.

Cardelli, L., and P. Wegner. 1985. On understanding types, data abstraction, and polymorphism. ACM Computing Surveys (CSUR) **17**:471 - 523.

Chen, J. L., and J. F. Reynolds. 1997. GePSi: A generic plant simulator based on object-oriented principles. Ecological Modelling **94**:53-66.

Cornell, G., C. S. Horstmann, and C. S. Forstmann. 2002. Core Java 2: Fundamentals. Sun Microsystems.

Dahl, O. J., and K. Nygaard. 1966. SIMULA-an algol-based simulation language. Communications of the ACM **9**:671-678.

Ferreira, J. G. 1994. ECOWIN - an object-oriented ecological model for aquatic ecosystems. Ecological Modelling **79**:21-34.

Flanagan, M. T. 2005. Michael Thomas Flanagan's Java Library RungeKutta Class. *in*. Flanagan, M. T.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. Design Patterns. Addison-Wesley Professional.

Gofen, A. 2001. From Pascal to Delphi to Object Pascal-2000. ACM Press **36**:38-49.

Goldberg, A., and D. Robson. 1983. Smalltalk-80: The Language and Its Implementation. Addison-Wesley, Reading.

Heine, D. L., and M. S. Lam. 2003. A practical flow-sensitive and context-sensitive C and C++ memory leak detector. ACM SIGPLAN Notices, Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation **38**:168-185.

Jacobson, I. 1992. Object-Oriented Software Engineering: A Use Case Driven Approach, 1st edition. ACM Press, Addison-Wesley, New York.

Jenkins, T., and G. Hardman. 2004. How to program using Java. Palgrave Macmillan, New York.

Lemmon, H., and N. Chuk. 1997. Object-oriented design of a cotton crop model. Ecological Modelling **94**:45-51.

McMillan, T., and W. Collins. 1990. Implementing abstract data types in Turbo Pascal. ACM Press **22**:134-138.

Meyer, B. 1988. Object-oriented software engineering. Prentic Hall International (UK) Ltd., Hertfordshire.

Meyer, B. 1992. Eiffel: The Language. Prentice Hall International.

Meyer, B. 1996. The many faces of inheritance: a taxonomy of taxonomy. IEEE Computer **29**:105-108.

Microsoft. 2006a. Learn C#. *in*. Microsoft Corporation.

Microsoft. 2006b. Microsoft Internet Explorer. *in*. Microsoft Windows Group.

MySQL AB. 1995-2005. *in*. MySQL AB.

Object Management Group. 1997-2005. Unified Modeling Language. *in*. Object Management Group.

Papajorgji, P., H. W. Beck, and J. L. Braga. 2004. An architecture for developing service-oriented and component-based environmental models. Ecological Modelling **179**:61-76.

Parker, R. A. 1968. Simulation of An Aquatic Ecosystem. The Biometric Society **24**.

Parnas, D. L. 1971. Information distribution aspects of design methodology. Proceedings of the IFIP Congress:339-344.

Pressman, R. S. 2001. Software Engineering: A Practitioner's Approach, 5th edition. McGraw-Hill Professional, Boston.

Pugh, J. R., W. R. LaLonde, and D. A. Thomas. 1987. Introducing Object-Oriented Programming into the Computer Science Curriculum. ACM Press **19**:98-102.

Rational Software. 2005. Rational Software. *in*. IBM.

Recknagel, F. 1989. Applied Systems Ecology. Akademie Verlag, Berlin.

Recknagel, F., and J. Benndorf. 1982. Validation of the ecological simulation model SALMO. Int. Rev. Ges. Hydrobiol **67 (1)**:113-125.

Recknagel, F., M. Hosomi, T. Fukushima, and D. S. Kong. 1995. Short - and long-term control of exernal and internal phosphorus loads in lakes - a scenario analysis. Water Research **29**:1767-1779.

Ryder, B. G., and M. Burnett. 2005. The impact of software engineering research on modern programming language. ACM Transactions on Software Engineering and Methodology **14**:431-477.

Schach, S. R. 2002a. Object-oriented and classical software engineering, fifth edition. The McGraw-Hill Companies, Inc.

Schach, S. R. 2002b. Object-oriented and classical software engineering, fifth edition. The McGraw-Hill Companies, Inc.

Sekine, M., H. Nakanishi, M. Ukita, and S. Murakami. 1991. A shallow-sea ecological model using an object-oriented programming language. Ecological Modelling **57**:221-236.

Sequeira, R. A., P. J. H. Sharpe, N. D. Stone, K. M. El-Zik, and M. E. Makela. 1991. Object-oriented simulation: plant growth and discrete organ to organ interactions. Ecological Modelling **58**:55-89.

Silvert, W. 1992. Object-oriented ecosystem modelling. Ecological Modelling **68**:91-118.

Soetaert, K., V. deClippele, and P. Herman. 2002. FEMME, a flexible environment for mathematically modelling the environment. Ecological Modelling **151**:177-193.

Stroustrup, B. 1986. An Overview of C++. ACM SIGPLAN Notices **17**.

Stroustrup, B. 1995. Why C++ is not just an object-oriented programming language. ACM Press **6**:1-13.

Sun Microsystems. 1995-2005. What Is an Object. *in*. Sun Microsystems, Inc.

Sun Microsystems, I. 1994-2005a. Java Applet Technology. *in*. Sun Microsystems, Inc.

Sun Microsystems, I. 1994-2005b. Java Code Conventions. *in*. Sun Microsystems, Inc.

Sun Microsystems, I. 1994-2005c. Java Servlet Technology. *in*. Sun Microsystems, Inc.

Sun Microsystems, I. 1994-2005d. JavaServer Pages Technology. *in*. Sun Microsystems, Inc.

Wenderholm, E. 2003. Eclpss: a Java-based framework for parallel ecosystem simulation and modelling. Environmental Modelling & Software.