Abstract

PARANDEKAR, AMEY, VIJAY. Development of a Decision Support Framework for Integrated Watershed Water Quality Management and a Generic Genetic Algorithm Based Optimizer. (Under the direction of Dr. S. Ranji Ranjithan.)

The watershed management approach is a framework for addressing water quality problems at a watershed scale in an integrated manner that considers many conflicting issues including cost, environmental impact and equity in evaluating alternative control strategies. This framework enhances the capabilities of current environmental analysis frameworks by the inclusion of additional systems analytic tools such as optimization algorithms that enable efficient search for cost effective control strategies and uncertainty analysis procedures that estimate the reliability in achieving water quality targets. Traditional optimization procedures impose severe restrictions in using complex nonlinear environmental processes within a systematic search. Hence, genetic algorithms (GAs), a class of general, probabilistic, heuristic, global, search procedures, are used. Current implementation of this framework is coupled with US EPA's BASINS software system.

A component of the current research is also the development of GA object classes and optimization model classes for generic use. A graphical user interface allows users to formulate mathematical programming problems and solve them using GA methodology. This set of GA object and the user interface classes together comprise the Generic Genetic Algorithm Based Optimizer (GeGAOpt), which is demonstrated through applications in solving interactively several unconstrained as well as constrained function optimization problems.

Design of these systems is based on object oriented paradigm and current software engineering practices such as object oriented analysis (OOA) and object oriented design (OOD). The development follows the waterfall model for software development. The Unified Modeling Language (UML) is used for the design. The implementation is carried out using the Java™ programming environment.

Development of a Decision Support Framework for Integrated Watershed Water Quality Management and a Generic Genetic Algorithm Based Optimizer

by

Amey V. Parandekar

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the requirements
for the Degree of Masters of Science

Civil Engineering

Raleigh
August 1999

Approved By:

Dr. S. Ranji Ranjithan
Chair of Advisory Committee

Dr. E. Downey Brill, Jr.

Dr. John W. Baugh, Jr.

Dr. Daniel H. Loughlin

Biography

Amey V. Parandekar is a graduate research assistant in the Department of Civil Engineering at North Carolina State University in Raleigh, North Carolina, since August 1997. Mr. Parandekar received a bachelor's degree in Civil Engineering in April 1997 from Indian Institute of Technology, Bombay, India. He has been pursuing graduate course work and research in the area of Environmental Systems Analysis.

As part of his research, Mr. Parandekar has participated in the Department of Civil Engineering Systems Analysis research group. His research interests include: mathematical programming, conventional as well as heuristic optimization techniques, decision support frameworks to aid environmental decision making, uncertainty analysis, modeling and analysis of environmental systems, environmental economics and policy, software engineering, object oriented analysis and design, software development.

His contributions to the research at NCSU include: A GA based procedure for optimal management of solar access, development of a generic GA based optimizer, design and development of a decision support system for watershed management, and cross-media modeling and assessment.

Mr. Parandekar is looking forward to pursue a career in environmental systems analysis and related areas of computer aided engineering and operations research. His intermediate goal will be doctoral studies in areas related to the above fields.

Acknowledgments

I would like to express my heartfelt gratitude to my advisor and chair, Dr. S. Ranji Ranjithan for his thoughtful advice, support, patience and vision. I am very grateful to him for inspiring me to become a better researcher and allowing me the freedom to work in interesting research areas. I would also like to express my gratitude to committee members, Dr. E. Downey Brill and Dr. John Baugh, Jr and Dr. Daniel Loughlin for their encouragement and support and feedback.

I would like to especially thank S. Kishan Chetan for collaborating and working with me for the development of the DSS for Watershed Management. A special thanks to my colleagues Sujay V. Kumar and Raghu N. Kurlagunda for their help, ideas, suggestions and constructive criticism. I would also like express my gratitude to my friends Ranjit Bharvirkar, M. Venkatraman, Abhijit Hayatnagarkar and Vinay Karle for interesting discussions and making my stay here a very pleasant experience. Thanks to all my office mates for making work in this place enjoyable. I thank my parents and sister for the motivation to excel in life, their enthusiasm, encouragement and support.

Table of Contents

List of Figures vii
List of Tables ix

1 Overview 1

2 A Generic Genetic Algorithm-Based Optimizer 4

2.1 Introduction 4
2.2 Organization 6
2.3 Object Oriented Modeling and Design 6
2.3.1 The Object Model 7
2.3.2 Object Model Representation 8
2.3.3 Visual Modeling – The Unified Modeling Language 8
2.4 GeGAOpt Design and Implementation 9
2.4.1 Data Structure 9
2.4.2 Decision Variable Representations 10
2.4.3 Selection Schemes 11
2.4.4 Crossover Operators 12
2.4.5 Mutation Operators 13
2.5 A Flexible Framework for Optimization Model Formulation 15
2.5.1 Optimization Model Structure 15
2.5.2 A Framework for Solving Constrained Optimization Problems 17
2.5.3 A Flexible Optimization Model Representation 20
2.6 Design Features 21
2.7 GUI features – Interactive problem solving 25
2.7.1 Problem Formulation 25
2.7.2 Controlling the GA Run 26
2.7.3 Monitoring Progress 26
2.8 Test Problems and Results 27
2.8.1 Unconstrained Optimization Problems 28
2.8.2 Constrained Optimization Problems 30
2.9 Conclusions and Recommendations 33

2.9.1 Conclusions *33*
2.9.2 Recommendations for Future Development *33*
References 34
3 Decision Support Framework for Integrated Watershed Water Quality Management 36
3.1 Introduction 36
3.1.1 Current Frameworks for Watershed Management *36*
3.1.2 Limitations of Current Watershed Management Frameworks *37*
3.1.3 Objectives of the Decision Support Framework *37*
3.2 DSS Components and Capabilities 38
3.2.1 BASINS *38*
3.2.2 Watershed Water Quality Modeling with HSPF *39*
3.2.3 Control Strategies for Watershed Management *40*
3.2.4 Analysis of Control Strategies *40*
3.2.5 Optimization Models for Control Strategies *42*
3.2.6 Optimization Procedures *50*
3.2.7 Uncertainty Analyses *51*
3.3 Design of the DSS 53
3.3.1 Integration of the Watershed Model (HSPF) *53*
3.3.2 User Interfaces *54*
3.3.3 Integrating the DSS with BASINS *55*
References 58
4 DSS: Design and Implementation 60
4.1 Introduction to a Software Development Approach 60
4.2 Object Oriented Modeling and Design 61
4.2.1 The Object Model *61*
4.2.2 Object Model Representation *63*
4.2.3 Visual Modeling – The Unified Modeling Language *63*
4.3 A Software Development Approach 64
4.4 Requirements Engineering 66
4.4.1 Capture 'shall' Statements *66*
4.4.2 Allocate Requirements *66*

4.5 Systems Object Oriented Analysis (OOA) – Static View 71
4.5.1 Identification of Software Use Cases 71
4.5.2 Development of Scenarios 71
4.5.3 Establishment of Project Categories 73
4.5.4 Allocation of Use Cases to Categories 73
4.5.5 Development of System Category Diagram 73
4.6 Systems Object Oriented Analysis – Dynamic View 74
4.6.1 Development of Interaction Diagrams 74
4.7 Software Object Oriented Analysis – Static View 76
4.7.1 Initiation of the Category Class Diagram 76
4.7.2 Refinement of Inheritance and Aggregation hierarchies 76
4.7.3 Decomposition and Analysis of Scenarios 76
4.7.4 Addition of View Classes 77
4.8 Software Object Oriented Design – Static View 77
4.8.1 Inter-Class Visibility 77
4.8.2 Exception Handling 78
4.8.3 Abstract Classes 78
4.9 Class Diagrams for the DSS 78
4.10 System Description 93
4.10.1 Water Quality Management – Category WQ_Management 94
4.10.2 Point Source Management – Category PS_Management 95
4.10.3 Non Point Source Management – Category NPS_Management 96
4.10.4 Water Quality Model – Category WQ_Model 97
4.10.5 Optimization Algorithm – Category Optimization_Algorithm 98
4.10.6 Uncertainty Analysis – Category Uncertainty 101
4.10.7 Graphical User Interface Display – Category View 101
4.11 Hotspots and Plug-Points for the DSS Framework 102
References 104

5 Summary 105

Appendix A 108
Appendix B 111

List of Figures

Figure 2.1 A Simple Genetic Algorithm 4
Figure 2.2 Composition of GA *classes* 10
Figure 2.3 Crossover Procedure 12
Figure 2.4 User-Defined Crossover 13
Figure 2.5 User Defined Mutation 14
Figure 2.6 Operators and Representations supported in the current implementation of GeGAOpt.. 15
Figure 2.7 Optimization Model Structure 16
Figure 2.8 Linear and Exponential Penalty Functions 19
Figure 2.9 The Intelligent Problem Converter 20
Figure 2.10 Class Diagram depicting the Genetic Algorithm *classes*.. 23
Figure 2.11 Class Diagram depicting the Optimization Model classes and linkages 24
Figure 2.12 Problem Formulation Panel and Sample Input File.. 25
Figure 2.13 Control Panel and the Results Display 27
Figure 3.1 Genetic Algorithm-based Search Procedure.. 51
Figure 3.2 Users Control Input (UCI) 54
Figure 3.3 DSS Components and Interactions 57
Figure 4.1 Software Development Approach for the DSS.. 65
Figure 4.2 Sample Scenario 72
Figure 4.3 System Category Diagram 74
Figure 4.4 Interaction diagram for UC # 40.. 75
Figure 4.5 Class diagram 1 for the *category* `WQ_Management` 79
Figure 4.6 Class Diagram 2 for the *category* `WQ_Management` 80
Figure 4.7 Class Diagram 1 for the *category* `PS_Management` 81
Figure 4.8 Class Diagram 2 for the *category* `PS_Management` 82
Figure 4.9 Class Diagram 3 for the *category* `PS_Management` 83
Figure 4.10 Class Diagram for the *category* `NPS_Management`.. 84
Figure 4.11 Class Diagram for the *category* `LandUse_Planning`.. 85
Figure 4.12 Class Diagram for the *category* `WQ_Model` 86
Figure 4.13 Class Diagram for the *category* `Optimization_Algorithm` 87

Figure 4.14 Class Diagram for child *category* `GeneticAlgorithm` of `Optimization_Algorithm` 88
Figure 4.15 Class Diagram for the child *category* `LU` of `GeneticAlgorithm` 89
Figure 4.16 Class Diagram for the child *Category* `PS` of `GeneticAlgorithm` 90
Figure 4.17 Class Diagram for the *category* `Uncertainty` 91
Figure 4.18 Class Diagram for the c*ategory* `View` 92
Figure 4.19 Package Structure 93

List of Tables

Table 2.1 Results for Unconstrained Optimization 29

Table 2.2 Results for Constrained Optimization 31

Table 4.1 Requirements Trace Matrix (RTM) 67

Table 4.2 Probability Distributions 101

1 Overview

Environmental management problems associated with water quality in watersheds are inherently complex and are difficult to analyze due to many interactions among governing physical, chemical and biological processes and the impacts they undergo as a result of anthropogenic activities. Further, finding good management alternatives becomes exceedingly difficult due to conflicting issues such as cost, environmental impact and equity that need simultaneous consideration. Solutions to these problems require an integrated approach to the modeling, analysis and management of the watershed system.

The Better Assessment Science Integrating Point and Nonpoint Sources (BASINS) software system, (USEPA 1998) integrates a geographic information system, national watershed data, and state-of-the-art environmental assessment and modeling tools under a single platform. Within the current scope of BASINS, a user can employ existing computational utilities to develop alternative total maximum daily load (TMDL) strategies to achieve desired water quality targets. Although efficient, the BASINS system supports, at best, a trial-and -error approach to determining feasible TMDLs. This is still limiting given the large number of variables that a user can change and the needs for considering targets on numerous water quality parameters and multiple management criteria. The number of feasible combinations is very large, which makes a trail-and-error search process inefficient. A systematic search process is therefore recommended.

The primary focus of the ongoing research is to develop a computer-based framework that integrates formal optimization procedures with BASINS to enhance the existing capabilities for decision support in watershed water quality management. This decision support system (DSS) enhances the environmental analysis and management capabilities of BASINS: by automating the search for feasible as well as optimal environmental management strategies; by estimating the uncertainty in the achievement of environmental targets by these strategies; and by facilitating comparisons of these alternatives with respect to quantified and unmodeled issues. A formal optimization procedure that is integrated into this DSS enables the user to identify good strategies that meet cost, environmental and reliability goals. Using the highly interactive capabilities of the DSS, the user may further modify these alternative strategies by trail-and-error to explore and examine more alternatives. Using these capabilities collectively, the user may engage in iterative search to efficiently identify "good" alternatives. The ambient watershed water quality is simulated by using water quality models supported within BASINS. The current implementation uses the Hydrologic Simulation Program – FORTRAN (HSPF) that is embedded within BASINS.

The DSS enables users to formulate various point source control strategies such as command-and-control (CAC), effluent charges and transferable discharge permit (TDP) programs, and non-point source control strategies such as land use planning, riparian buffer zoning, and system-wide detention ponds. An explicit quantitative analysis of the uncertainty to estimate the reliability of achieving the water quality targets is also supported within the DSS. The DSS also facilitates the propagation of the uncertainty in model parameters by using various stochastic sampling procedures such as Monte Carlo and Latin Hypercube sampling.

The mathematical models including the HSPF-based water quality modeling that represent the governing processes associated with watershed water quality are typically non-continuous and nonlinear. Therefore, the management optimization models that integrate these non-linear models are inherently nonlinear. As solving these management models using conventional optimization procedures such as linear programming (LP) and non-linear programming (NLP) solvers are not viable, a heuristic search procedure based on genetic algorithms (GAs) is employed.

The primary contribution of the work reported in this thesis is the design of a decision support framework that supports these models, analysis and search procedures, and interactive capabilities. To maintain the modularity of the various components and achieve better system design and promote code reusability, the software design of this framework is based on object oriented (OO) paradigm and design principles. Development of the DSS is based on current software engineering practices such as object oriented analysis (OOA) and object oriented design (OOD). This approach integrates front-end activities of requirements engineering based on use cases with the back end design and implementation. The development follows the waterfall model for software development. The Model-View-Controller (MVC) architecture is used to couple the domain classes of the DSS with the user interface components. The Unified Modeling Language (UML) is used for the design. The implementation is carried out using the Java^{TM} platform (JDK1.2 and JFC 1.1).

A major component of this thesis research is the development of GA object classes and optimization model classes for use in solving optimization problems. A graphical user interface allows users to formulate non-linear, unconstrained as well as constrained optimization problems and solve them using a GA procedure. This set of GA object and user interface classes together comprise the Generic Genetic Algorithm-Based Optimizer (GeGAOpt). This tool improves upon existing general purpose GA tools and software by:

- allowing multiple decision variable representations;
- building optimization model structure that allows users to define the objective function and the constraints as functions of the decision variables;
- allowing for the use of various penalty function approaches for solving constrained optimization problems; and
- enabling users to access and run the GeGAOpt over the internet

The development of the GeGAOpt is also based on object oriented (OO) paradigm and design principles. A subset of the Unified Modeling Language (UML) is used for the OO design for the GeGAOpt. The implementation is carried out using the Java^{TM} platform. The use and performance of the GeGAOpt in solving interactively several standard unconstrained as well as constrained test problems are also demonstrated.

2 A Generic Genetic Algorithm-Based Optimizer

2.1 Introduction

Genetic algorithms (GAs) are adaptive, global search procedures that are designed to mimic the underlying processes governing natural selection and evolution. A GA works on a population of *individuals* to identify relatively better *individuals* among the population, and combines the information embedded in the *individuals* to form newer *individuals* that are generally better. Each *individual* represents a solution to an optimization problem and the fitness of the solution is evaluated based on the performance of that solution in solving the given optimization problem. Through repeated application of a set of basic GA operators, including selection, mutation, and crossover, on the population of the *individuals*, the population generally converges to a solution that performs well with respect to the evaluation criterion. Although GAs do not guarantee global optima, in practice a population based search is robust as compared to local search approaches, and has been demonstrated to be quite successful in finding good solutions in an efficient manner. The following figure briefly illustrates a simple steady state GA procedure.

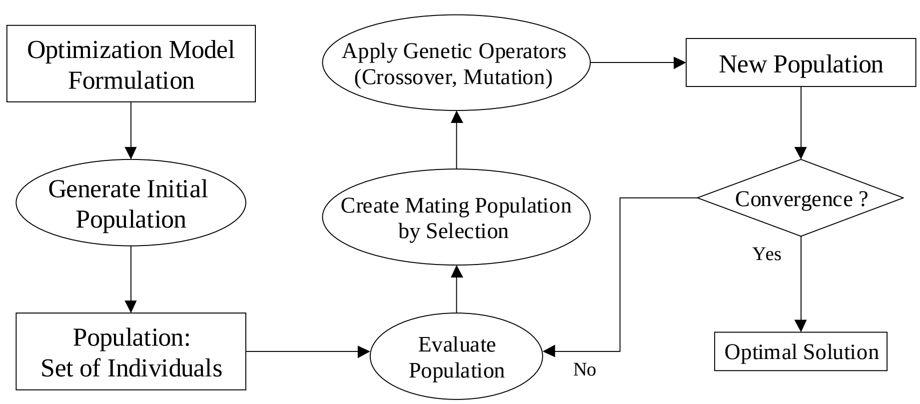


Figure 2.1 A Simple Genetic Algorithm

Over the last decade, there has been considerable research addressing the development of GA methodology and its use in finding benchmark solutions to a gamut of engineering applications. However, very few implementations enable users to formulate generic mathematical optimization problems and solve them using a GA approach.

Various class libraries that include tools for using GAs to solve optimization problems have been designed. Wall (1995) developed the “GALib – A C++ Library of Genetic Algorithm Components” that enables users to perform optimization in C++ programs using several representations and genetic operators. The “GA Optimization Toolbox for Matlab” developed by Houck et al. (1995) implements simulated evolution in the Matlab environment using both binary and real representations and offers flexibility in choosing genetic operators, selection schemes and termination criteria. A user of these tools, however, should be knowledgeable of the various GA representations and operators, and sufficiently knowledgeable about the specific programming environment. To model problems that require multiple decision variable representations, the core modules of these tools need to be extended and built upon. For example, a mixed integer non-linear programming problem may require real and binary decision variables and the real decision variables may need to be represented by either real or binary numbers. For these mixed representations, the crossover and mutation operators need to be customized. The solution of complex constrained optimization problems using GAs requires these tools be supplemented with a complex fitness evaluation module that considers the relative significance of the constraint violations and their contribution to the fitness function. This requires the evaluation function to be hard coded for each problem.

The approach discussed in this paper results in the development of the tool Generic Genetic Algorithm Based Optimizer (GeGAOpt) that improves upon earlier GA tools by:

- supporting multiple representations within a single problem;
- obviating the need to re-code conventional genetic operators for complex problems that require multiple representations;
- constructing an optimization model structure that allows users to define the objective function and the constraints as functions of the decision variables and use various penalty function approaches to specify the relative significance of the constraint violations in the fitness function;

- enabling all of the above tasks with minimal user knowledge of the programming environment or of the coding of the GA operators; and
- enabling users to access remotely and run the GeGAOpt over the internet.

The objectives of this research are to:

- develop GA object classes and classes that represent a generic optimization model for generic use;
- implement a graphical user interface that enables users to formulate generic mathematical function optimization problems and solve them by selecting from a combination of various decision variable representations, GA operators, and various approaches for handling constraint violations;
- demonstrate the use of the above interface for solving standard unconstrained and constrained optimization problems; and,
- assist users in understanding and exploring the working of GAs and in learning about GAs.

2.2 Organization

Section 2.3 discusses the software design and development approach adopted for the development of the GeGAOpt. Section 2.4 describes the development of the core GA modules, i.e., the data structures, representations, GA operators, etc. Section 2.5 describes an optimization model structure that allows users to define generic mathematical optimization problems and the use of various penalty function approaches for solving constrained optimization problems. The software design of the GeGAOpt is discussed in Section 2.6. Section 2.7 describes the features of the user interface of GeGAOpt. Section 2.8 discusses the solution of examples of non-linear and constrained function optimization problems that are solved using the various representations, operators, and penalty function formulations supported within the GeGAOpt.

2.3 Object Oriented Modeling and Design

Developing a flexible, generic, and modular system that is capable of representing the GA data structures and operators calls for better modeling and design approaches to build the software system. Object oriented (OO) modeling and design promotes better understanding of requirements, cleaner designs and maintainable systems. OO technology is a way of thinking abstractedly about a problem using real world concepts, rather than computer concepts. The primary themes underlying OO technology are discussed briefly below:

- **Abstraction** consists of focusing on the inherent aspects of an object and functionality. This avoids design and implementation decisions being made before the problem is understood.
- **Encapsulation** consists of separating the external aspects of an object that are accessible to other objects from the internal aspects of the object. This prevents small changes in object structure from having massive ripple effects that affect the entire class hierarchy.
- **Inheritance** of the data structure and behavior allows common structure to be shared among various *classes*. Inheritance is implemented using *abstract classes* and *interfaces*. These are used as templates for creating *subclasses*.

The development of the GeGAOpt is based on these OO paradigm and design principles.

2.3.1 The Object Model

The object model captures the static structure of a system by showing its objects, the identity of and interrelationships among the objects. An *object* is defined as a concept or an abstraction. It captures concepts from the real world that are important to the application. A *class* is a static abstraction of a set of real world entities that have the same characteristics and share the same behavior. The principal features of a *class* are its *attributes*, *operations* and its relationship to other *classes*. An *attribute* is a data value held by *objects* in a *class*. An *operation* is any function that may be performed by objects. *Classes* are arranged into hierarchies sharing common structure and behavior.

An *association* is a semantic relationship that exists between two *classes*. There are three kinds of associations: aggregation relationship, inheritance relationship, and association relationship. The relationships among the *classes* are discussed briefly below.

Aggregation Relationship

This is an *association* based on the ‘whole/part’ concept. There are two types of aggregation relationships:

- *Aggregation* is a relationship where the ‘whole’ *class* does not have to create its ‘part’ *class*, but refers to the ‘part’ *classes* by reference,
- *Composition* is a relationship where the ‘whole’ part is responsible for creating its ‘part’ *classes* directly. This indicates a tighter coupling between the ‘whole’ and ‘part’ *classes* than is indicated by *aggregation* with reference.

Inheritance Relationship

This is an *association* between *classes* that focuses on similarities and dissimilarities among the *classes* with respect to their *attributes* and *methods*. An *inheritance relationship* exists between a *superclass* and a *subclass*. A *subclass* is a *class* derived from its *superclass* and inherits all the *attributes* and *methods* of its parent *superclass*.

Association Relationship

An *association relationship* defines the nature of the coupling between two *classes*. In this relationship, the interacting parts are visible to each other and may be shared between different aggregation hierarchies.

2.3.2 Object Model Representation

The *object model* is represented graphically with diagrams containing *classes*. *Classes* are arranged into hierarchies sharing common structure and behavior. A *package* is a collection of *classes* that are logically related. A *class diagram* describes the types of objects in the system and the various kinds of static relationships that exist among them. *Class diagrams* also show the *attributes* and *operations* of a *class*.

2.3.3 Visual Modeling – The Unified Modeling Language

The Unified Modeling Language (UML) is a standard graphical language for specifying, constructing, visualizing, and documenting software-intensive systems. It covers concepts such as system processes and functions, as well as specific items such as programming language *classes*, database schemas, and reusable software components. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML provides:

- a ready-to-use, expressive visual modeling language that can be used to develop and exchange meaningful models;
- extensibility and specialization mechanisms to extend the core concepts;
- a system representation that is independent of particular programming languages and development processes; and
- a formal basis for understanding the modeling language.

2.4 GeGAOpt Design and Implementation

This section discusses the conceptualization, the design, development and implementation of a set of object classes to represent the data structures and operators used in GeGAOpt.

2.4.1 Data Structure

Designing appropriate data structures to represent the hierarchy of genetic information content is of central importance to the GA search procedure. It is also essential to keep the design modular and sufficiently abstract so that it facilitates flexibility for further development.

Typically, the hierarchy can be represented adequately by a *gene* at the lowest level. A *chromosome* is a vector of similar types of genes. An *individual* is a container for a set of *chromosomes*. The information necessary for evaluation of the fitness properties is encapsulated within the *individual*. A *population* comprises of a set of *individuals* and is the basic evolving unit. *Individuals* in a *population* undergo crossover and mutation to produce the next generation of *individuals*. Figure 2.2 depicts the composition of the various GA *classes*. The ovals represent the *classes* and the nested structure reflects the *composition relationship* amongst the *classes*.

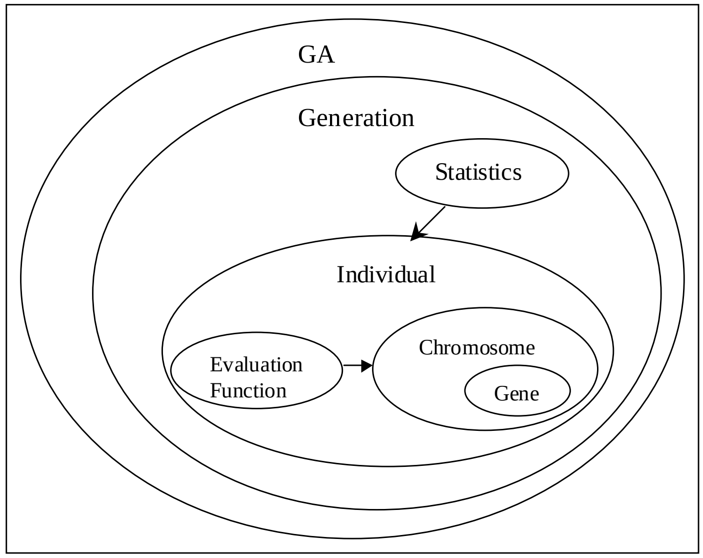


Figure 2.2 Composition of GA *classes*

The links represent the *association relationships* between the *classes*. A unidirectional link, i.e. an arrow, indicates that the *association relationship* is one sided. In this case, Chromosome 'is visible' to Evaluation Function.

2.4.2 Decision Variable Representations

The representation of a potential solution by an individual is the manner in which the genetic material is coded. Defining an appropriate representation is part of the art of using GAs. The representation should be minimal but completely expressive. The user should select a representation so that short, low-order schemata are relevant to the underlying problem and relatively unrelated to schemata over other fixed positions (Goldberg 1989).

The current GeGAOpt implementation supports real (***r***) and binary (***b***) representations for decision variables. A set of real decision variables can be represented either as a real string, as a set of binary strings, or a combination of both. Integer programming problems can be formulated by specifying the real decision variable to be of type ***r(0)*** or of type ***b(0)*** where the number in the parentheses indicates the number of decimal digits desired.

These representations are adequate for solving function optimization problems. Different problems, however, need different representations and these can be easily coded by the user by writing a *class* that implements the *interface* Operators. The primary operations implemented are `crossWith()`, `mutate()` and `getDecisionVariableValue()`. The user must select the crossover and mutation operators so that the integrity of the *chromosome* is preserved. For example, in a problem that depends on a sequence of items that lends itself to an order-based representation, crossover must generate reordered lists without duplicating any element in the list.

2.4.3 Selection Schemes

The selection operator ensures that the GA yields incrementally better *individuals* in each generation. The *individuals* with larger fitness values get higher probability of being selected for mating than *individuals* having low fitness values. The selected *individuals* are then placed in a mating pool. *Individuals* from this mating pool then undergo crossover. The following selection schemes are supported:

Roulette Wheel

In this type of selection, the probability of an *individual* being selected and placed in the mating subset is proportional to its relative fitness.

$$P_{select} = \frac{f_i}{\sum_{i=1}^{N} f_i} \tag{2.1}$$

$$\begin{aligned} &where \\ &f_i = fitness\ of\ individual\ i \\ &N = population\ size \end{aligned}$$

This ensures that *individuals* with higher fitness have a better representation in the mating subset.

Binary Tournament Selection with Replacement

Two *individuals* are drawn at random from the population and the *individual* with the higher fitness score is placed in the mating subset. Both *individuals* are returned to the population and this procedure continues until the mating subset is full. A characteristic of this selection scheme is that the worst *individual* in the population is never selected for inclusion in the mating subset whereas the best *individual* may carry multiple copies into the mating subset.

2.4.4 Crossover Operators

The crossover operator defines the procedure for generating child *individual*(s) by mating two parent *individuals*. A crossover probability determines if the parent *individuals* will cross to produce child *individuals* and the level of mixing of genetic information. Convergence is rapid with higher crossover probabilities and may lead to premature convergence. The ideal crossover probability is problem dependent. *Individuals* may consist of multiple *chromosomes,* in which case the corresponding *chromosomes* of the two parent *individuals* P_1 and P_2 undergo crossover. Figure 2.3 illustrates the crossover procedure.

Procedure 'Crossover'

1. Select two parent individuals, P_1 and P_2, for crossover.

2. if(Crossover) then

Do (P_1 [Chromosome$_i$] X P_2 [Chromosome$_i$]) $\forall$ $i=1,N_c$

where N_c is the number of chromosomes in an individual

3. else

Place P_1 and P_2 in the next generation.

Figure 2.3 Crossover Procedure

In the current implementation, the following crossover operators are supported.

<u>Single Point Crossover</u>

For *chromosomes* $c_g^t = \langle g_1, ..., g_n \rangle$ and $c_h^t = \langle h_1, ..., h_n \rangle$, a position $k \in (1, n)$ is determined randomly. The resulting child *chromosomes* are $c_g^{t+1} = \langle g_1, ..., g_k, h_{k+1}, ..., h_n \rangle$ and $c_h^{t+1} = \langle h_1, ..., h_k, g_{k+1}, ..., g_n \rangle$.

<u>Multi Point Crossover</u>

For *chromosomes* $c_g^t = \langle g_1, ..., g_n \rangle$ and $c_h^t = \langle h_1, ..., h_n \rangle$, a set of positions $K = \{k \mid k \in (1, n)\}$ is determined randomly. The resulting child *chromosomes* are combinations of the genes of the parent *chromosomes* generated in a manner similar to the single point crossover.

Uniform Crossover

For *chromosomes* $c_g^t = \langle g_1,...,g_n \rangle$ and $c_h^t = \langle h_1,...,h_n \rangle$, a binary template $T = \langle t_1,...,t_n \rangle \,|\, t_i = 0{,}1$ is generated randomly. The child *chromosomes* c_g^{t+1} and c_h^{t+1} are constructed as follows:

if(t_i = 0) then $g_i \in c_g^{t+1}$ *and* $h_i \in c_h^{t+1}$ (no gene crossover)

else $h_i \in c_g^{t+1}$ and $g_i \in c_h^{t+1}$ (the parent genes g_i and h_i are interchanged)

Uniform Crossover with Recombination

This operator is similar to the uniform crossover except that the genes g_i and h_i are combined to generate the child genes $g_i^{t+1} = fg_i + (1-f)h_i$ and $h_i^{t+1} = (1-f)g_i + fh_i$. The fraction f is chosen randomly. A recombination rate controls the probability of recombination.

User defined crossover

The user may extend the abstract *class Chromosome* to implement specific crossover operators. Figure 2.4 illustrates an example of this.

```
class MyChromosome extends Chromosome {
      …
      public Object[] conventionalCross {
            …user fills in code…
      }
}
```

Figure 2.4 User-Defined Crossover

2.4.5 Mutation Operators

The mutation operator defines the procedure for mutating each *chromosome*. Mutation introduces new genetic material into the gene pool of the population of *individuals*. This may result in the exploration of previously unexplored points in the decision space or reintroduce lost genetic material. The mutation operator contributes to the global search by enabling the procedure to explore potentially new parts of the decision space as well as to maintain population diversity. The mutation is controlled by a mutation probability that determines if a particular *chromosome* undergoes mutation. Larger mutation probability increases the probability of disruption of good schemata, but increases population diversity. The ideal mutation rate depends on the characteristics of the decision and objective spaces. Hessner and Manner (1991) suggest that a good estimate of the mutation probability is $\frac{1}{(M\sqrt{L})}$, where M is the population size and L is the length of the *chromosome*.

Mutation is different for different data types. For example, a typical mutation for a binary gene is flipping the bit with a given probability. In the current GeGAOpt, the following mutation operators are supported.

Random change mutation

This operator changes the value of a randomly chosen set of genes. The number of genes in this set can be specified by the user. For example, for a real number gene, this operator will randomly set the value of this real number to be a number between the minimum and maximum bounds of that real number gene. This mutation operator is suitable for *chromosomes* of real number genes and genes having more complex data structures.

Swap mutation

This consists of randomly selecting two distinct locations within the *chromosome* and swapping the genes between the selected locations. This mutation is suitable for *chromosomes* of binary genes and *chromosomes* where the permutation of genes needs to be preserved, but is not applicable to complicated chromosomal data structures where the types of genes at different locations are different.

User Specific Mutation

The user can specify mutation operators in either of the ways shown in Figure 2.5.

```
class MyGene implements Operators {
      …
      public void userSpecificMutate {
            …user fills in code…
      }
}
```

```
class MyChromosome extends Chromosome {
      …
      public void userSpecificMutate {
            …user fills in code…
      }
}
```

Figure 2.5 User Defined Mutation

Elitism

Elitism is an optional GA operation that ensures that the best *individual* in a population is not lost during the iterative search process. The current implementation supports generational elitism in which the worst *individual* in a *generation* is replaced with the best *individual* from the previous generation.

Figure 2.6 shows the different types of operators discussed above.

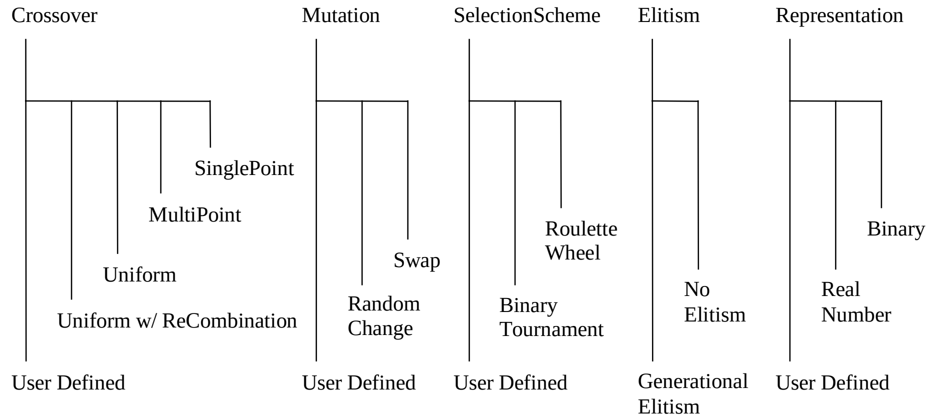


Figure 2.6 Operators and Representations supported in the current implementation of GeGAOpt

2.5 A Flexible Framework for Optimization Model Formulation

2.5.1 Optimization Model Structure

A typical mathematical programming problem consists of the definitions of the decision variables, the objective function, and a set of constraints. The constraints and the objective function are expressed as mathematical expressions written in terms of the decision variables.

In the framework presented here, the mathematical programming problem is represented by a dynamic set of data structures. A set of *classes* is developed that emulates the conceptual framework of a mathematical programming problem. Using this set of *classes*, an instance of a user's definition of the optimization problem can be constructed. This instance can then be solved to yield a solution to the optimization problem.

The system of *classes* so developed provides the user the powerful capability of specifying the optimization problem in an algebraic format that is similar to that used by conventional mathematical programming solvers such as LINDO, MATLAB, etc. A user's definition of an optimization problem is then translated by a *problem translator* to an internal system representation of the optimization problem.

Figure 2.7 shows the composition among the *classes* that collectively represent the optimization model. In the following discussion, the Courier New font is used for the name of a *class*. The name of a *class* that represents an entity or object is placed in parentheses, immediately following the entity or object.

The optimization model consists of the decision variables, the objective function, the set of constraints, and the fitness function. The objective function consists of a function (`Function`). A constraint (`Constraint`) is composed of the *classes*, `Function`, `Sign`, `PenaltyFunction` and an instance of `Bound`. The dashed ovals indicate that `Bound`, `Violation` and `Penalty` are instances of the same *class*. The arrows indicate the uni-directional associations that imply that one *class* in the participating association is visible to the other.

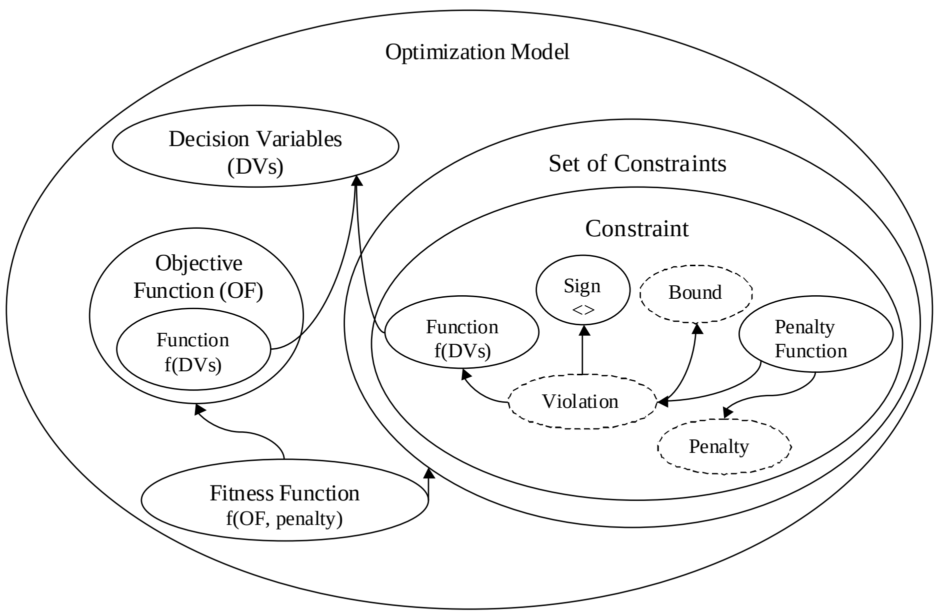


Figure 2.7 Optimization Model Structure

The aggregation hierarchy shown Figure 2.7 represents the mathematical definition of an optimization problem given by:

$$\begin{aligned}
&MAX \quad f(X) \\
&subject\ to: \\
&X \in F_{R^N} \\
&where: \\
&X = vector\ of\ decision\ variables \\
&f = objective\ function \\
&F_{R^N} = feasible\ search\ space
\end{aligned} \tag{2.2}$$

The feasible search space F_{R^N} is defined by the constraints that are represented as:

$$\begin{aligned}
&g_i(X) \le rhs_i \quad \forall i = 1, q \\
&h_j(X) = 0 \qquad \forall j = q+1, m \\
&where \\
&g_i, h_j = constraint\ functions \\
&rhs_i = the\ bound\ for\ constraint\ i \\
&q = number\ of\ inequality\ constraints \\
&m = total\ number\ of\ constraints
\end{aligned} \tag{2.3}$$

The objective function *f* and the constraint functions g_I and h_j can be expressed in terms of any combinations of standard functions of the decision variables. Without loss of generality, the optimization model described above considers maximization problems only. Minimization of the objective function *f* is achieved by maximizing the function *-f*.

2.5.2 A Framework for Solving Constrained Optimization Problems

It is difficult to develop a generalized constraint handling method that maintains the feasibility of solutions in a GA. Many approaches have been proposed for handling solutions that violate one or more constraints. Some of these approaches are described below:

2.5.2.1 Eliminating infeasible solutions

In this approach, infeasible *individuals* are eliminated from the population. It has been shown by Michalewicz (1995) that this approach is not effective for problems having a small ratio ($\frac{F_{R^N} \cap S_{R^N}}{S_{R^N}}$) of the feasible search space to the entire decision space S_{R^N}. Eliminating good *individuals* that slightly violate some constraints, but are not too far from the feasible search space, may result in loss of good genes.

2.5.2.2 *Repair operators for infeasible solutions*

New genetic material is introduced into the population by the crossover and mutation operators. The GA can be seeded initially with feasible *individuals* and the feasibility is maintained by application of repair operators. The design of this approach, however, is problem specific and involves having knowledge about a particular set of genes that causes the infeasibility. In complex problems, it may be difficult to check for all possible sets of genes for infeasibilities. The repair operators may add a significant computational burden to the GA, or they may distort some of the superior parent genes. In many cases, the problem of finding a feasible solution is itself NP-hard. The user should consider these shortcomings while designing the repair operators. The current GeGAOpt framework provides the user the flexibility to code specific repair operators such that feasibility is maintained, or use penalty functions to penalize the constraint violations.

2.5.2.3 *Penalty Function Approach*

Most methods proposed for handling constraints in GAs handle infeasible solutions by penalizing *individuals* for constraint violation. The constrained optimization problem is converted to an unconstrained problem by including the penalty explicitly in the objective function. This modified objective function becomes the fitness function for the GA. The modified fitness function is written as:

$$\begin{aligned}
&F(X) = F(f(X), f_p[p_1(X),..., p_i(X),..., p_m(X)]) \\
&where \\
&F = fitness\ \ function \\
&f = objective\ \ function \\
&f_p = function\ of\ \ constraint\ \ penalties \\
&p_i = penalty\ \ function\ \ for\ constrait\ i,\ \forall i = 1, m
\end{aligned} \tag{2.4}$$

The penalty for constraint *i*, p_i, is zero for no constraint violation and is negative otherwise. The functions *F* and f_p are based on the users' judgement of the relative significance of the constraint violations to the GA search procedure. These functions can be defined to impose an adaptive penalty that changes as the GA progresses The following penalty functions (Figure 2.1)are supported in the current implementation:

Linear Penalty Function

The penalty varies linearly with constraint violation and varies linearly with the generation number.

Exponential Penalty Function

The penalty varies exponentially with constraint violation and varies linearly with the generation number.

These penalty functions along with the function f_p can be used to formulate fitness functions used in most penalty approaches, notably those reported by Homaifar (1994), Joines and Houck (1994), Michalewicz and Attia (1994) and Harrell and Ranjithan (1999). The users can define the optimization problem and then experiment with various penalty functions and their associated parameters.

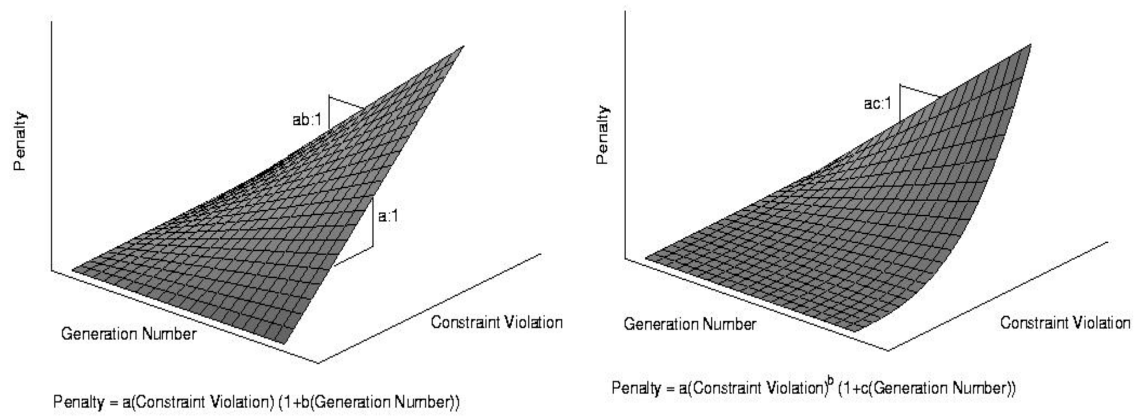


Figure 2.8 Linear and Exponential Penalty Functions

A problem that has equality constraints cannot be optimized easily for by GAs using the penalty approach. Even the slightest deviation from feasibility can cause an oscillatory behavior. In such cases, specifying a tolerance for constraint violation is known to result in stable convergence and improved performance. The constraint violation tolerance δ_{cv} can be specified for the constraints that are particularly hard to satisfy. The penalty for constraint *i* is then calculated as:

$$\begin{aligned} penalty_i &= 0 && \text{if } Violation \le \delta_{cv} \\ penalty_i &= p_i(X) && \text{otherwise} \end{aligned} \qquad (2.5)$$

It is difficult to find an ideal combination of the penalty functions, p_i, and the weights on the penalties that will yield feasible, near-optimal solutions. Frequently, this combination is highly problem specific and depends on the response surface of the actual problem domain. The penalties obtained by using the above penalty functions are not dynamic, i.e., they are not varied based on observed population characteristics. A number of adaptive penalty methods that use the population characteristics for scaling the penalties based on the severity of the constraints have been proposed in the GA literature; e.g. Siedlecki and Sklansky (1989) and Smith et al. (1996).

The difference in the fitness values of the best feasible solution and the best infeasible solution is a measure of the severity of the constraints. This measure can be used to adaptively adjust the contribution of the constraint penalties to the fitness function as follows:

$$\begin{aligned} &F(X) = f(X) + (f_b - F_{bf})\, f_p[p_1(X),...,p_i(X),...,p_m(X)] \\ &where \\ &f_b = objective\ value\ of\ the\ best\ solution \\ &F_{bf} = fitness\ value\ of\ the\ best\ feasible\ solution \end{aligned} \qquad (2.6)$$

In this formulation, the best feasible solution yet found is comparable to infeasible solutions that have higher fitness values. As the sum of the penalties is weighted by the term ($f_b - F_{bf}$), this term adjusts the magnitude of the imposed penalty during the progress of the GA search. This approach is adequate for representing various adaptive penalty formulations including the one by Smith et al. (1996). This procedure is included in the current implementation of the GeGAOpt.

2.5.3 A Flexible Optimization Model Representation

A graphical user interface (GUI) is developed to provide the user the capability of formulating the optimization model in an algebraic textual format that is similar to the one used by conventional solvers. To provide this capability, an "intelligent problem convertor" is developed that converts a users' mathematical definition of the problem to an internal problem representation that is based on the data structure shown in Figure 2.9.

Figure 2.9 The Intelligent Problem Converter

Providing the flexibility for users to define an optimization problem that consists of user defined decision variables and functions requires the use of an intelligent function parser that calculates the values of the user defined functions. The development of a function parser or interpreter is not a difficult task, but this results in considerable performance loss due to the fact the Java is an interpreted language itself.

To provide users the flexibility of defining a generic function optimization problem and to reduce the time associated with the evaluation, a Java Expressions Library (JEL), developed by Metlov (1998) is used. JEL enables the evaluation of the user-defined functions. JEL compiles mathematical expressions directly to Java byte-codes, allowing their fast evaluation. JEL supports all the Java primitive types as well as object types. Functions in JEL are methods of the Java objects. It does not require any front-end development for the functions. It is possible to use Java objects directly, exporting their functions to JEL. The integration of the optimization model with *classes* in JEL, the GA *classes*, and the linkage with the user interface are described in the next section.

2.6 Design Features

The design of the GeGAOpt is based on UML methodology. The implementation of this system is carried out using the Java™ platform. The Java programming environment provides a portable, interpreted, high-performance, simple, object-oriented programming language. This makes the design and implementation of the system simple, highly modular and promotes code reusability.

Figure 2.10 shows the GA *classes* and the inter-relationships among them. This figure is an UML class diagram that represents the conceptual GA structure discussed in Section 2.4.1. The gene *classes* as well as the *class* `Chromosome` implement the *interface* `DecisionVariables`. This facilitates the construction and evaluation of mathematical functions of objects that implement this *interface*. The *class* `GAUpdate` facilitates the synchronization of update threads (`UpdateThread`) spawned from the user interface seeking to update the GA operators and parameters.

Figure 2.11 shows the *classes* that facilitate the evaluation of an *individual*. This figure is an UML class diagram that represents the optimization model structure discussed in Section 2.5.1. The class `IndividualFunction` represents the optimization model and is responsible for the evaluation of the objective and fitness values. The arcs indicate the linkages between these *classes* and the GA data structure *classes*, the Java Expressions Library (JEL), and the user interface *classes*. The *class* `FunctionEvaluator` facilitates the evaluation of user-defined mathematical functions of the decision variables. The *class* `FitnessFunctionEvaluator` enables the evaluation of the fitness function that is a function of the objective function and individual constraint penalties. The user interface *classes* `InputPanel` and `ProblemEchoArea` are responsible for enabling users to input and visualize any function optimization problem. The *class* `ProblemFormulator` converts the users' definition of the optimization problem to an internal optimization problem representation. In this context, it represents the 'intelligent problem converter' in Figure 2.9.

Table 1 in Appendix B explains the class diagram conventions.

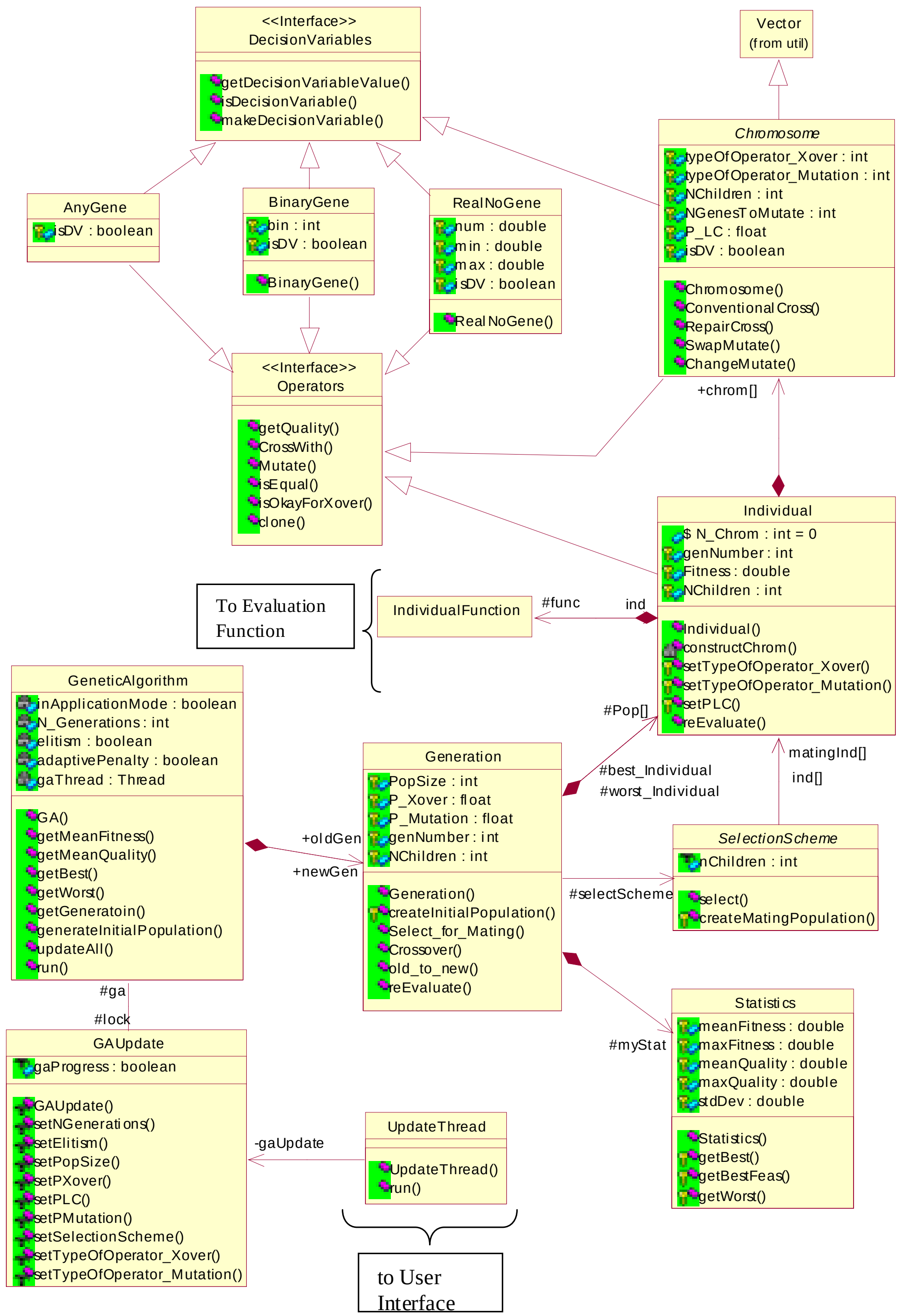


Figure 2.10 Class Diagram depicting the Genetic Algorithm *classes*

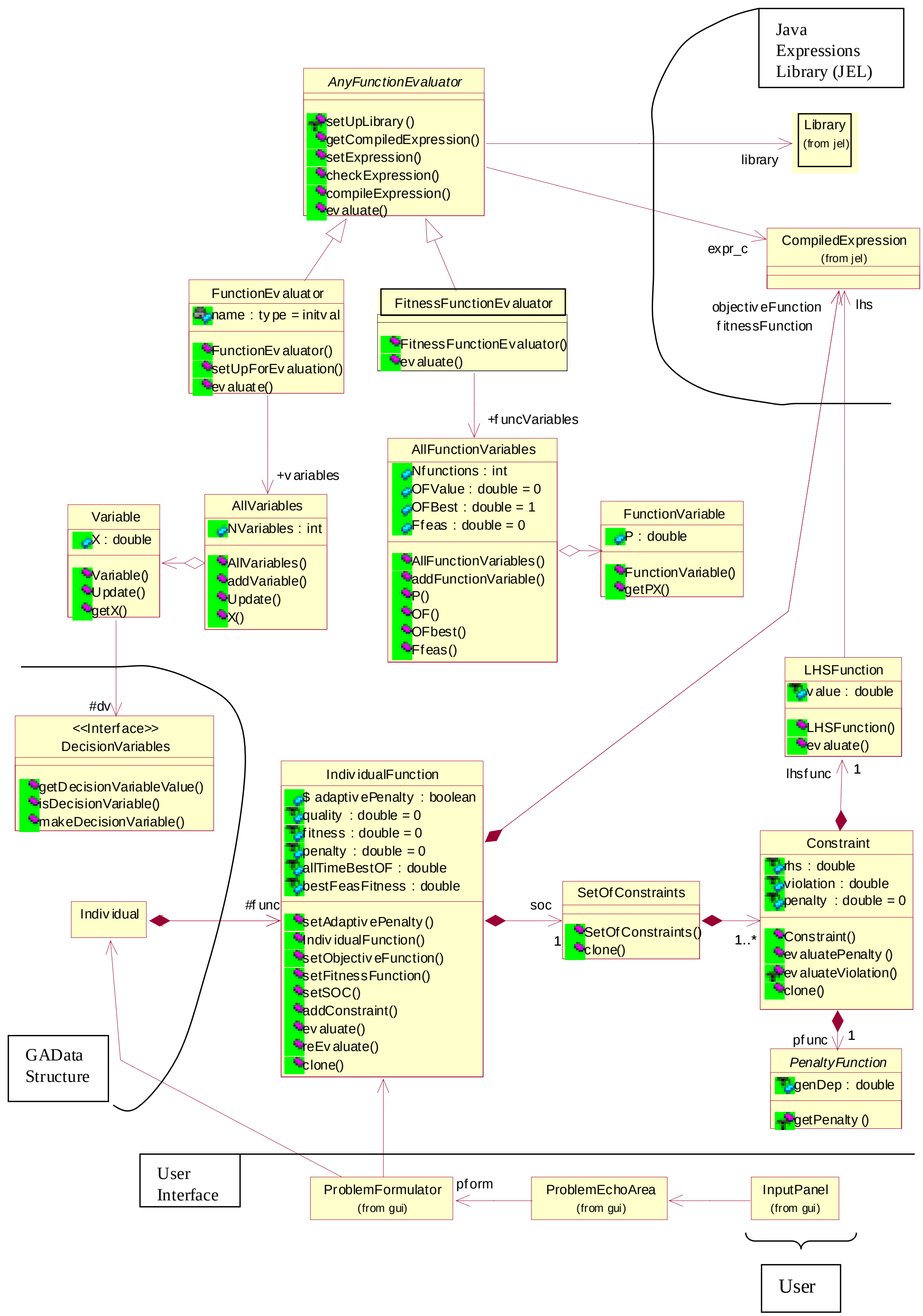
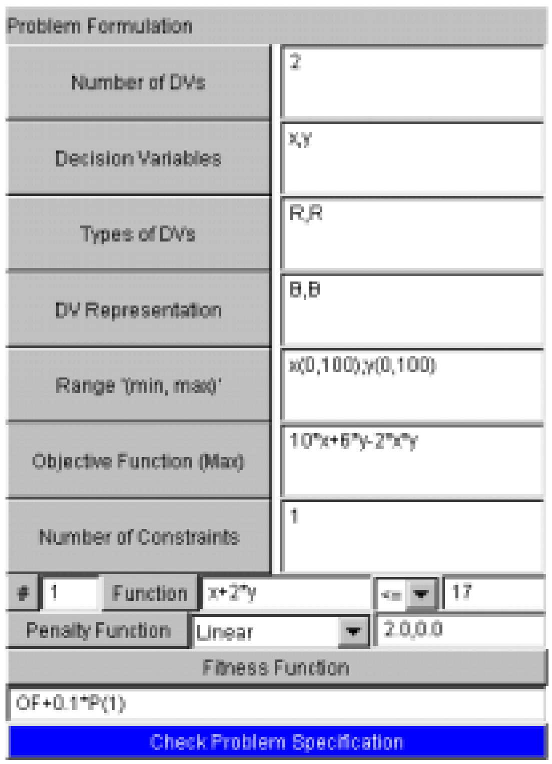


Figure 2.11 Class Diagram depicting the Optimization Model classes and linkages

2.7 GUI features – Interactive problem solving

2.7.1 Problem Formulation

The user can formulate a function optimization problem through the problem formulation interface by specifying decision variables, an objective function, constraints, and a fitness function. The problem formulation panel is shown on the left in Figure 2.12.




```
//Function Optimization Problem
//Saved file: omagosh.prb
//Non-Linear Constrained Optimization Demo
//Decision Space R(0,100) x R(0,100)
//Solution: at (x, y) = (17, 0)

Number of DVs: 2

Decision Variables: x,y

Types of DVs: R,R

DV Representation: B,B

Range '(min, max)': x(0,100);y(0,100)

Objective Function (Max): 10*x + 6*y - 2*x*y

Number of Constraints: 1

CONSTRAINTS
#: 1
x + 2*y <= 17
Penalty Function: Linear
Parameters: 2.0,0.0

Fitness Function: OF + 0.1*P(1)
```

Figure 2.12 Problem Formulation Panel and Sample Input File

The problem formulation can be stored in the form of specially formatted files. An example of an input file is shown on the right in Figure 2.12. The input file follows a free algebraic format. This gives the user the capability of formulating constrained, non-linear or combinatorial problems in a format similar to that used in conventional solvers. The user can then solve the problem by selecting from a combination of various decision variable representations, genetic operators and penalty functions.

2.7.2 Controlling the GA Run

The user can dynamically control the GA parameters and operators listed below.

- Crossover Probability
- Mutation Probability
- Rate of recombination
- Type of crossover
- Type of mutation
- Selection scheme
- Elitism
- Population Size
- Number of Generations
- Seed for the random number generator.

Any combination of these parameters can be changed by the user at any point during the course of the GA run. The changed parameters take effect in subsequent generations. Based on the number of generations, the number of iterations that the GA population is allowed to evolve, the GA thread is suspended when this number is reached. The user can then either increase the number of generations to continue the GA run, or stop the GA run. The values for these parameters can be set by the user via the control panel which is shown on the left in Figure 2.13.

2.7.3 Monitoring Progress

The user can follow the progress of the GA run by monitoring the following:

- **Fitness** function, **objective** function and **penalty** function(s) values. A dynamic display plots the convergence of the GA. The penalty values for the constraints can also be viewed.
- **Decision space** that depicts the values of the decision variables in the current population (supported only for a two-variable problem).
- **Objective space** that depicts the values of the objective and fitness functions of *individuals* in the current population (supported only for a two variable problem).

- **Generation Statistics**. The best as well as the mean fitness and objective function values of *individuals* and the standard deviation of the fitnesses of *individuals* in a population can be viewed on another dynamic display.
- **The best solution**. The characteristics of the best solution, i.e., the decision variable values, the constraint violations, penalties, objective function and fitness values can be monitored.

A display of these outputs is shown on the right in Figure 2.13.

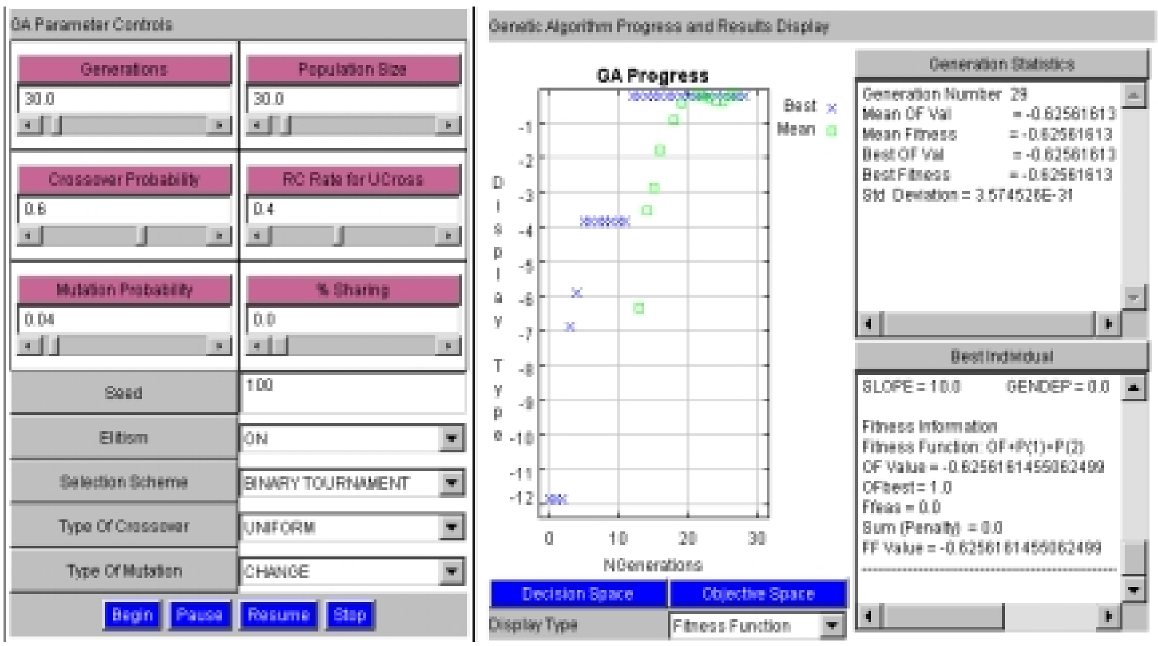


Figure 2.13 Control Panel and the Results Display

2.8 Test Problems and Results

The GA class library and the GUI implemented within GeGAOpt was used for solving interactively unconstrained as well as constrained function optimization problems. These test problems were selected from GA literature. The GeGAOpt features that provide users flexibility in defining an optimization problem, choosing the type of representations, experimenting with different penalty function formulations, selecting from a range of genetic operators and tweaking the GA parameters during the course of the GA run were utilized in solving these test problems. The following sections discuss the test problems, the solution methodology and the GA run characteristics.

2.8.1 Unconstrained Optimization Problems

Table 2.1 shows the results of a series of GA runs for unconstrained function minimization problems. It reports the best, mean, and the worst *solution* found over a series of five independent runs that were carried out with different random number seeds. The *solution* over a run is the *individual* that has the best objective function value. A population size of 30 was used for these problems. The crossover probability varied from 0.5-0.7 and the number of generations required for convergence varied from 50-100. The source of each function optimization problem is shown in italicized parentheses below the function name.

Table 2.1 Results for Unconstrained Optimization

#	Problem	Problem Description	Bounds on Decision Variables	Known Solution		Objective Function Value
1	DeJong F1 *(DeJong, 1975)*	$Min \sum_{i=1}^{3} x_i^2$	$-5.12 \le x_i \le 5.12$ $\forall$ i = 1,2,3	Min = 0 $(x_1,x_2,x_3) = (0,0,0)$	*best* *medium* *worst*	0 -6.67E-6 -2E-5
2	DeJong F2 *(DeJong, 1975)*	$Min\ 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$-2.048 \le x_i \le 2.048$ $\forall i = 1,2$	Min = 0 $(x_1,x_2,) = (1,1)$	*best* *medium* *worst*	0 0 0
3	Rosenbrock's Function *(Mathworks, 1997)*	$Min\ 100(x_2 - x_1)^2 + (1 - x_1)^2$		Min = 0 $(x_1, x_2) = (1,1)$	*best* *medium* *worst*	0 0 0
4	Goldstein-Price Function *(Goldstein and Price, 1971)*	$Min\ [1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 +$ $3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)].$ $[30 + (2x_1 - 3x_2)^2 (18 - 32x_1 +$ $12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	$-2 \le x_i \le 2$ $\forall i = 1,2$	Min = 3 $(x_1,x_2) = (0,-1)$	*best* *medium* *worst*	3 3.001947 3.0057
5	Test UF1 *(Michalewicz, 1996)*	$Max - \{21.5 + x_1.\sin(4\pi x_1) +$ $x_2.\sin(20\pi x_2)\}$	$-3 \le x_1 \le 12.1$ $4.1 \le x_2 \le 5.8$	Min = 38.85 $(x_1,x_2) = (11.626, 5.725)$	*best* *medium* *worst*	38.85 38.182 37.127
6	Test UF2 *(Mathworks, 1997)*	$Min\ e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1)$		Min = 0 $(x_1, x_2) = (0.5, -1)$	*best* *medium* *worst*	0 0 0

2.8.2 Constrained Optimization Problems

Table 2.2 reports the results of a series of GA runs for constrained function optimization problems. It reports the best, mean and the worst *solution* found over a series of five independent runs that were carried out with different random number seeds. The *solution* over a run is the *individual* that has the best objective function value and is feasible with respect to all the constraints. A population size of 50 was used for problems TestCF1 through TestCF4. The crossover probability was varied from 0.5-0.7 and the number of generations required for convergence varied from 50-100.

A population size of 60-70, a generation number of 300-400, crossover probability of 0.4 and a mutation probability of 0.04 were used for problems TestCF5, TestCF6 and TestCF6. The value $r_f = \frac{F_{R^N} \cap S_{R^N}}{S_{R^N}}$, which indicates the ratio of the feasible search space to the total search space, reported by Michalewicz (1995) for these problems are also shown in this table. The low r_f values indicate that these are highly constrained problems. Due to the highly constrained nature, the performance of the GA search is sensitive to the penalty functions and fitness formulations used in solving these problems. A suitable set of the penalty functions was identified through a trial-and-error process and then five independent runs were carried out with different random number seeds.

The performance of the GA can be improved further by fine tuning the penalty and fitness functions, and running the GA for a larger number of generations with lower crossover probability.

Table 2.2 Results for Constrained Optimization

#	Problem	Problem Description	Bounds on Decision Variables	Known Solution		Objective Function Value
1	Test CF1	$Max: 10x + 6y - 2xy$ $st.\ x + 2y \le 17$		Max = 170 $(x_1,x_2) = (17,0)$	*best* *medium* *worst*	170 169.62 169.12
2	Test CF2 *(Mathworks, 1997)*	$Min: e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1)$ $st:$ $1.5 + x_1x_2 - x_1 - x_2 \le 0$ $-x_1x_2 <= 10$		Min = 0.0236 $(x_1,x_2,) = (-9.5474, 1.0474)$	*best* *medium* *worst*	0.0236 0.0258 0.0296
3	Test CF3 *(Mathworks, 1997)*	#2	$x_1 \ge 0,$ $x_2 \ge 0$	Min = 8.5 $(x_1,x_2) = (0,1.5)$	*best* *medium* *worst*	8.5 8.5 8.5
4	Test CF4 *(Floudas and Pardolas,. 1997)*	$Min\ (-12*x - 7*y + y*y)$ $st. -2x^4 + 2 - y = 0$		Min = -16.7389 $(x, y) = (0.71751,1.470)$	*Best* *medium* *worst*	-16.739 -16.728 -16.703
5	Test CF5 *(Michalewicz, 1995)*	$Min: 5(\sum_{i=1}^{4} x_i - x_i^2) - \sum_{i=5}^{13} x_i)$ $st.$ $2(x_1 + x_2) + x_{10} + x_{11} \le 10$ $2(x_1 + x_3) + x_{10} + x_{12} \le 10$ $2(x_2 + x_3) + x_{11} + x_{12} \le 10$ $-8x_i + x_{9+i} \le 0\ \forall i = 1,2,3$ $-2x_4 - x_5 + x_{10} \le 0$ $-2x_6 - x_7 + x_{11} \le 0$ $-2x_8 - x_9 + x_{12} \le 0$	$0 \le x_i \le 1\ \forall i = 1,...,9$ $0 \le x_i \le 100, i = 10,11,$ $0 \le x_{13} \le 1$	Min = -15 X = (1,1,1,1,1,1,1,1,1,3,3,3,1) Six constraints active at the global optimum all except, $-8x_i + x_{9+i} \le 0\ \forall i = 1,2,3$ $\frac{F_{R^N} \cap S_{R^N}}{S_{R^N}} = 1.11E-4$	*best* *medium* *worst*	-14.91 -14.21 -13.64

Table 2.2 Continued

#	Problem	Problem Description	Bounds on Decision Variables	Known Solution		Objective Function Value
6	Test CF6 *(Michalewicz, 1995)*	$Min: (x_1 - 10)^2 + 5(x_2 - 12)^2 +$ $x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 +$ $x_7^4 - 4x_6x_7 - 10x_6 - 8x_{7)}$ $st.$ $127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0$ $282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0$ $196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0$ $-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0$	$-10 \leq x_i \leq 10$ $\forall i = 1,...,7$	Min=680.63 X= (2.3305, 1.9514, -0.4775, 4.3657, -0.6245, 1.0381, 1.5942) The first and last constraints are active at the global optimum. $\frac{F_{R^N} \cap S_{R^N}}{S_{R^N}} = 5.121E-3$	*best* *medium* *worst*	681.34 684.93 688.8
7	Test CF7 *(Michalewicz, 1995)*	$Min: x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2$ $+ 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2$ $+ 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 +$ 45 $st.$ $105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0$ $-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0$ $8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0$ $-3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0$ $-5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0$ $-x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0$ $-0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0$ $3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0$	$-10 \leq x_i \leq 10,$ $i = 1,...,10$	Min =24.306 X= (2.172, 2.3639, 8.7739, 5.096, 0.9906, 1.4306, 1.3216, 9.8287, 8.2801, 8.376) Six out of eight constraints are active at the global optimum (all except the last two) $\frac{F_{R^N} \cap S_{R^N}}{S_{R^N}} = 3E-6$	*best* *medium* *worst*	24.49 26.536 38.439

2.9 Conclusions and Recommendations

2.9.1 Conclusions

The GA object classes and the optimization model classes can easily be customized for application to a wide range of problems and be used in education and research. The GeGAOpt developed in this study facilitates the formulation of non-linear unconstrained as well as constrained optimization problems and their solution using the underlying GA framework. This makes it possible for a user familiar with mathematical programming to solve optimization problems using GAs and explore the effect of various representations, genetic operators, penalty functions, and static and adaptive penalty techniques on the GA search. The results discussed in Section 2.8 effectively demonstrate the use of the GeGAOpt in this solving various unconstrained as well as constrained optimization problems.

This framework has been developed based on OO paradigm and design methodology. This makes it easier for further development and enhancement of the current capabilities. The framework is generic in that, it can be easily extended to represent complex chromosomal data structures such as trees and networks.

The graphical user interface to this framework currently enables only the formulation of function optimization problems. The availability of the GeGAOpt via the internet enables users to explore the features of the GeGAOpt with the help of a Java compatible web browser.

2.9.2 Recommendations for Future Development

The current framework can be extended to support two as well as three-dimensional matrix representation of *chromosomes* and complex data structures such as *n- ary* trees, *graphs* and networks. Genetic operators that are appropriate for these data structures have been discussed in GA literature. These operators can be implemented so that other conventional optimization problems, such as the Travelling Salesman Problem (TSP), the Knapsack Problem, and Network Optimization problems, can be solved. This will further enhance the ability of this framework to solve other *classes* of optimization problems using the underlying GA methodology.

References

1. Booch, G., Rumbaugh, J. and Jacobsen, I., (1997). *Unified Modeling Language for Object-Oriented Development*, V 1.1, Rational Software Corporation, Santa Clara, CA.
2. De Jong, K.A., (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Doctoral dissertation, University of Michigan, Abstract International}, 36(10), 5140B, (University Microfilms No 76-9381).
3. Floudas, C.A., and Pardalos, P.M., (1987). *A Collection of Test Problems for Constrained Global Optimization Algorithms*, Springer-Verlag, Lecture Notes in Computer Science, Vol.455.
4. Fowler, M. and Scott, K., (1997). *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley.
5. Goldberg., D. E., (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley.
6. Goldstein, A.A and Price, J.F., (1971). *On descent from local minima*. Mathematics of Computation, 25:569-574.
7. Gosling, J., Joy, B., Steele, G, (1989). *The Java Language Specification*, Addition-Wesley Publishing Co., Inc., Reading, MA.
8. Joines, J.A., and Houck, C.R., (1994). *On the Use of Non-stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems with GAs*, in Proceedings of the First IEEE Conference on Evolutionary Computation, pp. 579-584.
9. Harrell, L.J., Ranjithan, S.R., (1999). *Evaluation of Alternative Penalty Function Implementations in a Watershed Management Design Problem*, Proceedings of the Genetic and Evolutionary Computation Conference, July 13-17 at Orlando, FL, pp. 1551-1556.
10. Hessner, J.; Männer, R., (1991). in Proceedings of the First Workshop on Parallel Problem Solving from Nature (Lecture Notes in Computer Science, Vol. 496); P. Schwefel and R. Männer (Eds.); Springer-Verlag, Berlin, pp 23-31.
11. Holland, J.H, (1975). *Adaptation in Natural and Artificial Systems*; University of Michigan Press: Ann Arbor, MI.

12. Homaifar., A., Lai, S.H., Qu, X., (1994). *Constrained Optimization via Genetic Algorithms*, Simulation, Vol. 62, No. 4, pp. 242-254.
13. Houck, C. R., Joines, J.A. and Kay, M. G., (1995). A genetic algorithm for function optimization: A Matlab implementation . NCSU-IE Technical Report 95-09.
14. Mathworks, (1997). *Matlab v. 5.3 Manual*, URL: file:/afs/bp.ncsu.edu/dist/matlab53/help/fulldocset.html.
15. Metlov, C., 1998. *Java Expressions Library (JEL) Reference Manual*, URL: http://galaxy.fzu.cz/JEL.
16. Michalewicz, Z., (1996). *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag.
17. Michalewicz, Z., (1995). *Genetic Algorithms, Numerical Optimization and Constraints*, Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan-Kaufman, pp.151-159.
18. Michalewicz, Z., and Attia, N., (1994). *Evolutionary Optimization of Constrained Problems*, Proceedings of the 3rd Annual Conference on Evolutionary Programming, World Scientific, pp. 98-108.
19. Object Management Group (1997). *UML 1.1 Documentation*. URL: http://www.rational.com/.
20. Siedlecki, W., and Sklansky, J., (1989). *Constrained Genetic Optimization via Dynamic Reward-Penalty Balancing and Its Use in Pattern Recognition*, in Proceedings of the Third International Conference on Genetic Algorithms, pp. 141-150.
21. Smith, A.E., Coit, D.W., and Tate, D.M., (1996). *Adaptive Penalty Methods for Genetic Optimization of Constrained Combinatorial Problems*, INFORMS Journal on Computing, Vol. 8, No. 2, pp. 173-182.
22. Tamiz, M. and Mardle, S.J., (1998). *An interactive graphics-based linear, integer and goal program modeling environment*, Decision Support Systems, 23, pp. 285-296.
23. Wall, M., (1995). *GALib: A C++ Library of Genetic Algorithm Components*, URL: http://lancet.mit.edu.

3 Decision Support Framework for Integrated Watershed Water Quality Management

3.1 Introduction

Environmental problems associated with water quality in watersheds and associated ecosystems are becoming increasingly critical. These problems are inherently complex and are difficult to analyze due to conflicting issues such as cost, environmental impact and equity that need simultaneous consideration while evaluating alternative management strategies. Solutions to these problems require an integrated analysis and modeling of the watershed system. Traditionally, the water quality problems and subsequent degradation of environmental resources have been attributed to point sources. In recent years, despite strict controls on point sources, the adverse affects on water quality in watersheds have not decreased proportionately. It has been well documented (Line et.al, 1997) that non-point sources such as nutrient runoff and atmospheric deposition are significant sources of pollution. The modeling of these sources and their control is a complex problem that transcends city or county boundaries. A watershed is a hydrologic unit that can be used for effectively addressing the water quality issues. The watershed approach is a coordinating framework for environmental management that focuses public and private sector efforts to address the water quality problems in an integrated manner within hydrologically defined geographic areas. The focus of this research is to develop an integrated watershed management approach that can be used for the characterization and solution of the water quality problems within watersheds in an efficient manner.

3.1.1 Current Frameworks for Watershed Management

A framework that integrates point and non-point source data, monitoring and meteorological data, and environmental simulation models is required to address the watershed management problem in a holistic manner. Better Assessment Science Integrating Point and Nonpoint Sources (USEPA, 1998) is a multi-purpose environmental analysis system used in performing watershed water quality studies. It integrates environmental data, analytical tools and modeling tools to support these studies. It facilitates the assessment of large amounts of point source and nonpoint source data and the water quality at selected stream sites.

An integral part of watershed management is the use of watershed models to characterize the ambient water quality. Some of the existing models are Storm Water Management Model (SWMM), Soil and Water Assessment Tool (SWAT), and Hydrologic Simulation Program – FORTRAN (HSPF). These models are mechanistic simulation models that use weather driven non-point source sub-models and river reach contaminant transport models to model the water quality indicators on land segments and in river reaches.

3.1.2 Limitations of Current Watershed Management Frameworks

The solution to a watershed management problem requires the formulation of control strategies that meet the water quality targets for particular water bodies. A control strategy is defined by a set of controls that curb source emissions. Within the current scope of BASINS, a user can employ existing computational utilities to develop alternative total maximum daily load (TMDL) strategies to achieve desired water quality targets. This approach is a trial-and-error process. It is difficult to modify the point and non-point source information in the BASINS databases or the input files to the watershed model (HSPF) to simulate the water quality impacts of a particular control strategy. Given the large number of combinations of the decision variables, it is unlikely that a random search will identify a cost efficient control strategy that meets reliably the water quality targets. A systematic search can be used to identify feasible as well as cost efficient control strategies that achieve desired water quality targets with acceptable reliability and perform well with respect to unmodeled issues such as equity.

3.1.3 Objectives of the Decision Support Framework

Decision support frameworks facilitate efficient search for ‘good’ control strategies. A ‘good’ control strategy is one that is cost efficient and meets the water quality targets. The objectives of this study are to develop a decision support framework to assist decision-makers in efficiently exploring watershed management alternatives through creation, modification, evaluation and comparison of various control strategies.

This framework enables the efficient search for cost efficient and robust control strategies by the inclusion of additional systems analytic tools such as optimization algorithms uncertainty analysis procedures. This system enables a decision-maker to analyze the water quality impacts, the cost-benefits, the equity issues and the reliability of the following control strategies.

Point Source Control Strategies

- Command and control (CAC) regimes
- Effluent charge programs
- Transferable discharge permit (TDP) programs
- Optimization of point source control strategies

Non-point Source Control Strategies

- Land use planning
- Riparian buffer zoning
- Regional detention ponds

The optimization algorithms integrated into this framework enable the user to identify optimal control strategies with respect to several criteria including cost, water quality and equity. Robust decision making calls for explicit quantitative analysis of the effect that the parameter uncertainties have on the outcome of any control strategy. The uncertainty analysis procedure facilitates the explicit quantification of the uncertainty in achieving the water quality targets.

Founded on this decision support framework, a decision support system (DSS) is implemented to enhance the current capabilities of BASINS with these additional watershed management capabilities. Section 3.2 describes individual components of the DSS and their capabilities. Section 3.3 describes the inter-linkages among these components and their integration into the DSS.

3.2 DSS Components and Capabilities

3.2.1 BASINS

BASINS integrates a geographic information system (ArcView GIS), national watershed data, and state-of-the-art environmental assessment and modeling tools into one convenient software system. This makes watershed and water quality studies easier by bringing key data and analytical components together under a single platform. The significant features of BASINS are the following.

- It facilitates the examination of environmental information.
- It provides a watershed modeling framework that integrates national databases, watershed models such as the Hydrological Simulation Program Fortran (HSPF) and other watershed analysis utilities.
- It supports analysis of point and nonpoint source management alternatives.
- It supports the development of total maximum daily loads (TMDLs).

The current version of BASINS incorporates the Hydrologic Simulation Program Fortran (HSPF) to simulate hydrologic processes. BASINS uses processed emission, meteorological and other data as an input to HSPF for modeling the watershed water quality.

3.2.2 Watershed Water Quality Modeling with HSPF

HSPF is a comprehensive model for simulation of watershed hydrology and water quality for both conventional and toxic organic pollutants. Contaminants that can be modeled using HSPF include organic tracers, sediments, carbon, nitrogen and phosphorous cycles, etc. HSPF incorporates various submodels into a basin-scale framework for water quality analysis that includes pollutant transport and transformation in one- dimensional stream channels. This model allows the integrated simulation of land based pollutant runoff processes with point sources and in-stream contaminant transport and sediment chemical interactions.

Each model segment contains information generated by a hydraulic submodel, a nonpoint source submodel, and a river submodel. The hydraulic submodel uses rainfall, evaporation, and meteorological data to calculate runoff and subsurface flow for all the basin land uses including pervious (PERLND) and impervious (IMPLND) lands. The surface and subsurface flow ultimately drives the nonpoint source submodel that simulates soil erosion and the pollutant loads from the land to the river reaches. The river reach submodel (RCHRES) routes flow and associated pollutant loads from the land through the lakes, rivers, and reservoirs.

The result of these simulations is a time history of runoff flow rate, sediment load, nutrient and organic pollutant concentrations, along with a time history of water quality and quantity at any location in the river reach file.

3.2.3 Control Strategies for Watershed Management

A control strategy is defined by a set of controls that curb source emissions. Point source emission controls may consist of particular control technologies or processes. Non-point source emission controls may consist of approaches such as land use planning to reduce pollutant runoff and implementing best management practices (BMPs) to reduce the non-point source runoff and concentration. The decision variables that characterize a control strategy are the emission controls, their efficiencies and costs, the water quality impacts of the new emissions that result due to the application of emission controls, etc.

The formulation of any practical policy for pollution abatement requires the analyses of the costs of implementing the targeted reductions in pollutant loads. Typically, the implementation of a control strategy on a system-wide scale can result in annualized treatment costs of the order of millions of dollars to the sources that are targeted by the strategy. Hence, it is imperative to estimate the costs involved in controlling the various sources in the design stages of the control strategy. The relevant costs in watershed management are the treatment costs incurred by point sources, the economic and social costs of the land use plans, and the costs associated with the implementation of various best management practices (BMPs). The eventual goal of this decision support system is to integrate the cost information within a single framework.

Currently, the framework has the capability of including the point source treatment cost in the analyses. An already existing detention pond cost module developed by Harrell (1998) will be integrated in the future into the decision support framework.

3.2.4 Analysis of Control Strategies

Analysis of controls strategies consists of defining the decision variables of the individual control strategies and performing a simulation of the water quality indicators to assess the environmental impact and the costs of the control strategies. The control strategy can be defined by users through a convenient interface. The DSS then performs the necessary data manipulations to reflect the changes in the input files (UCI) required by the watershed model (HSPF), runs the model, and then displays the results.

3.2.4.1 Point Source Management

Analysis of point source control strategies consists of defining point source emission values either directly or indirectly and simulating the water quality indicators to assess the environmental impact and the incurred treatment costs.

Command and Control

Traditionally, command and control (CAC) approaches have been used to control point source emissions. The regulatory agency mandates the reduction of the emissions by a certain amount. Uniform reduction consists of applying a certain emission reduction to all baseline point source emissions. In zoned uniform reduction, the point sources are classified into groups and a uniform reduction is applied to each group. These regulations do not account for the geographical locations of the point sources, and therefore, are inefficient in terms of incurred treatment cost. The DSS facilitates the analysis of CAC strategies.

Effluent Charges

An effluent charge is a charge levied by a regulatory agency that is intended to provide an economic incentive for a discharger to reduce their emission to an acceptable level (Brill, 1997). The discharger is expected to increase the efficiency of controlling the emission as long as the marginal cost of doing so is less than the incremental cost of paying the charge. The DSS enables a user to assess the response of the dischargers to specific effluent charges and estimate the associated water quality impacts. This feature can also be used in an iterative analysis (Brill 1997) to determine the effluent charge that ensures that the water quality targets are met. Analysis of zoned effluent charge programs can be performed by grouping the point sources into groups.

3.2.4.2 Non Point Source Management

Land Use Planning Analysis

Since 1973, the U.S. Water Resources Council has recommended the inclusion of national economic development and environmental quality as two essential non-commensurable objectives in water and related land resource planning.

Land use planning analysis consists of defining either directly or indirectly the incremental changes in land use plans and simulating the change in the water quality indicators. The user defines incremental changes to the baseline land use plans of various land use management units. Sub-watersheds and watersheds are examples of land use management units. The DSS then performs a water quality simulation for the changed land use scenarios and presents results to the user. This enables users to perform a 'what-if' analysis that can be used to refine benchmark land use planning solutions to facilitate iterative decision making.

Analysis of Best Management Practices (BMPs)

Various BMPs can be used to reduce the weather driven non-point pollution processes. Examples of such BMPs are riparian buffers and system-wide detention ponds. Harrell (1998) developed a procedure for the estimation of the costs and environmental benefits of system-wide detention ponds. The watershed model used (HSPF), however, is not designed to simulate directly the effect of other non-point source controls such as buffer zoning and detention ponds. With the incorporation of these capabilities within HSPF, the current DSS framework can be extended easily to facilitate analysis and optimization of these other BMPs

3.2.5 Optimization Models for Control Strategies

Optimization algorithms can be used to enhance to the practices used in environmental management and decision making. Lack of use of optimization in solving environmental problems can be attributed to several reasons such as the complex nonlinear nature of contaminant transport processes and the inadequacy of representing these processes by explicit mathematical functions. On the other hand, the formulation of system-wide or regional control strategies for pollution abatement requires the detailed analyses of the cost and environmental impacts of the control strategy. When designing a control strategy for a region with a large number of varied pollutant sources such as point source dischargers, non-point source runoff, agricultural nutrient loading, the number of potential solutions is very large. There has to be an efficient method of searching through this non-finite set of potential solutions to identify a 'good' control strategy.

Typically, optimization implementations in water quality management seek to use information, such as point source inventories, available costs and contaminant transport models to predict cost efficient ways of complying with the water quality standards. An optimization procedure facilitates a systematic search for ‘good’ control strategies.

3.2.5.1 Point Source Control Strategy Optimization

The point source control strategy can be optimized for point source treatment costs and water quality impacts. The optimization model consists of defining the decision variables, objective function and the constraints.

Optimization Model Formulation

Primary Decision Variables: The choice of control technologies that are used to curb the point source emissions and their efficiencies are the primary decision variables.

$$\begin{aligned} x_{ij} &= 1 \text{ if control technology } j \text{ is used at source } i, \\ &\quad 0 \text{ otherwise} \end{aligned} \tag{3.1}$$

Dependent Decision Variables: The application of control technologies results in reduced emissions. These reduced emissions are a function of the applied control technologies and hence are the derived decision variables.

$$\begin{aligned} & e_i = f_\eta(\eta_{i1}, \eta_{i2}, \ldots, \eta_{in}) \\ & W_i^f = W_i^s (1 - e_i) \\ & \text{where} \\ & e_i = \text{total emission control efficiency for source } i \\ & \eta_{ij} = \text{efficiency of control technology } j \text{ at source } i \\ & n = \text{number of control technology options at source } i \\ & f_\eta = \text{a function that models the combined effect of the} \\ & \qquad \text{individual control technologies} \\ & W_i^f = \text{final emission level for source } i \\ & W_i^s = \text{initial emission level for source } i \end{aligned} \tag{3.2}$$

In cases where the emission reduction cannot be strictly attributed to set of control technologies, the reduced emissions may be treated as the decision variables. For example, this approach might be necessary for example, in the case of the removal of nitrates and ammonia from effluents. This treatment process cannot be modeled as discrete control technologies. Hence, the final emissions along with the appropriate cost curves are used for modeling the control strategy.

Objective Function: The objective function is typically a cost function that sums the treatment costs incurred by the point sources.

$$\begin{aligned}
&Total\ Cost = \sum_i \sum_j c_{ij} x_{ij} \\
&\text{where} \qquad (3.3) \\
&c_{ij} = \text{cost of control technology } j \text{ applied at source } i
\end{aligned}$$

The cost may also be obtained from treatment cost curves particular sources that estimate the costs of treatment (c_i) as a function of the emission reduction (e_i).

Constraints: The total amount of emissions is constrained in an emissions least cost(ELC) formulation as indicated by Eq. 3.4.

$$\begin{aligned}
&\sum_i W_i^f \le E_T \qquad (3.4) \\
&E_T = \text{Emission Reduction Target}
\end{aligned}$$

To identify cost efficient control strategies that meet the water quality targets the ambient least cost (ALC) formulation is used. If the environmental impacts of the source emissions are quantifiable, then the ALC constraints are represented as:

$$\begin{aligned}
&\sum_i W_i^f a_{ik} \le std_k \ \ \forall k \\
&\text{where} \qquad (3.5) \\
&a_{ik} = \text{water quality impact of source } i \text{ at location } k \\
&std_k = \text{water quality standard at location } k
\end{aligned}$$

The aggregate water quality impacts may also be obtained by running a watershed model such as HSPF.

The ELC and ALC formulations are discussed in the context of air quality management by Loughlin (1998).

3.2.5.2 Transferable Discharge Permit (TDP) programs

Transferable discharge permit (TDP) programs are market-based strategies for reducing pollutant loading. In a TDP program, a permit represents the right of a discharger to discharge a certain amount of a pollutant. The total number of permits is controlled by the regulatory agency, effectively limiting the total amount of pollutant discharged into the environmental system. After the initial allocation of permits, the permits may be bought or sold among interested parties. The strength of a TDP program is the flexibility that sources have in complying with the standards. Those sources with higher marginal cost of treatment can buy excess permits from sources with lower marginal costs. The result would be an efficient solution in terms of cost and equity to the point source management problem. The water quality is influenced by the geographical locations of the permits. Hence, the effectiveness of a TDP program in achieving ambient water quality targets needs careful evaluation. Brill et al. (1984) describe a framework to analyze the water quality impacts of BOD under transferable discharge permit programs. Loughlin et al. (1997) and Gillon (1999) have also done similar work in the air quality area. Tietenberg (1985), Atkinson and Tietenberg (1991) discuss various mathematical formulations of TDPs.

The objective of this component of the DSS is to develop an optimization procedure that can be used for the analysis of a TDP program implemented in any river basin. The procedure developed can used to estimate the worst case and best case scenarios resulting under the TDP program and to obtain a least cost solution that can be used as a benchmark for identifying management strategies.

Initially each discharger is allocated permits equal to its emission reduced by the targeted reduction factor $\boldsymbol{r_t}$. It is assumed that the dischargers behave in a rational manner; there is no accumulation of unused permits and the market is ideal. A mathematical programming approach for simulating scenarios resulting under a TDP program is given below.

Optimization Model Formulation

Primary Decision Variables: Individual permit trades:

$$
\begin{aligned}
& t_{ij} \quad \forall (i,j) \in (N,N) \\
& T_i = \sum_{j=1}^{N} t_{ij} \qquad \forall \mathrm{j} \neq \mathrm{i} \\
& \text{where:} \\
& t_{ij} \text{ represents a permit trade between dischargers } i \text{ and } j \\
& T_i \text{ is the total amount of permits traded by discharger } i \\
& N \text{ is the total number of dischargers participating in the TDP}
\end{aligned}
\tag{3.6}
$$

In a real market, a typical discharger either buys permits or sells permits. If discharger *i* is a buyer then all the dischargers *j* with which discharger *i* trades will be sellers. The following *(N X N)* matrix of individual trades (***T***) represents the entire TDP scenario.

$$T = \begin{vmatrix} 0 & t_{12} & . & t_{1n} \\ t_{21} & 0 & . & t_{2n} \\ . & . & 0 & t_{3n} \\ t_{n1} & . & . & 0 \end{vmatrix}$$

where: (3.7)

T = represents all the trades occuring under a TDP

It suffices to model the upper triangular matrix of trades, since the above matrix ***T*** is skew symmetric.

<u>Dependent Decision Variables</u>: The final emission levels result due to the permit trading and are hence, the dependent decision variables.

$$W_i^f = W_i^s + T_i \qquad \forall i \in (1, N) \tag{3.8}$$

where

T_i = traded permits

N = Number of dischargers

<u>Objective Function</u>: Optimize for the objective function $F = F(W_i^f)$. The objective function ***F*** used for a least cost simulation is a cost function that sums the treatment costs of all the dischargers. ***F*** used for the estimation of the best and worst case water quality impacts is a function of the water quality indicators.

<u>Constraints</u>: The above problem is constrained by constraints on total emission levels or constraints on the predicted water quality impacts. The problem is additionally constrained by a set of Trading Restrictions (***R***).

Formulation of Trading Restrictions

Each discharger has a minimum limit below which it cannot treat its waste load. This limit is imposed by the existing control technology for the emission reduction. The following set ***R*** represents a set of trading restrictions operating in the market.

$$
\begin{aligned}
& R = \{ \ |T_i| \le W_i^s (1-\eta_i), \forall \mathrm{i} \in \mathrm{S} \ \} \\
& \text{where} \\
& \eta_i = \min(\eta_i^{limit}, r_t), \\
& R = \text{set of trading restrictions} \\
& r_t = \text{the targeted reduction} \\
& W_i^s = \text{baseline emission levels} \\
& \eta_i = \text{control efficiency for discharger } i \\
& \eta_i^{limit} = \text{control efficiency limit imposed by BACT} \\
& \qquad \text{(BACT : Best Available Control Technology)} \\
& S = \text{set of sellers}
\end{aligned}
\tag{3.9}
$$

The set ***R*** does not include any restrictions such as zoning restrictions that may be imposed by decision-makers. However, these can be easily integrated into the existing model by grouping point sources into groups and then implementing a TDP within each group.

This approach simulates the trading market as an instantaneous, multilateral trading process. This approach improves over conventional emissions trading modeling approaches by modeling the individual trades. This information might be necessary to estimate the behavior of the permit cost in real trading market simulations.

Trading Algorithms

The optimization model simulates an ideal multilateral trading market. However, Atkinson and Tietenberg (1991) argue that actual trading is a bilateral process and hence the market outcomes are considerably different as compared to the results of the multilateral trading simulations. The TDP optimization model used in this DSS can be extended easily to such formulations by generating each of the trades t_{ij} in the matrix of individual trades ***T*** in a bilateral and sequential manner.

3.2.5.3 Land Use Planning Optimization

The amount of non-point source runoff that pollutes streams can be managed through appropriate land management plans. The land use plans are designed to meet future land use development requirements. A typical optimization formulation for land-use planning is given below:

Optimization Model Formulation

Decision Variables: The land use acreages in the various land-use management units.

$$
\begin{aligned}
&l_i^k \\
&\forall i = 1,\ldots,N_l^k \\
&\forall k = 1,\ldots,N_{cu} \\
&\text{where} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (3.10) \\
&l_i^k = \text{acreage of land use of type } i \text{ in land use management unit } k \\
&N_l^k = \text{number of land use types in land use management unit } k \\
&N_{cu} = \text{number of land use management units}
\end{aligned}
$$

Objective Function: The objective function is either the weighted sum of the acreages of particular types of land uses or is a function of the water quality indicators. For e.g., if the objective is to maximize a certain type of development, then the objective function can be written as:

$$
\begin{aligned}
&Max: w_1 \sum_{k=1}^{N_{cu}} l_1^k + w_2 \sum_{k=1}^{N_{cu}} l_2^k \\
&\text{where} \\
&l_1 = \text{area of land use of type 1} \qquad\qquad (3.11) \\
&l_2 = \text{area of land use of type 2} \\
&w_1 \text{ and } w_2 \text{ are user defined weights}
\end{aligned}
$$

Constraints: The search is typically constrained by constraints on the land use changes or constraints on the water quality indicators. Land use change constraints specify the bounds within which the areas of a particular land use can change.

Allowable Changes: The typical decision variables for the land use planning model are the final land use plans. In such land use planning models, the fact that the final land use distribution may not be achievable because certain land use changes are not practically feasible is conveniently ignored. An example of such an infeasible change would be the change of urban land to forest land. Therefore, the following approach is used to model the land use planning problem. Within this formulation, the actual decision variables are the pair-wise land use changes. These are represented the matrix that is given as:

$$D_l^k = \begin{vmatrix} 0 & d_{12}^k & . & . \\ 0 & 0 & . & d_{2n}^k \\ . & . & 0 & d_{3n}^k \\ d_{n1}^k & . & . & 0 \end{vmatrix} \quad \forall k = 1,..,N_{cu} \tag{3.12}$$

where

n = number of land use types

d_{ij}^k = percentage of land use of type i to type j

D_l^k = matrix of land use changes

k = land use management unit

An allowable change matrix in Eq. 3.13 indicates the inter-convertibility of land uses.

$$A_l^k = \begin{vmatrix} 0 & b_{12}^k & . & b_{1n}^k \\ 0 & 0 & . & b_{2n}^k \\ . & . & 0 & b_{3n}^k \\ b_{n1}^k & . & . & 0 \end{vmatrix} \quad \forall k = 1,..,N_{cu} \tag{3.13}$$

where

b_{ij} = binary variable

$= 1$ if land use $i \rightarrow$ land use j

$= 0$ otherwise

A_l^k = matrix of allowable land use changes

k = land use management unit

$\rightarrow$ represents a 'can be converted to' relation

In this model formulation, the land use acreages are then the dependent variables that are given by:

$$l_i^k = \sum_{j=1}^{N_l} d_{ij}^k b_{ij}^k \;\; \forall i = 1,...,N_l \; and \; \forall k = 1,...,N_{cu} \tag{3.14}$$

This approach yields the final land use plans as well as the land use inter-conversions that led to it. This represents a more realistic land use planning solution.

<u>Hierarchical Structure</u>: The land use constraints and the land use allowable change matrices can be specified for individual land use management units. For e.g., in a watershed that consists of multiple sub-watersheds, the constraints can be specified for the subwatersheds as well as watershed.

3.2.6 Optimization Procedures

In the past, optimization has been applied to water quality management with varying degrees of success. Researchers have been able to solve water quality management problems for non-reactive and weakly reactive pollutants where reactivity may be neglected or accounted for easily. These problems have been solved by a number of traditional optimization methods, including linear programming, non-linear programming, mixed integer programming and dynamic programming.

The comprehensive modeling of water quality constituents by simulation models such as HSPF significantly complicates the problem. One potential difficulty is the complex nature of the required optimization formulation. Factors that contribute to the complexity include:

- large number of decision variables and constraints needed to represent the sources and controls;
- non-linear cost functions that might be required to reflect the economies of scale of control equipment; and
- processes modeled by the watershed model (HSPF) cannot be represented as a set of mathematical equations.

Traditional optimization procedures impose severe restrictions in using complex nonlinear environmental models such as HSPF within a systematic search. Due to these reasons, conventional LP and NLP solvers are not suitable for optimizing the control strategy formulations. Genetic Algorithms (GAs) are general probabilistic heuristic search procedures for global optimization and have been shown to be powerful in addressing these issues successfully in environmental issues by Loughlin (1998) and Harrell (1998).

Genetic algorithms (GA) are adaptive, global search procedures that are designed to mimic the underlying processes governing natural selection and evolution. A genetic algorithm works on a population of individuals to identify relatively better individuals among the population, and combine the information embedded in the better individuals to form newer and improved individuals. Each individual represents a feasible solution and the evaluation of the solution is based on the objective function value of that solution. Through repeated application of a set of basic GA operators, including selection, mutation and crossover, on the population of the solutions, the procedure converges to a solution that performs best with respect to the evaluation criterion. Although genetic algorithms do not guarantee global optima, in practice they are quite successful in finding good solutions in an efficient manner. Figure 3.1 briefly illustrates the GA optimization methodology.

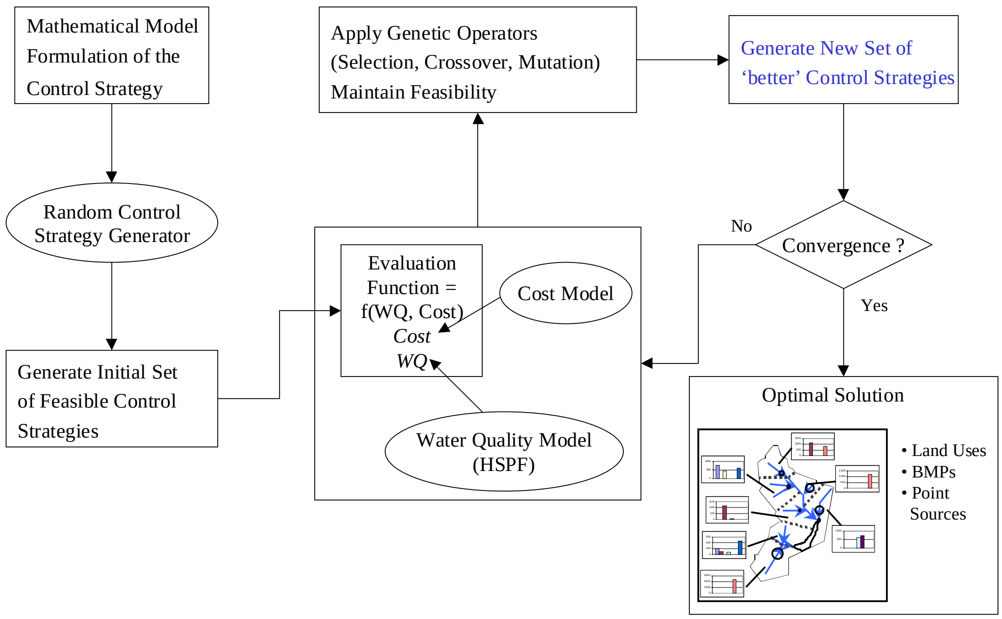


Figure 3.1 Genetic Algorithm-based Search Procedure

3.2.7 Uncertainty Analyses

Most of the models used to assess the environmental impacts of potential control strategies utilize several parameters that require extensive calibration and are highly uncertain. These uncertainties are introduced due to factors such as random and systematic measurement errors and reliance on other models or surrogate indicators. An explicit quantitative analysis of the uncertainty in the achievement of water quality targets is useful in determining whether environmental goals will be met.

3.2.7.1 Sources of Uncertainty

The DSS utilizes HSPF for modeling the watershed water quality. The model requires extensive calibration and utilizes empirical parameters that are highly uncertain. Hence, quantifying the uncertainties in the model outputs that are induced by the uncertainties in the model inputs and comparing the relative contributions to the output uncertainties are very crucial. Considering the imprecise nature of information about the emissions and control processes, the uncertainties in these also needs to be considered while assessing the reliability with which the water quality targets are achieved.

3.2.7.2 Framework for Robust Decision Making

The DSS allows the user to perform uncertainty analysis of the model parameters and thus characterize the uncertainty in achieving the water quality targets. The capability of modeling real world entities such as emissions in a probabilistic manner will be included in extensions to the DSS.

The uncertainty propagation methodology involves the generation of a set of representative samples for each uncertain parameter. A realization of these parameters is then formulated by sampling a single value from a sample set generated for each uncertain input parameter. The model is run for several realizations and output distributions are constructed from the output values for each run.

The distinction in the methods used to propagate uncertainty predominantly lies in the generation of the representative sample set for each of the parameters. Monte Carlo sampling and Latin Hypercube sampling are classical probabilistic sampling techniques that are used as the sampling procedures in the DSS. These techniques are discussed extensively by Morgan and Henrion (1995). A realization is a set of values, sampled from the probability distributions of the uncertain parameters. Several realizations of the uncertain parameters are generated and the model is run in an iterative framework for each of the realizations generated. The output probability distributions are a measure of the uncertainty in achieving the water quality targets.

Although several watershed analysis applications have used HSPF to model watershed water quality, there are very few studies addressing the sensitivity and uncertainty issues in the model parameters and their effect on the simulated water quality in a quantitative manner. A few preliminary sensitivity analysis studies were performed by Fontaine et.al (1997) and Jacomino et.al (1997). It is thought that with the DSS capability to perform the uncertainty analyses in a systematic manner, users would find it convenient to carry out such studies in the future.

3.3 Design of the DSS

The DSS comprises of the control strategy representations, the control strategy formulations, analysis and optimization procedures, the uncertainty analysis procedures and the user interface components. These components of the DSS are integrated into a single system to realize the objectives stated in Section 3.1.3.

3.3.1 Integration of the Watershed Model (HSPF)

The integration of HSPF into the DSS facilitates iterative decision making, the incorporation of the HSPF run results into the optimization model and the estimation of the uncertainties in achieving the water quality targets. This integration is brought about by a set of components that make the necessary modifications to the input file to HSPF, and run HSPF in an iterative framework.

The HSPF simulation requires two types of input files:

1. The user control input (UCI) file is a text file and is the principal input file for a HSPF simulation.
2. Watershed data management (WDM) file is a binary file containing input time series meteorological data such as the precipitation, ambient and dew point temperatures, atmospheric deposition, etc.

The UCI file contains the simulation period, parameters that characterize the hydrological and the contaminant transport processes, linkages between pervious, impervious land segments and the river reach file, specific pollutant loading information and output control information. This file contains the parameters that represent the decision variables for the various control strategies and the modeling parameters. Figure 3.2 shows the structure of the UCI file.

For large watersheds with multiple sub-watersheds having detailed river reach files and a larger number of land units, the UCI file can be significantly large (several MB). Considering the large number of decision variables that define a control strategy, it is very difficult for a user to perform ‘what-if’ analysis by changing these decision variable values in the UCI file either directly or by manipulating BASINS databases. The integration of the watershed model (HSPF) with the DSS facilitates such analysis.

Users Control Input (UCI) File

Global Block
Simulation Period

Files Block
Required Data Files: WDM files, Point Source Input

Sequence of Operation
Perlnd (1- *), Implnd (1-*), Rchres(1- *)

Perlnd (Pervious Land
Perlnd Sub-module activity flags
Perlnd Land Use Data
Perlnd Model Parameters

Implnd (Impervious Land
Implnd Sub-module activity flags
Implnd Land Use Data
Implnd Model Parameters

Rchres (River Reach File
Rchres Sub-module activity flags
River reach characteristics
Rchres Model parameters

EXT Time Series - (External contaminant loading, for e.g. Depositons)

Output Control Information

Network Connectivity
Connectivity of Perlnd segments to Rchres
Connectivity of Implnd segments to Rchres
Rchres connectivity

Figure 3.2 Users Control Input (UCI)

The DSS can be customized to incorporate different watershed models such as the Soil and Water Assessment Tool (SWAT) and the Storm Water Management Model (SWMM).

3.3.2 User Interfaces

The core components of the DSS that consist of the control strategies and uncertainty analyses and optimization algorithms are linked to the following user interfaces:

ArcView Interface

The user interacts with the system through the ArcView GIS interface. Additional utilities are included into the existing BASINS ArcView interface that enable a user to:

- Formulate various point and non-point control strategies
- Specify the uncertain parameters and the probability distributions for these parameters.

The DSS Graphical User Interface

A graphical user interface (GUI) enables users to:

- Edit the control strategy and the uncertainty analysis formulation.
- Change the GA parameters.
- Control the progress of the GA run and the uncertainty propagation procedure.

3.3.3 Integrating the DSS with BASINS

Figure 3.3 illustrates the DSS components and the coupling of the DSS with BASINS. The components of the DSS interact with the BASINS interface through a text based files. The arrows in Figure 3.3 illustrate a typical control strategy optimization or an uncertainty analysis run. The numbering of the arrow heads represents the sequence in which these tasks are carried out. The user enters the problem formulation through the ArcView interface (1). This formulation is then written to problem files (2, 3). The control strategy and the uncertainty analysis procedures are formulated (4, 5a, 5b). The genetic algorithm is used to optimize the control strategy (6a) or the uncertainty propagator is used to propagate the uncertainty (6b). The GA evaluation function and the uncertainty propagator then update the UCI file to reflect the decision variables (7a) and uncertain parameter values (7b) and run the watershed model HSPF with the updated UCI file. The results of the optimization procedure and the uncertainty analysis run are then returned for display to the ArcView interface (8,9,10).

Reliability based optimization can be performed for identifying cost effective control strategies that achieve the water quality targets with specified reliability. This is done by running HSPF for a specified number of realizations of the uncertain parameters and using this probabilistic information in the GA evaluation function.

This integration enables users that are familiar with BASINS, but are not aware of detailed control strategy formulations, optimization and uncertainty analyses algorithms to use these systems analytic tools in the identification of cost effective and robust control strategies.

The structure and functions of these components form the requirements that are the basis of a software development approach that is followed for better DSS design and development. The following chapter discusses the design and development of the DSS.

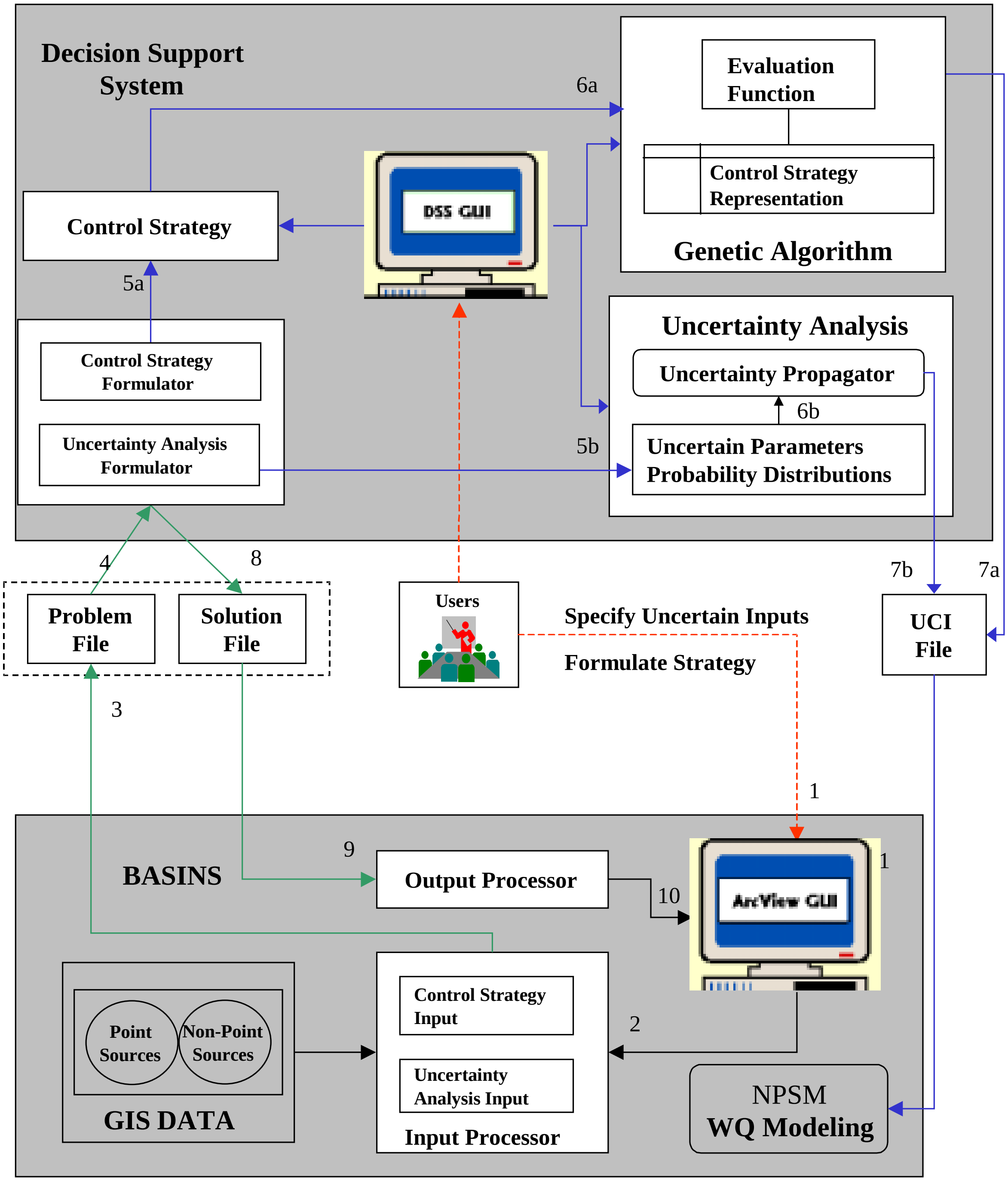


Figure 3.3 DSS Components and Interactions

References

1. Atkinson, S., and Tietenberg, T., (1991). *Market Failure in Incentive-Based Regulation: The Case of Emissions Trading*, Journal of Environmental Economics and Management, 21, pp. 17-31.
2. Barnwell, T. (1984), *HSPF – Hydrological Simulation Program Fortran, Users Manual*.
3. Brill, E.D., (1997). *Effluent Charges and Transferable Discharge Permits*, in Design and Operation of Civil and Environmental Engineering Systems, Ed. Charles Revelle, Wiley, NY, pp.657.
4. Brill, E.D, Eheart, J.W., Kshirsagar, S.R., and Lence, B.J., (1984). *Water quality impacts of biochemical oxygen demand under transferable discharge permit programs*, Water Resources Research, Vol.20, No.4, pp. 445-455.
5. Chang, N., Wen, C.G., Wu, S.L., (1995). *Optimal Management of Environmental and Land Resources in a Reservoir Watershed by Multiobjective Programming*, Journal of Environmental Management, 44, pp. 145-161.
6. Eheart, J.W., Brill, E.D., and Liebman, M.C., (1990). *Discharger Grouping for Water Quality Control*, Journal of Water Resources Planning and Management, Vol. 116, No. 1, pp. 21-37.
7. EPA Document, (1999), *BASINS – Better Assessment Science Integrating Point and Non-Point Sources, Users Manual*.
8. Fontaine, T.A., and Jacomino, V.M., (1997). *Sensitivity Analysis of Simulated Contaminated Sediment Transport*, Journal of the American Water Resources Association, Vol. 33, No. 2, pp. 313-326.
9. Gillon, D.L., (1999), *Modeling and Analysis of* NO_X *Emissions Trading to Achieve Ozone Standards*. Master's Thesis, Dept. of Civil and Environmental Engineering, North Carolina State University.
10. Harrell, L.J., (1998). *Methods for Generating Alternatives to Manage Water Quality in Watersheds*, Ph.D. Dissertation, Dept. of Civil and Environmental Engineering, North Carolina State University.
11. Harrell, L.J., and Ranjithan, S.R., (1997). *Generating Efficient Watershed Management Strategies Using a Genetic Algorithm-based Method*, Proceedings of the

ASCE 24th Annual Water Resources Planning and Management Conference, April 1997, pp. 272-277.

12. Jacomino, V.M., and Fields, D.E., (1997). *A Critical Approach To The Calibration of a Watershed Model*, Journal of the American Water Resources Association, Vol. 33, No. 1, pp. 143-154.
13. Line, D.E., Osmond, D.L., Gannon, R.W., Coffey, S.W., Jennings, G.D., Gale, J.A., Spooner, J., (1996). *Nonpoint sources*. Water Environment Research, Vol. 68, No. 4, pp. 720-728.
14. Loughlin, D.H, (1998). *Genetic Algorithm-Based Optimization in the Development of Tropospheric Ozone Control Strategies*, Ph.D. Dissertation, Civil Engineering, North Carolina State University.
15. Loughlin, D.H, Brill, E.D, Ranjithan, S., Baugh, J.W, Neal, K.N., Fine, S.S. and Vukovich, J.M., (1997). *Simulation of air quality management pollutant trading programs using mathematical optimization*, in Proceedings of Emissions Inventory, A&WMA Specialty Conference, RTP, NC.
16. Loughlin, D.H., Neal, J.K., Ranjithan, S., Brill, E.D., Baugh, J.W., (1995). *A Decision Support System for Air Quality Management*, Proceedings of the 2nd Congress on Computing in Civil Engineering, ASCE, Atlanta, GA.
17. Morgan, G.M., and Henrion, M., (1995). *Uncertainty: A Guide to Dealing with Uncertainty in Quantitative Risk and Policy Analysis*, Cambridge University Press.
18. Ribeiro, C.T., (1996). *Impact of land use on water resources, integrating HSPF and a raster-vector GIS*, in HydroGIS 96: Applications of Geographic Information Systems in Hydrology and Water Resources Management, Proceedings of the Vienna Conference, IAHS Publ. No. 235.
19. Tecle, A., (1992). *Selecting a multi-criterion decision making technique for watershed resources management*, Water Resources Bulletin, Vol. 28, No. 1, pp.129-140.
20. Tietenberg, T.H., (1985). *Emissions Trading: An Exercise in Reforming Pollution Policy*, Resources for the Future, Inc., Washington, D.C.

4 DSS: Design and Implementation

4.1 Introduction to a Software Development Approach

This decision support framework integrates environmental simulation models with systems-analytic tools that aid a decision-maker in the formulation of cost efficient as well as reliable environmental management strategies.

This framework enhances the environmental analysis and management capabilities of existing environmental analysis frameworks by including an automated search for feasible as well as optimal environmental control strategies, estimation of uncertainty in achieving environmental targets and comparison of the economic implications of various control strategies.

The development of this framework involves the design of software modules for analyzing the scenarios resulting from the application of environmental control strategies, various optimization algorithms that search for efficient control strategies, and uncertainty analysis algorithms. These conceptually different tools need to be integrated under a single platform in order to bring the decision support capabilities to the desk-top.

To maintain the modularity of the various components, achieve better system design and promote code reusability, the software design of this framework is based object oriented paradigm and design principles. Development of a model for the software system prior to its construction is as essential as having a blueprint for a large building.

The design of this decision support system is based on the Unified Modeling Language (UML) methodology. The implementation of this system is carried out using the Java^{TM} platform. The Java programming language environment provides a portable, interpreted, high-performance and simple, object-oriented programming language. This makes the design and implementation of the system simple, highly modular and promotes code reusability.

The following sections describe the development of the DSS based on current practices for software engineering such as object oriented analysis (OOA) and object oriented design (OOD).

4.2 Object Oriented Modeling and Design

Developing a flexible, generic and modular system that is capable of representing the environmental control strategy data structures, optimization and uncertainty analysis procedures calls for better modeling and design approaches to build the software system. The design of the system is based on object oriented (OO) paradigm. OO modeling and design promote better understanding of requirements, cleaner designs and more maintainable systems. OO technology is a way of thinking abstractedly about a problem using real world concepts, rather than computer concepts. The themes underlying OO technology are briefly discussed below:

- **Abstraction** consists of focusing on the inherent aspects of an object and its functionality. This avoids design and implementation decisions being made before the problem is understood.
- **Encapsulation** consists of separating the external aspects of an object that are accessible to other objects from the internal aspects of the object. This prevents small changes in object structure from having massive ripple effects that affect the entire *class* hierarchy.
- **Inheritance** of the data structure and behavior allows common structure to be shared among various *classes*.

4.2.1 The Object Model

The object model captures the static structure of a system by showing the objects in a system - their identity and their relationships to other objects. An *object* is defined as a concept or an abstraction. It captures concepts from the real world that are important to the application. The following terms define the various components that collectively represent an object model:

<u>Class</u>

A *class* is a static abstraction of a set of real world entities that have the same characteristics and share the same behavior. The principal features of a *class* are its *attributes*, *operations* and its relationship to other *classes*.

<u>Attribute</u>

An *attribute* is a data value held by *objects* of a *class*.

Operation

An *operation* is any function that may be performed by the objects.

Association

An *association* is a semantic relationship that exists between two *classes*. Each association has two roles; each role is a direction on the association. The role also has a multiplicity that indicates the number of objects participating in the given relationship. There are three kinds of associations: aggregation relationship, inheritance relationship and association relationship.

Aggregation Relationship

This is an *association* based on the 'whole/part' concept. There are two types of aggregation relationships:

- *Aggregation* is a relationship where the 'whole' *class* does not have to create its 'part' *class,* but refers to the 'part' *classes* by reference.
- *Composition* is a relationship where the 'whole' part is responsible for creating its 'part' *classes* directly. This is a tighter coupling between the 'whole' and 'part' *classes* than is indicated by *aggregation*.

Inheritance Relationship

This is an *association* between *classes* that focuses on similarities and dissimilarities between the *classes* with respect to the *classes*' *attributes* and *methods*. An *inheritance relationship* exists between a *superclass* and a *subclass*. A *subclass* is a *class* dependent from its *superclass* and inherits all the *attributes* and *methods* of its parent *superclass*.

Association Relationship

An *association relationship* defines the nature of the coupling between the two *classes*. In this relationship, the interacting parts are visible to each other and may be shared between different aggregation hierarchies.

Aggregation Hierarchy

Is defined as a set of *classes* related through an aggregation relationship, with one root *class*. All other *classes* are parts of and help comprise the hierarchy.

Inheritance hierarchy

Is defined as a set of *classes* related through inheritance relationships with one root *class*. All other *classes* in the hierarchy are *subclasses* of the root class.

4.2.2 Object Model Representation

The *object model* is represented graphically with diagrams containing *classes* or *packages*. *Classes* are arranged into hierarchies sharing common structure and behavior. A *package* is a collection of *classes* that are logically related and represent collectively a real-world or algorithmic system.

Package Diagram

A *package diagram* acts as a logical road map of the software system and helps one understand the logical pieces of the system and see the *dependencies*. A *package diagram* shows *packages* of *classes* and the *dependencies* among them. A *dependency* exists between two elements if changes to the definition of one element may cause changes to the other.

Class Diagram

A *class diagram* describes the types of objects in the system and the various kinds of static relationships that exist among them. *Class diagrams* also show the *attributes* and *operations* of a *class*.

4.2.3 Visual Modeling – The Unified Modeling Language

The Unified Modeling Language (UML) is a standard graphical language for specifying, constructing, visualizing, and documenting software-intensive systems. It covers concepts such as system processes and functions, as well as concrete things, such as programming language *classes*, database schemas, and reusable software components. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML provides:

- a ready-to-use, expressive visual modeling language that can be used to develop and exchange meaningful models;
- extensibility and specialization mechanisms to extend the core concepts;
- a system representation that is independent of particular programming languages and development processes; and
- a formal basis for understanding the modeling language.

4.3 A Software Development Approach

The development of the DSS is based on current practices for software engineering such as requirements engineering, object oriented analysis (OOA) and object oriented design (OOD). This approach integrates front-end activities of requirements engineering based on *use cases* with OOA, OOD and implementation. The basis for this approach is the *use case* introduced by Jacobsen (1994). The software engineering approach illustrated in Figure 4.1 is adopted for the development of the DSS.

The Model View Controller (MVC) architecture is used in the development. MVC architecture consists of three types of *classes*. The model consists of the DSS *domain classes* that represent the control strategies, optimization and uncertainty analysis algorithms. The view consists of the DSS *view classes* that display the information content of the *domain classes* through a combination of graphics and text. The controller consists of the *controller classes* that define how user interaction is handled by the DSS. The controller interprets inputs from the user and maps these user actions into commands that are sent to the model and/or the view to effect the appropriate change. The MVC architecture de-couples these functional components, allowing for easier reuse of code than in a traditional user interface.

Sections 4.4 through 4.8 describe the tasks in each phase of the development approach. Section 4.9 shows the class diagrams that form the basis of the DSS structure. Section 4.10 describes the components of the DSS and their interactions.

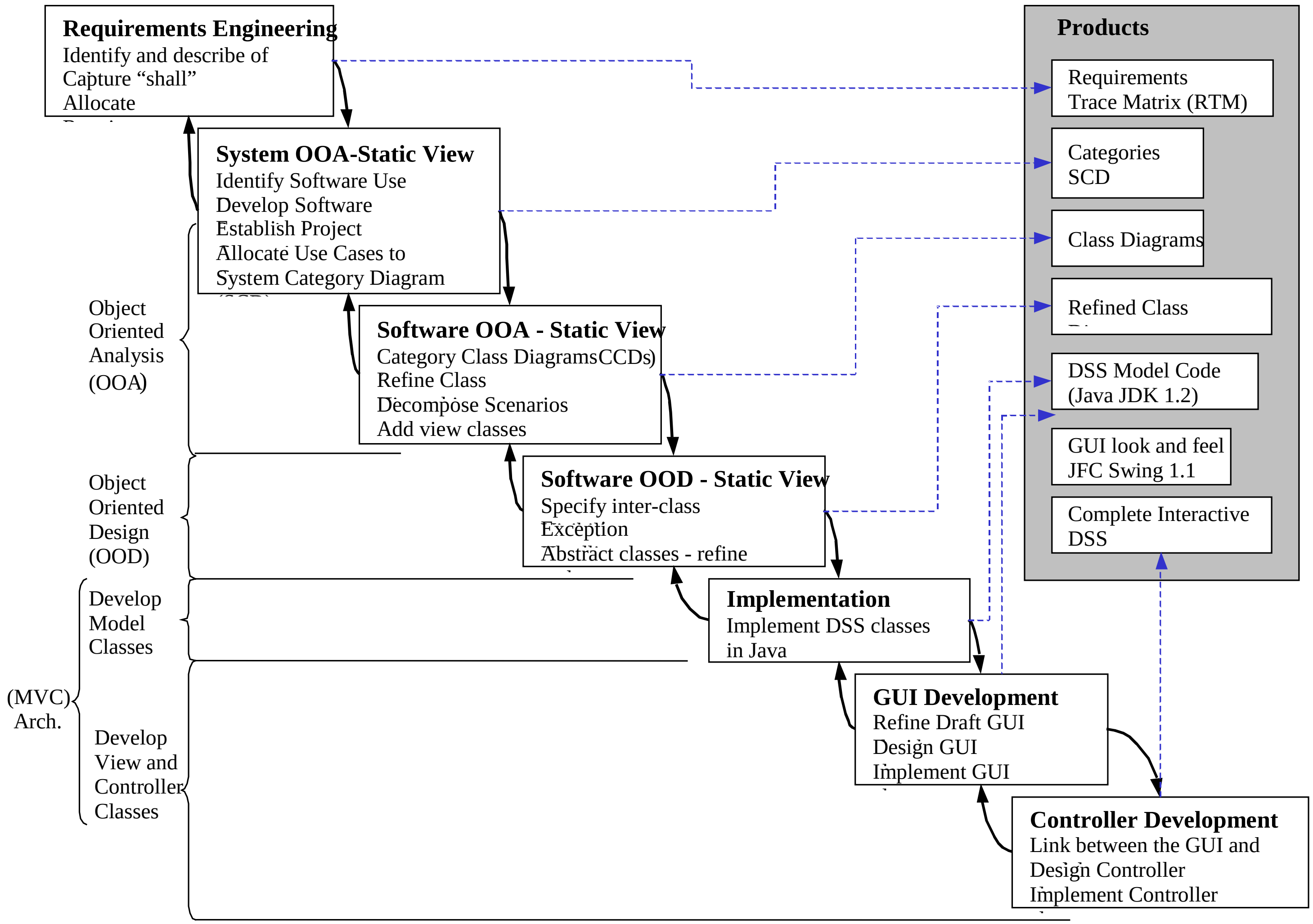


Figure 4.1 Software Development Approach for the DSS

4.4 Requirements Engineering

Requirements engineering revolves around the identification and description of the various tasks that the software system is designed to perform. It is defined as the process of establishing a set of software requirements for the system development process. The process begins with the explicit representation of requirements as a table called the 'Requirements Trace Matrix (RTM)' that maintains all the requirements for the system as individual entries. Each software functional requirement is expressed in a format that clearly defines the actions performed by the software system. A functional requirement expressed in this manner is called the *use case*. This phase consists of the following activities.

4.4.1 Capture 'shall' Statements

Each functional requirement is expressed in a statement that includes the word 'shall'. This word emphasizes that the software system 'shall' fulfill the expressed requirement. These requirements then become the cornerstone of all future development activity.

4.4.2 Allocate Requirements

The responsibilities of the software system are defined clearly. Each entry in the RTM is categorized according to its 'type'. The following are examples of various 'types' of requirements:

- Performance Requirement (P) is a task that the software system is required to perform.
- Software Requirement (SW) is a task that shall be executed by the software system.
- Software Constraint (SWC) is a restriction potentially placed on the software implementation.
- Nice to Have (NTH) is a task that will be good feature to have in the software system, but which is not critical.

Table 4.1 shows some of the *use cases* in the RTM for the DSS.

Table 4.1 Requirements Trace Matrix (RTM)

Requirements Trace Matrix

Entry #	Problem Statement	Type	Build	Use Case Name	Category	Implement
	WQMDSS Requirements					
2	The WQMM shall consist of three modules: 1) Point Source Control Module (PSCM) 2) Non-Point Source Control Module (NPSCM) 3) Uncertainty Analyses Module (UCAM)	P	B1	N/A		
3	The Point Source Control Module (PSCM) shall consist of modules for the following point source control strategies: 1)Effluent Charge Programs Module (ECPM) 2)Trading Permit Programs Module (TDPM) 3)Command and Control Module (CACM) 4)Cost Optimization Module (COM)	P	B1	N/A		
4.1	The NonPoint Source Control Module (NPSCM) shall consist of modules for the following management strategies: 1)Land Use Planning (LUPM)	P	B1	N/A		
5	UCAM shall enable the propagation of uncertainty through the various models.	P	B1	N/A		
	Feature Selection Requirements					
6.3	The user shall be able to group the point sources into various categories. There shall be a default classification scheme	SW	B1	UC6.3_User_Groups_Point_Sources		Avenue/Java
	Problem Formulation Requirements					
13	The user shall be able to formulate his/her own optimization model formulation.	SW	B2	UC13_User_Formulates_Optimization_Model		Avenue/Java
13.1	The user shall be able to formulate the Cost Optimization model for point sources.	SW	B2	UC13.1_User_Formulates_Cost_Optimization_Model		Avenue/Java
13.2	The user shall be able to formulate the Trading Permit Program (TDP) model for point sources.	SW	B2	UC13.2_User_Formulates_TDP_Model		Avenue/Java
13.3	The user shall be able to formulate the Land Use (LU) Planning Optimization model for selected watersheds and subwatersheds.	SW	B2	UC13.3_User_Formulates_LU_Optimization_Model		Avenue/Java
13.6	The user shall be able to formulate the Water Quality(WQ) Optimization model (WQOM)	SW	B2	UC13.6_User_Formulates_WQ_Optimization_Model		Avenue/Java

Table 4.1 continued

Entry #	Problem Statement	Type	Build	Use Case Name	Category	Implement
14	The user shall be able to formulate his/her analysis model formulation.	SW	B2	UC14_User_Formulates_Analysis_Model		Avenue/Java
14.1	The user shall be able to formulate the Command and Control analysis model for point sources.	SW	B2	UC14.1_User_Formulates_Command_And_Control_Analysis_Model		Avenue/Java
14.2	The user shall be able to formulate the Effluent Charge Program analysis model for point sources.	SW	B2	UC14.2_User_Formulates_Effluent_Charges_Analysis_Model		Avenue/Java
14.3	The user shall be able to formulate the Land Use Planning analysis model.	SW	B2	UC14.3_User_Formulates_LandUse_Analysis_Model		Avenue/Java
19	The optimization model formulation: decision variable names, objective function, constraints etc. shall be mapped to the internal problem representation that is used by the optimization / uncertainty analysis algorithms.	SWC	B2	SWC19_Users_Optimization_Model_Maps_To_Internal_Optimization_Problem_Representation		Java
	Input / Output Requirements					
23.1	The user shall be able to save a particular problem formulation to a file or a set of files.B124	SW	B1	UC23.1_User_Saves_WQM_Problem_Formulation_To_File		Java
	Optimization Model Requirements					
27	A Genetic Algorithm shall be used as the Optimization Algorithm	P	B1	N/A	Genetic_Algorithm	Java
28	The Genetic Algorithm design shall be fairly general such that it can model various entities such as point-sources, land uses etc.	SWC	B1	N/A	Genetic_Algorithm_CAT	Java
29	The objective function and constraints for the Genetic Algorithm Optimization run shall be dynamically set based on the user's problem formulation	SW	B1	UC29_OptModel_Controller_Dynamically_Sets_Optimization_Model_For_GA_Evaluator	Genetic_Algorithm_CAT	Java
30.1	The UCI Controller shall make incremental changes / additions to the user control input (*.UCI) file to HSPF based on the instructions given to it by the Genetic Algorithm.	SW	B1	UC30.1_UCI_Controller_Changes_UCI_File	WQModel_CAT	Java
31	The Water Quality Model Runner shall run the HSPF model (xnpsm11x.exe) .	SW	B1	UC31_WQModel_Runner_Runs_HSPF_Model	WQModel_CAT	Java
32	HSPF output parser shall parse the HSPF output and return appropriate values to the Evaluation Function	SW	B1	UC32_ HSPF_Output_Parser_Parses_HSPF_Outp	WQModel_CAT	Java
	Uncertainty Analysis Procedure Requirements					
34.1	The Uncertainty shall be propagated through the models by Latin Hypercube Sampling or Monte Carlo Simulation.	P	B1	N/A	Uncertainty_CAT	Java
34.2	The user shall be able to represent any quantity in the model as a uncertain parameter i.e. model it as a probability distribution	SWC	B1	N/A	Uncertainty_CAT	Java

Table 4.1 continued

Entry #	Problem Statement	Type	Build	Use Case Name	Category	Implement
35	The user shall be able to select the uncertain parameters from a list of Parameters	SW	B2	UC35_User_Selects_Uncertain_Parameters	Uncertainty_CAT	Avenue/Java
39	The user shall be able to select the type of Sampling Procedure: 1) Monte Carlo 2) Latin Hypercube Sampling	SW	B1	UC39_User_Selects_Sampling_Procedure	Uncertainty_CAT	Java
40	The UCAM shall propagate the uncertainty through water quality model for a finite set of realizations	SW	B1	UC40_UCAM_Propagates_Uncertainty_Through_WQModel	Uncertainty_CAT	Java
	Optimization Model Formulation Requirements					
52	An optimization model shall consist of an objective function and a set of constraints. The set of constraints shall consist of constraints.	SWC	B1	N/A	Optimization_Algorithm_CAT	Java
53	The constraints shall be handled by the penalty function approach. The user shall have the choice of selecting types of penalty functions.	SW	B1	User_Selects_Penalty_Functions	Optimization_Algorithm_CAT	Java
54	The objective function or the lhs functions of the constraints shall consist of any 'function'. A function shall be any mathematical function of various objects.	SWC	B1	N/A	Optimization_Algorithm_CAT	Java
	Point Source Management					
58	The user shall be able to select controls that are applied on the emissions.	SW	B1	UC58_User_Selects_Emission_Controls	PS_Strategy_CAT	Java
59	The controls shall be of two types:Discrete and Continuous	SWC	B1	N/A	PS_Strategy_CAT	Java
60	Continuous Control implies that the control efficiency shall be a real continuous variable. The cost for this control shall be represented in a cost curve type format	SWC	B1	N/A	PS_Strategy_CAT	Java
61	Discrete control shall consists of a number of control technologies. Each control technology shall have an efficiency and an associated cost.	SWC	B1	N/A	PS_Strategy_CAT	Java
62	A point source shall have a list of emissions	SWC	B1	N/A	PS_Strategy_CAT	Java
64	The objective function and the lhs functions of the constraints can be an Emission Function or a Cost Function	SWC	B1	N/A	PS_Strategy_CAT	Java
	Land Use Planning					
67	A subwatershed shall be the basic unit for land use planning.	SWC	B1	N/A	LandUse_Planning_CAT	Java

Table 4.1 continued

Entry #	Problem Statement	Type	Build	Use Case Name	Category	Implement
68	The levels of land use planning shall be regional, watershed level and subwatershed level.	SWC	B1	N/A	LandUse_Planning_CAT	Java
69	Land use Constraints shall specify the upper limit on any land use type change in an unit	SWC	B1	N/A	LandUse_Planning_CAT	Java
70	Allowable Change Matrices shall specify whether one type of a land use can be converted to another type of land use.	SWC	B1	N/A	LandUse_Planning_CAT	Java
71	The land use constraints and allowable change matrices can be specified at all the levels of land use planning.	SWC	B1	N/A	LandUse_Planning_CAT	Java
73	The land use planning optimization model shall consist of an objective function and/or constraints on land use change.	SWC	B1	N/A	LandUse_Planning_CAT	Java
	Water Quality Management					
74	The water quality optimization model shall consist of an objective function and/or water quality constraints	SWC	B1	N/A	WQ_Model_CAT	Java
75	The functions shall typically consist of math functions of water quality constituents.	SWC	B1	N/A	WQ_Model_CAT	Java

	Modules
WQMM	Water Quality Management Module
PSCM	Point Source Control Module
NPSCM	Non-Point Source Control Module
UCAM	Uncertainty Analysis Module
ECPM	Effluent Charge Programs Module
TDPM	Trading Permit Programs Module
CACM	Command and Control Module
COM	Cost Optimization Module
LUPM	Land Use Planning Module
	Abbreviations
UC	Uncertainty Analysis
HSPF	Hydrologic Simulation Program in Fortran
UCI	User Control Input - The Input File to HSPF
	Programming Languages
Avenue	The programming environment for ArcView GIS
Java	JavaTM (Java Development Kit 1.2)

	Type
SW	Software Requirement
SWC	Software Constraint
P	Performance Requirement
NTH	Nice To Have

	Build	**Status**
B1	Core model and data structures	Implemented
B2	User Interface	Ongoing

4.5 Systems Object Oriented Analysis (OOA) – Static View

The RTM developed in the requirements engineering phase contains the set of software specifications. This phase begins with the reformatting the set of software requirements in the RTM into a *use case* format.

4.5.1 Identification of Software Use Cases

A *use case* is a typical interaction between a user and a system. *Use cases* are an essential tool in requirements capture and in planning. The software requirements in the RTM are reformatted to aid the discovery of potential *categories*. The *use case* is a statement of software functionality written in the format:

Actor *Action* *Subject*

where, *Action* represents the capability of the software, *Actor* represents a stimulus to the software system, and, *Subject* represents the entity whose data structure is accessed by the action. The *use cases* in the RTM enable the development of *categories* and a thorough requirements trace.

4.5.2 Development of Scenarios

A *scenario* provides the operational concept behind a *use case*. It is a description of software actions that are required for the completion of a *use case*. The scenario represents an operational concept and is hence independent of the implementation. The following is an example of a scenario for *use case* # 40.

This scenario illustrates the software actions that are required for the completion of *use case* # 40. In this *use case*, the uncertainty in the model parameters is propagated through the water quality model in response to the users stimulus. The 'action' is the users stimulus and the 'software reaction' is the set of tasks carried out. The task numbers indicate the order in which the tasks are carried out. The 'pre-conditions' and 'post-conditions' specify the states of the software system prior to and after the 'software reaction' occurs. The 'exceptions' specify any error conditions that may occur during the execution of the 'software reaction'. The '*use cases* utilized' specify any other *use cases* that may be used in the execution of the 'software reaction'.

Use Case # 40: UC40_UCAM_Propagates_Uncertainty_Through_WQModel

Overview:
This use case enables the UCAM to propagate the uncertainty through water quality model for a finite set of realizations

Preconditions:
1. The DSS Java GUI is displayed.
2. The Uncertainty menu is active.

Scenario:

#	Action	Software Reaction
1.	User clicks "Propagate Uncertainty" in the Uncertainty menu in the DSS Java GUI.	1. UCAM requests a realization from the sampling procedure. 2. The Sampling procedure formulates a realization and returns it to UCAM. 3. UCAM instructs the UCI controller to modify the UCI file based on the realization and the selected parameters. 4. The UCI controller modifies the UCI file. 5. UC #30.3 6. UC #31. 7. Steps 1 through 6 are repeated for the number of realizations input by the user. 8. UC #41.

Scenario Notes: None

Post Conditions: Uncertainty in the selected uncertain parameters is propagated through the water quality model.

Exceptions:
1. IOException
2. Output Parser Exception.

Required GUI: DSS Java GUI.

Use Cases Utilized: UC #30. 3, #31, #41

Timing Constraints: May be done only after UC #39.

Primary Implementation: Java.

Figure 4.2 Sample Scenario

4.5.3 Establishment of Project Categories

A *category* is a collection of logically related *classes*, related through *inheritance* and *aggregation*. *Categories* provide the kernel for all future development efforts and lay the foundation for the entire project structure. The following tasks comprise the process of extraction of project *categories*:

- Development of candidate *category* list. Actors and subjects from the *use cases* form the basis for a candidate list of *categories*.
- Extraction of a *category* as the root of an *inheritance hierarchy*. The candidate *category* list is examined for similar features that can potentially lead to the development of an *inheritance hierarchy*. The *category* that is the root of this hierarchy is retained as the project *category*.
- Extraction of a *category* as the root of an *aggregation hierarchy*. The candidate *category* list is examined for whole-part relations between *categories* that can potentially lead to the development of an *aggregation* hierarchy. The 'whole' *class* is retained as the project *category*.

4.5.4 Allocation of Use Cases to Categories

The *use cases* are allocated to the project *categories* discovered in the previous task. This allocation clarifies the responsibility for the development of a specific *use case*. This is depicted by the '*Use case*' and '*Category*' columns in Table 4.1.

4.5.5 Development of System Category Diagram

The *system category diagram* (SCD) depicts the high-level associations that exist between the *categories*. Each *category* on the SCD is a *package*. Figure 4.3 shows the SCD for the DSS. The association between the *categories* is one of *dependency*. A *dependency* implies visibility requirements in the subsequent design. For example, in the SCD illustrated by Figure 4.3, the *category* 'water quality management' depends on the *categories* 'point source control strategy' and 'non-point control strategy'. The SCD explains the overall view of the system and validates the *categories*.

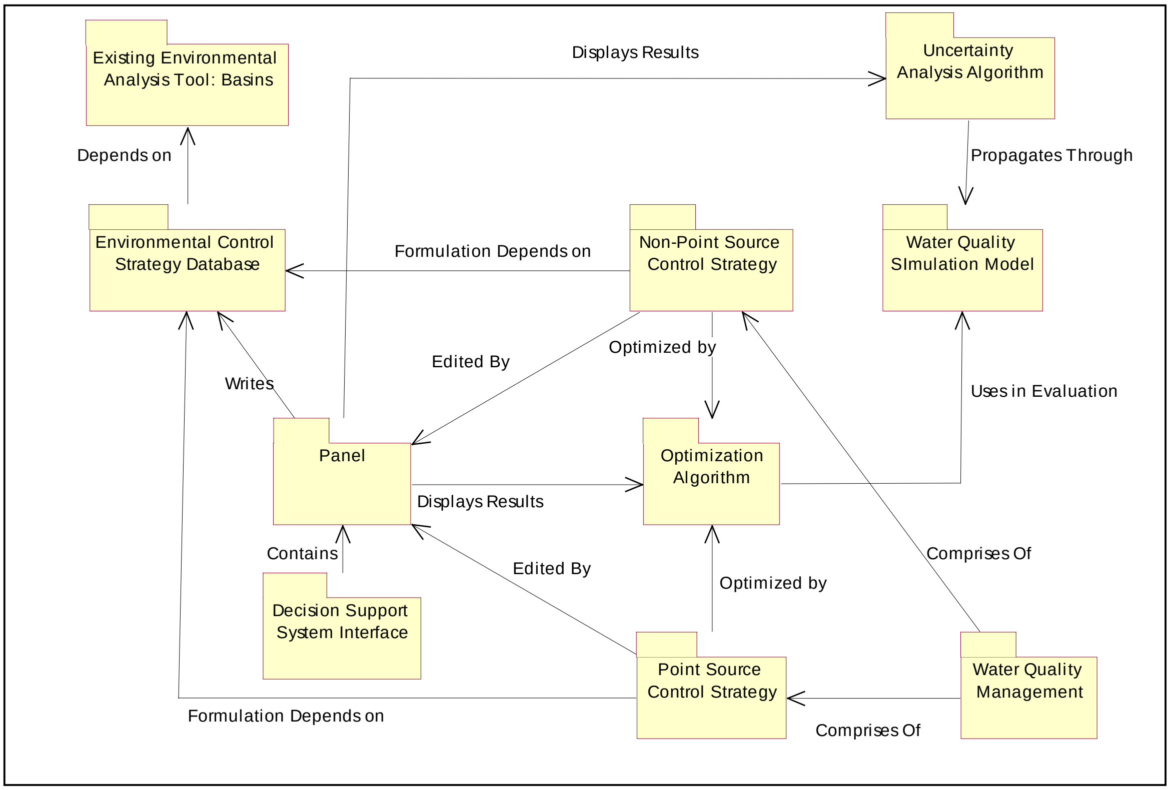


Figure 4.3 System Category Diagram

4.6 Systems Object Oriented Analysis – Dynamic View

The previous phase of design focused on the identification of the major components of the software system, *categories* and the *associations* between them. This phase focuses on the dynamic view of the system i.e. the interaction between the *categories* that is necessary for the implementation of a *use case*.

4.6.1 Development of Interaction Diagrams

An *interaction diagram* (ID) defines the functional behavior of the system. It depicts the collaboration of *classes* for the implementation of a *use case*. It determines which *class* has the responsibility for implementing a specific portion of functionality for a *use case*. Figure 4.4 shows the ID for *use case* # 40. A rectangle represents an instance of a *class*. The horizontal arrows represent the sequential order in which messages are passed among the *classes*.

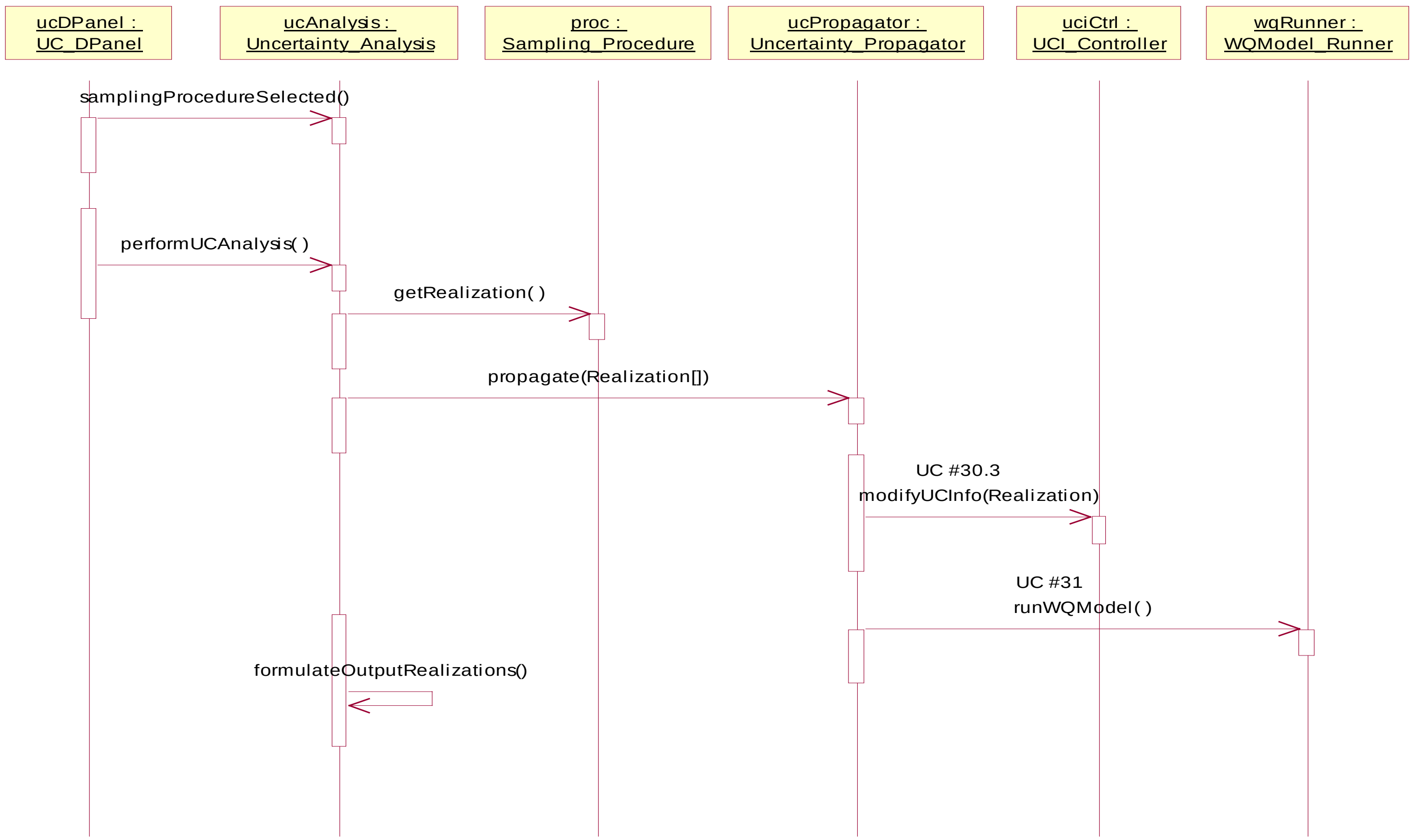


Figure 4.4 Interaction diagram for UC # 40

4.7 Software Object Oriented Analysis – Static View

This phase involves the design of the various *classes*, identification of their *attributes* and *methods*, and the development of the *associations* between the *classes*. For each *category*, a *category class diagram* is produced.

4.7.1 Initiation of the Category Class Diagram

A *category class diagram* (CCD) is a *class diagram* that depicts all the *classes* owned by the *category*, all *classes* required that are owned by other *categories* and all the *associations* between these *classes*. There is one CCD per *category*. A CCD may also depict *child categories*. A *child category* is a *category* owned completely by another *category* and represents a subset of logical collection of *classes* of the parent *category*.

4.7.2 Refinement of Inheritance and Aggregation hierarchies

The *inheritance* and *aggregation hierarchies* for each *category* are developed and refined. The *attributes* and *operations* of the root *class* are identified. If these are applicable to other similar *classes*, this points to the existence of an *inheritance hierarchy*. The identification of the *associations* that exist between the part and the whole *classes* leads to the discovery of potential *aggregation hierarchies*. This validates the envisioned hierarchies.

4.7.3 Decomposition and Analysis of Scenarios

The *scenarios* developed in Section 4.5.2 are analyzed to verify whether the *scenario* implementation supports the particular *use case*. During this process, *classes* required for the *scenario* implementation and appropriate *attributes* and *operations* for these *classes* are identified. Nouns in the scenario are represented as *attributes* or *classes*, verbs are represented as *operations* and adjectives indicate the possibility of the existence of *inheritance hierarchies*. The newly discovered *classes, attributes* and *operations* and the *class associations* are added to the *category class diagrams*.

4.7.4 Addition of View *Classes*

A *view class* is a *class* that defines the manner in which information in a *domain class* is displayed by the system. It may additionally provide controls for user interaction. A *domain class* is a *class* that defines a real world entity from the problem definition. Graphical User Interface (GUI) or *view classes* are added to the current object model that is represented by the *category class diagrams*. The *association relationships* that relate the *view classes* to the *domain classes* are also added.

4.8 Software Object Oriented Design – Static View

The static logical view of the software system developed during the preceding phases is transitioned to an implementation view. The *category class diagrams* (CCDs) are modified to include additional *collection classes* and *abstract classes*. Access rights for *attributes* and *operations* are specified. *Exceptions* that need to be thrown by particular operations are identified. This activity consists of the following tasks.

4.8.1 Inter-Class Visibility

An *association relationship* does not explicitly identify the *class* that provides the services and the *class* that uses the services. The *association relationships* are changed to uni-directional associations that explicitly specify the *client* and *supplier classes*. A *client class* is a *class* that depends on the *supplier class* for some functionality. A *client class* contains a reference to its *supplier class* in the form of an *attribute*. The uni-directional association relationship implies that the *client class* needs visibility to the *supplier class*.

The visibility is characterized by the public, protected and private access identifiers. A public access allows all *client classes* visibility to a *class's* public *attributes* and *operations*. A protected access permits only *subclasses* to access their *superclasses* protected *attributes* and *operations*. A private access right prohibits access by external *classes* to a *class's* private *attributes* and *operations*. These identifiers control the hiding and visibility of data and operations. It is a good OO practice to not have any public *attributes*.

4.8.2 Exception Handling

An *exception* is an error condition that occurs during run-time and that without associated corrective processing, results in system failure. Those *operations* of particular *classes* that might encounter error conditions are identified. These *operations* are then documented by declaring the *operation* to throw the particular *exception*. *Exception* handling results in fewer runtime problems and error conditions and is an efficient way of handling errant program flow conditions.

4.8.3 Abstract *Classes*

An *abstract class* is the *superclass* of an *inheritance hierarchy* that does not have any instances. *Abstract classes* organize features common to several *classes*. It is useful to create abstract *superclass* to encapsulate *classes* that participate in the same associations. Some *abstract classes* are deliberately introduced as a mechanism to promote code reuse. Programming with *abstract classes* and *interfaces* make the software system more flexible, scalable and promotes code reuse.

4.9 Class Diagrams for the DSS

The software development process illustrated in Sections 4.4 to 4.8 was carried out in an incremental manner. The results of this process are the *class diagrams* for each of the individual project *categories* of the DSS. Figure 4.5 through Figure 4.18 show the *class diagrams* for of the DSS and their child-*categories*. The UML notations and conventions are explained in Appendix B. The *class diagrams* are discussed in Section 4.10.

Only principal *classes* in a *category* and the their *associations* are depicted on the *class diagrams*. The important *operations* of these *classes* are shown on the CCDs.

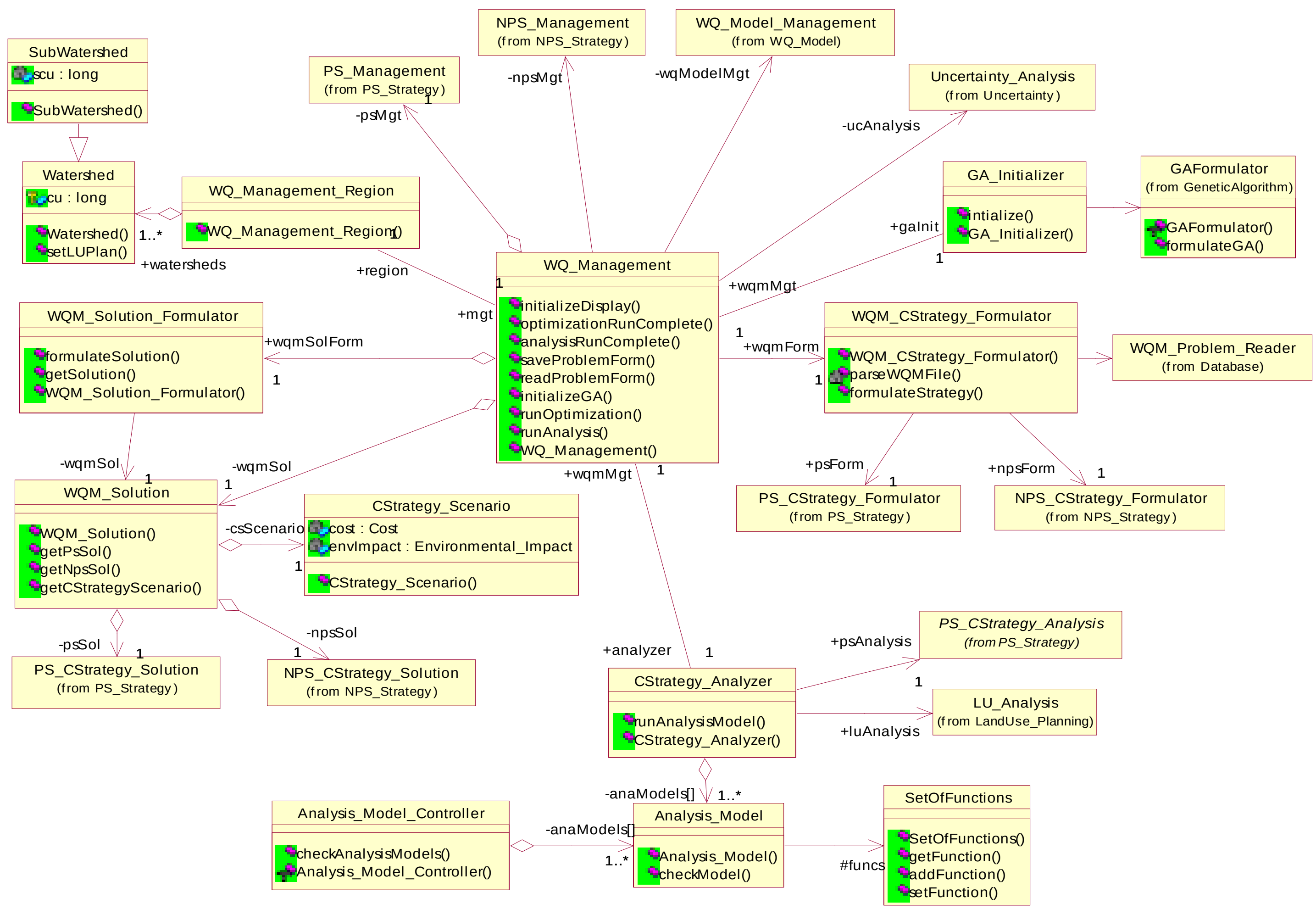


Figure 4.5 Class diagram 1 for the *category* `WQ_Management`

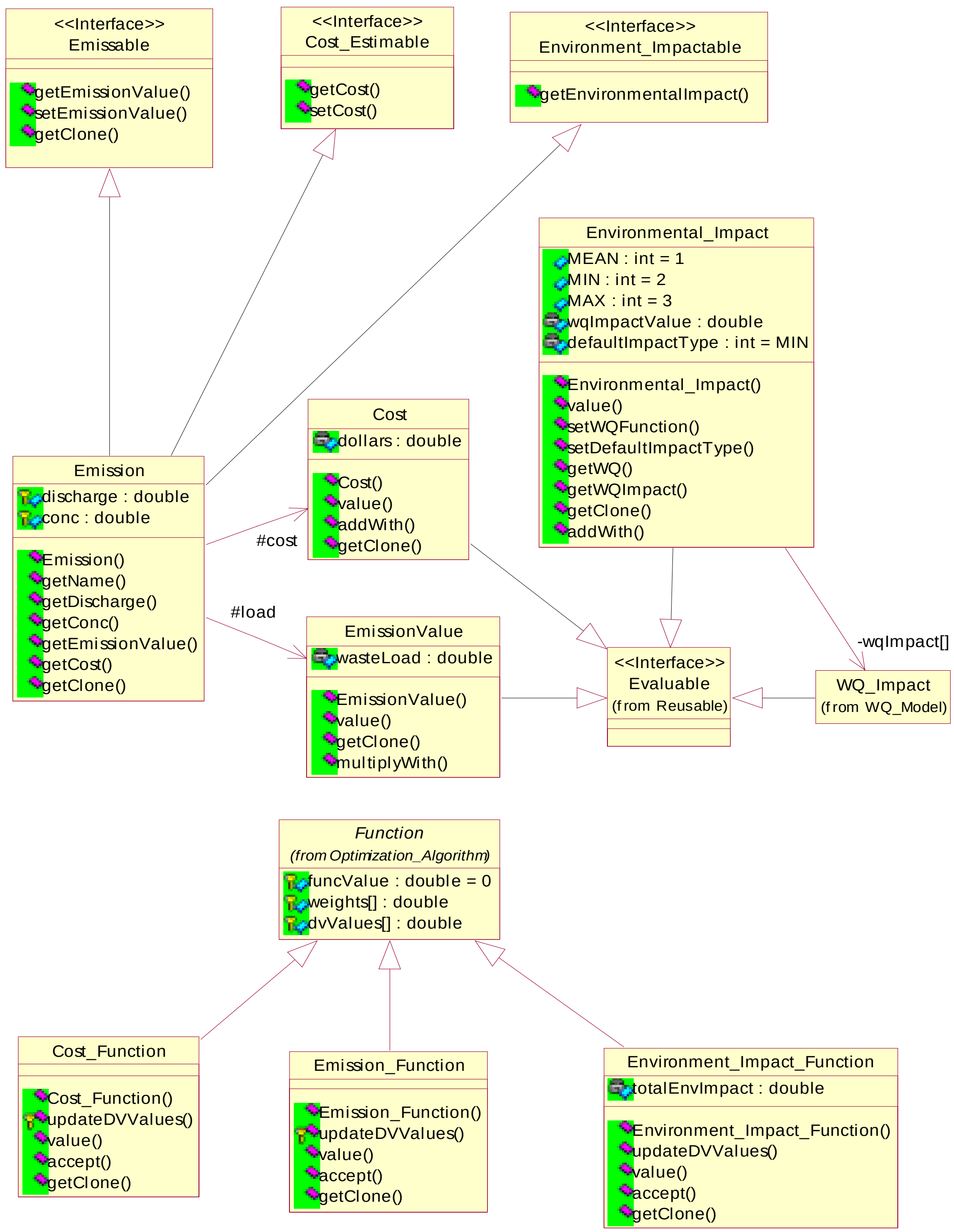


Figure 4.6 Class Diagram 2 for the *category* `WQ_Management`

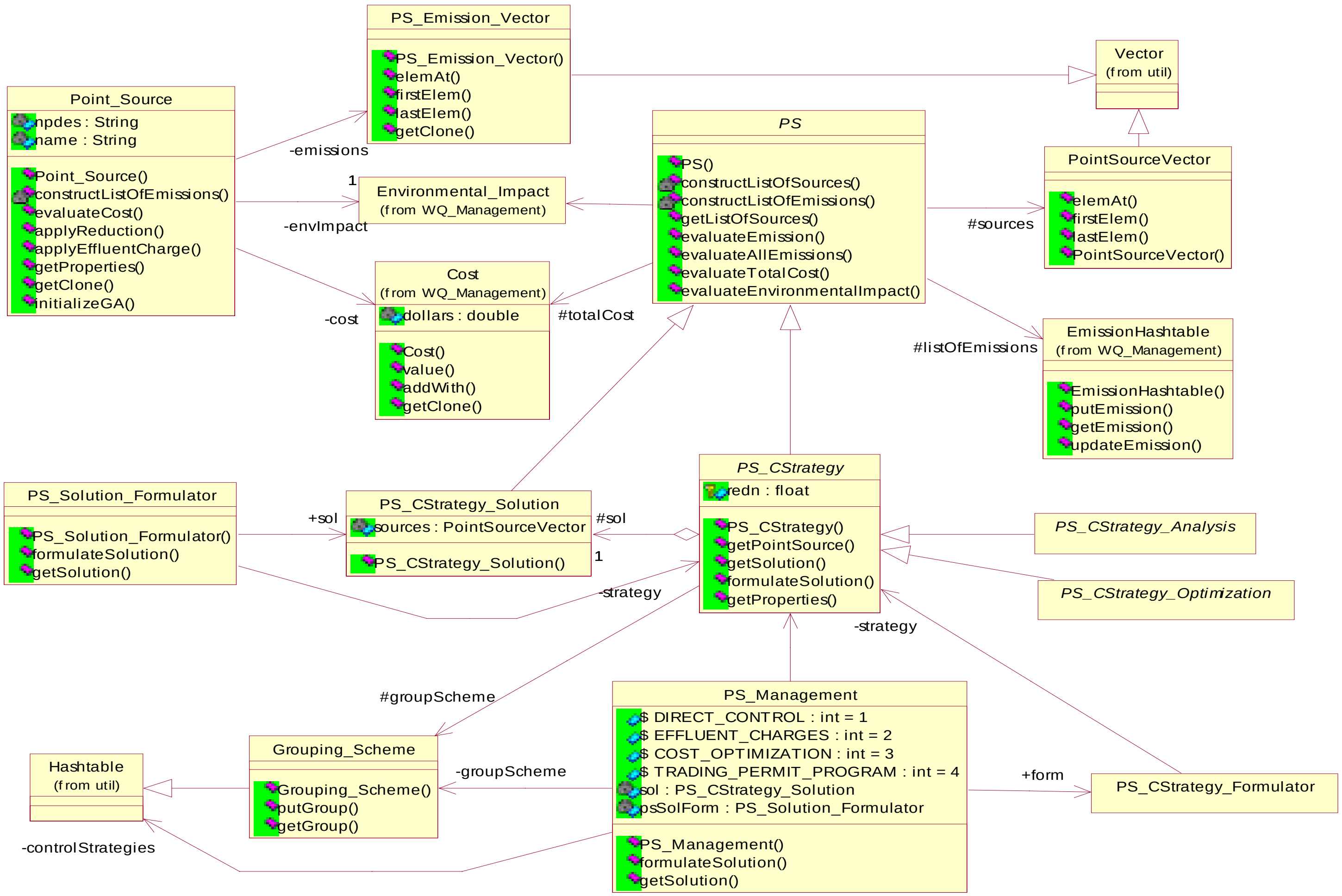


Figure 4.7 Class Diagram 1 for the *category* `PS_Management`

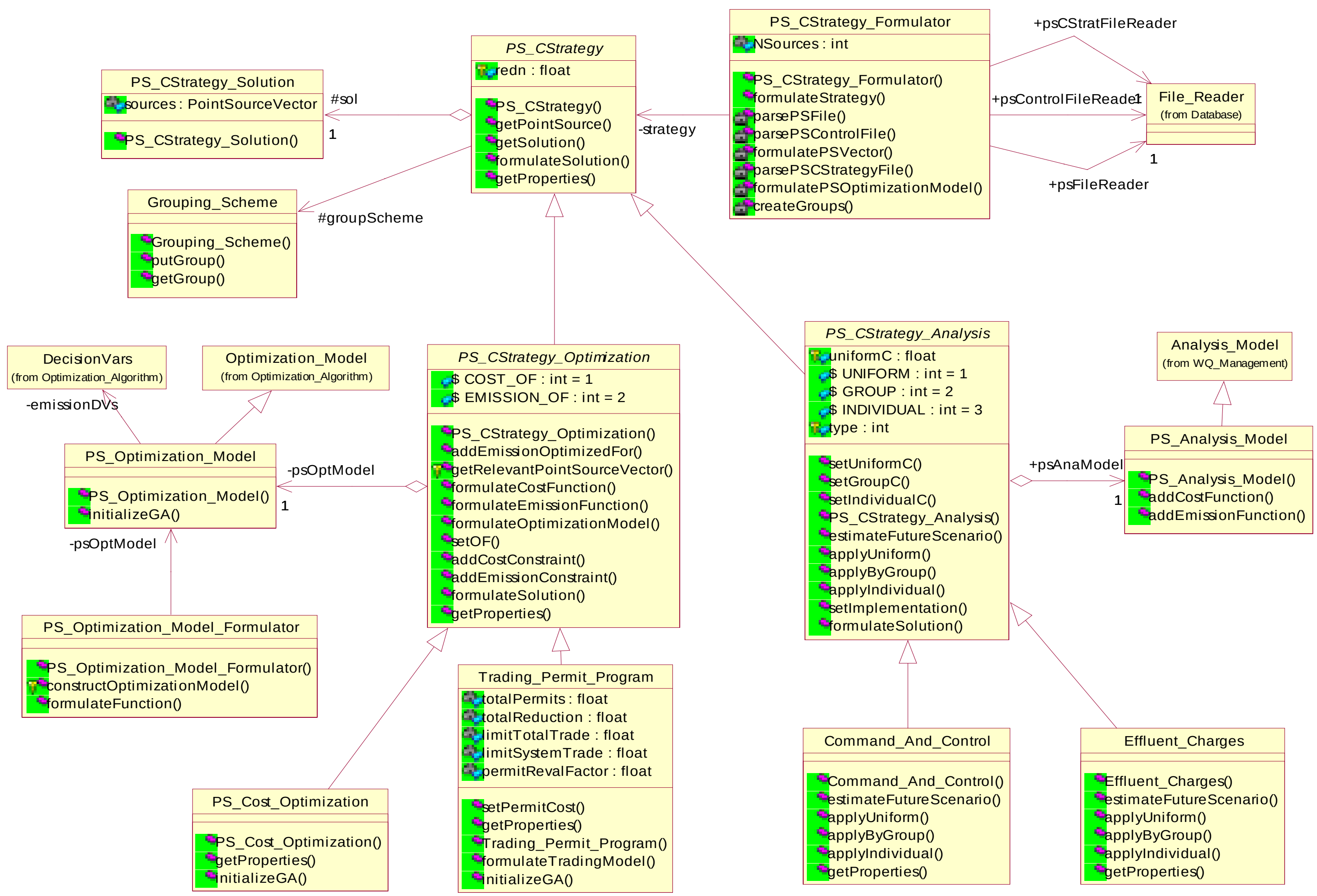


Figure 4.8 Class Diagram 2 for the *category* `PS_Management`

Figure 4.9 Class Diagram 3 for the *category* `PS_Management`

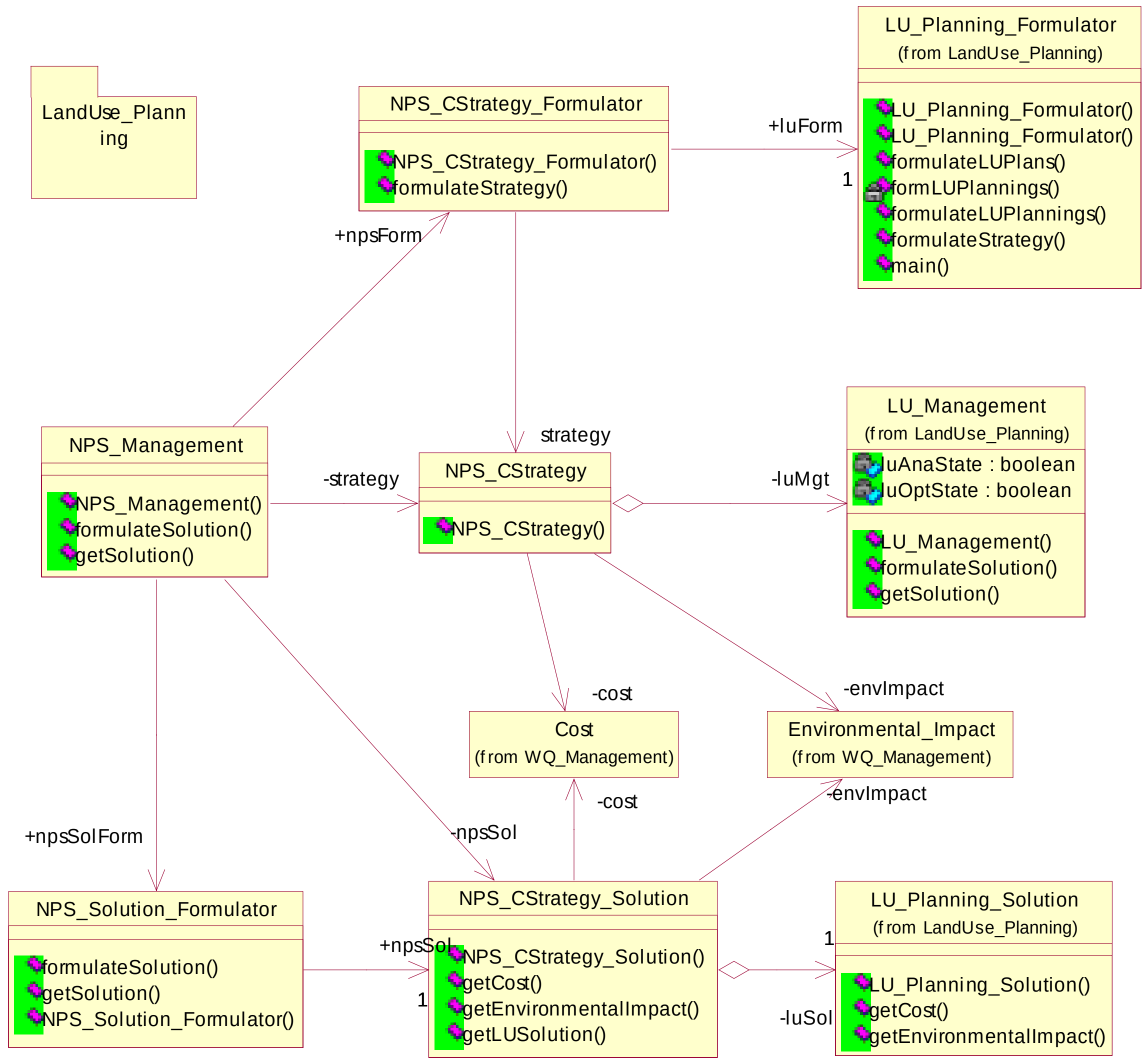


Figure 4.10 Class Diagram for the *category* `NPS_Management`

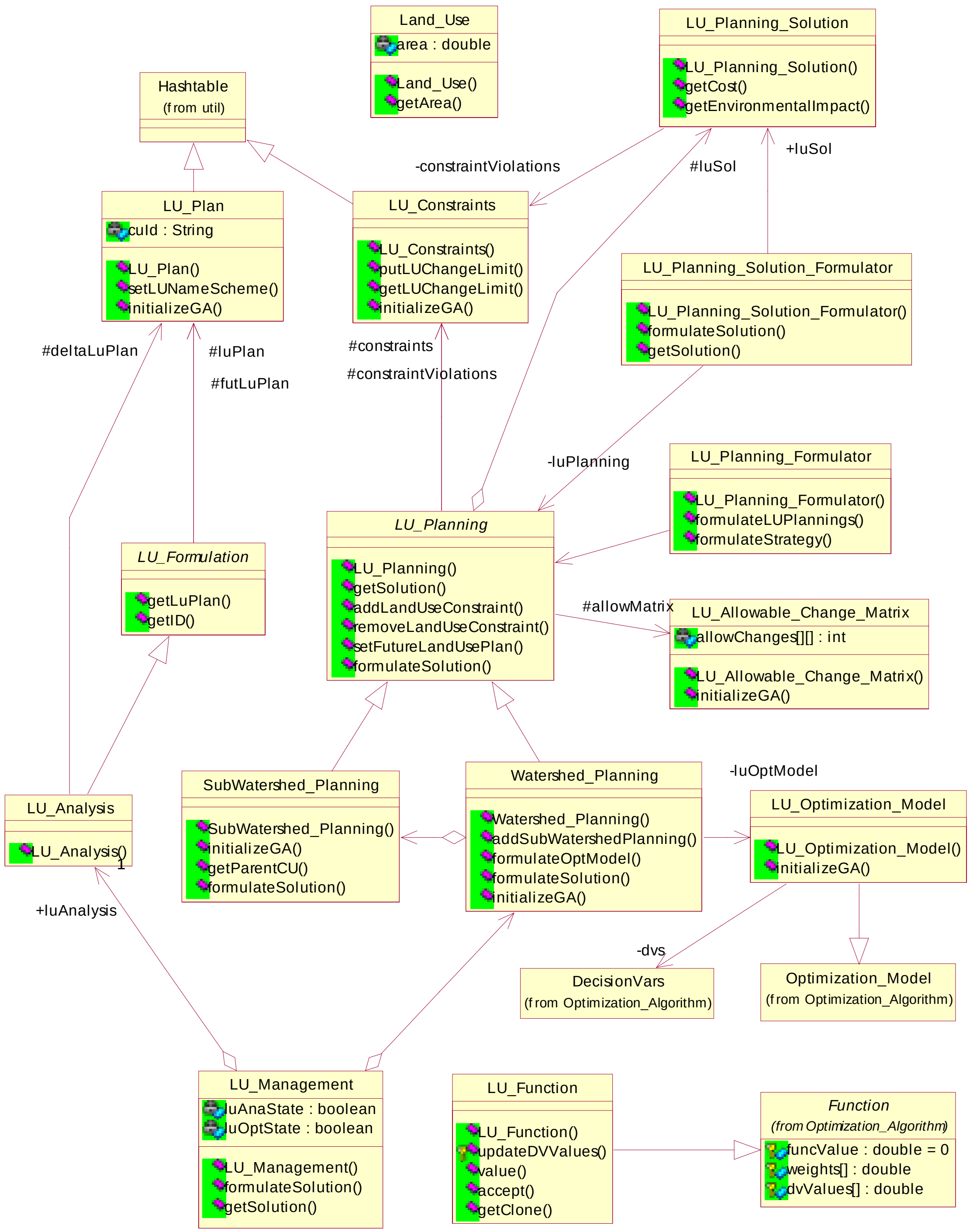


Figure 4.11 Class Diagram for the *category* `LandUse_Planning`

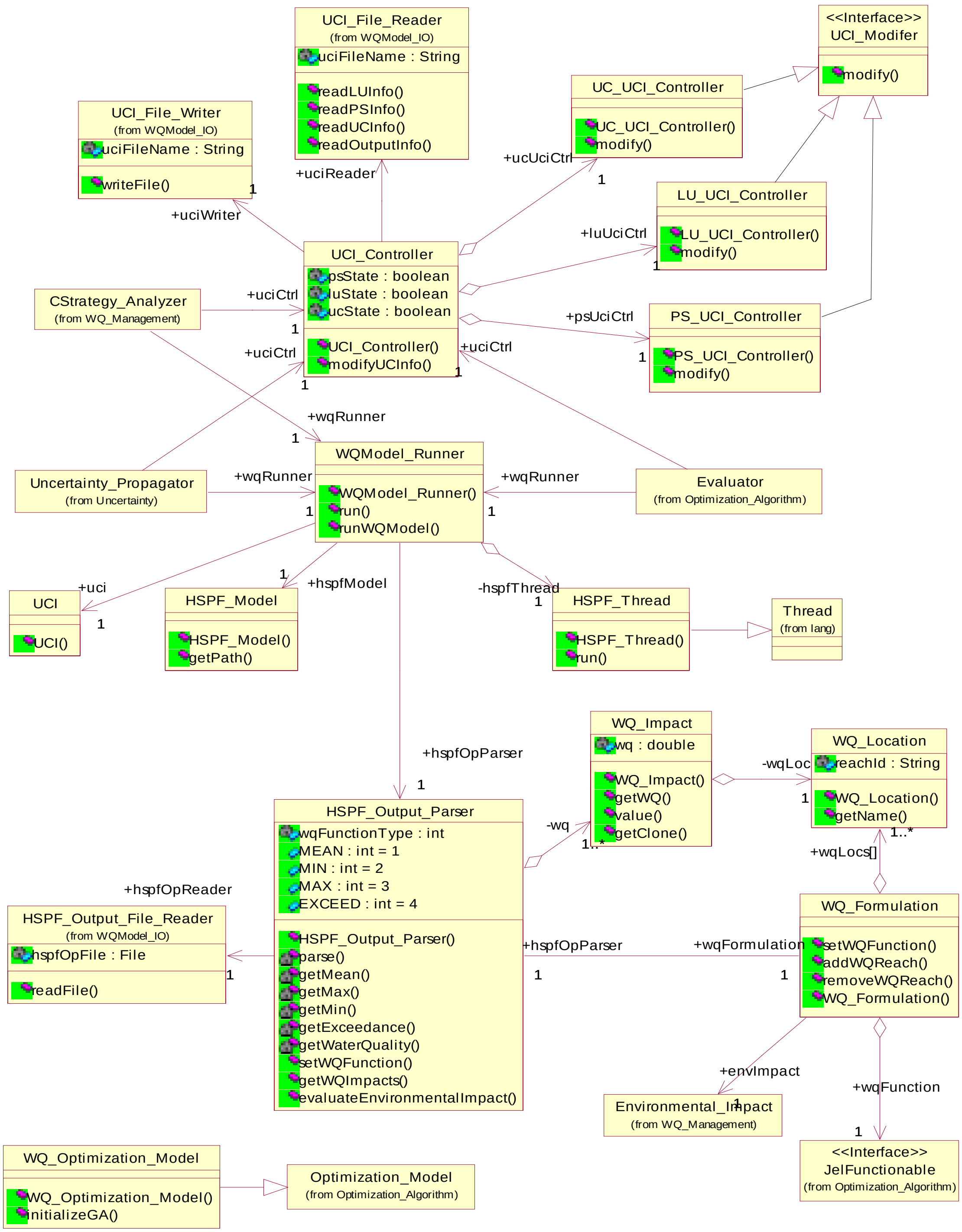


Figure 4.12 Class Diagram for the *category* `WQ_Model`

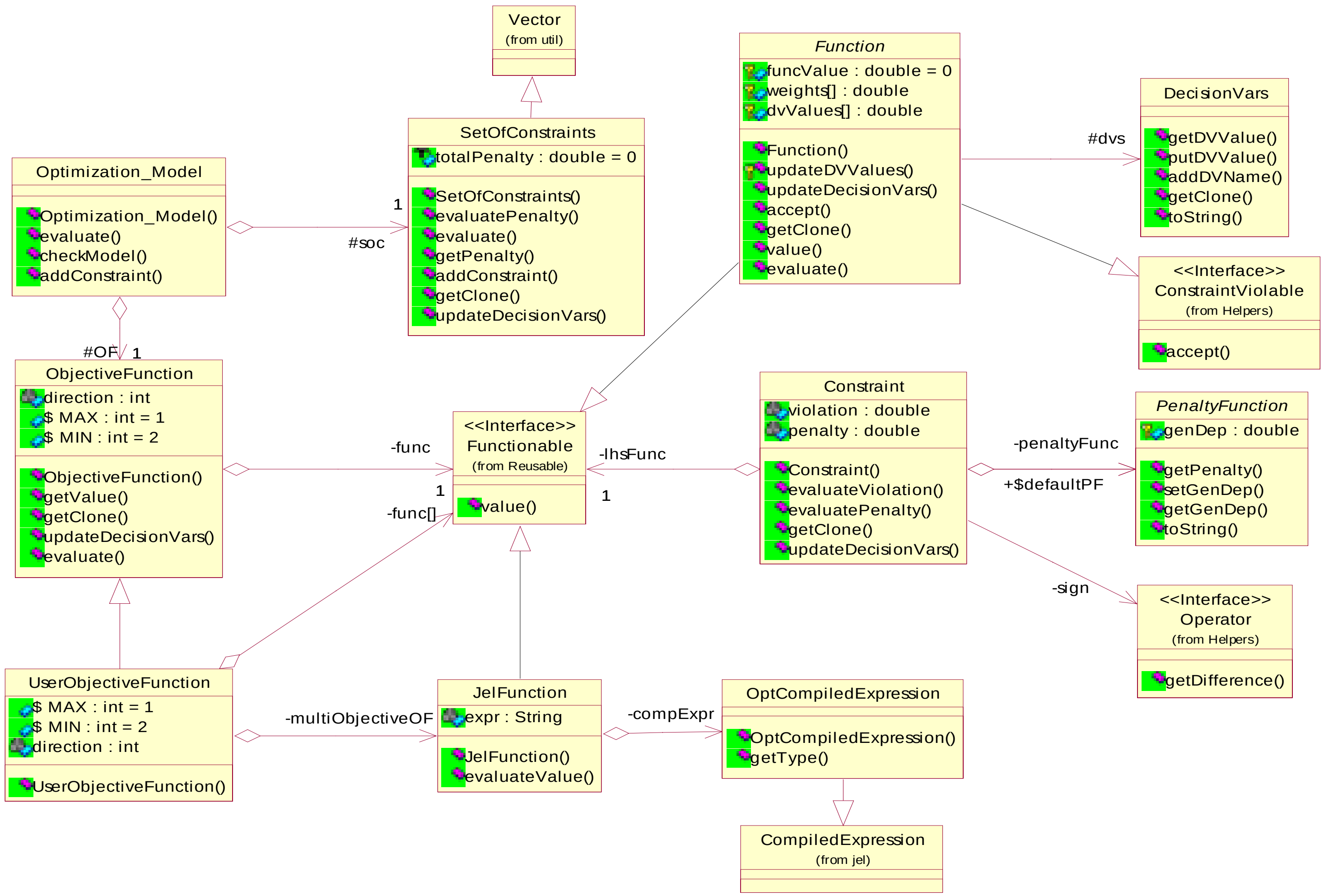


Figure 4.13 Class Diagram for the *category* `Optimization_Algorithm`

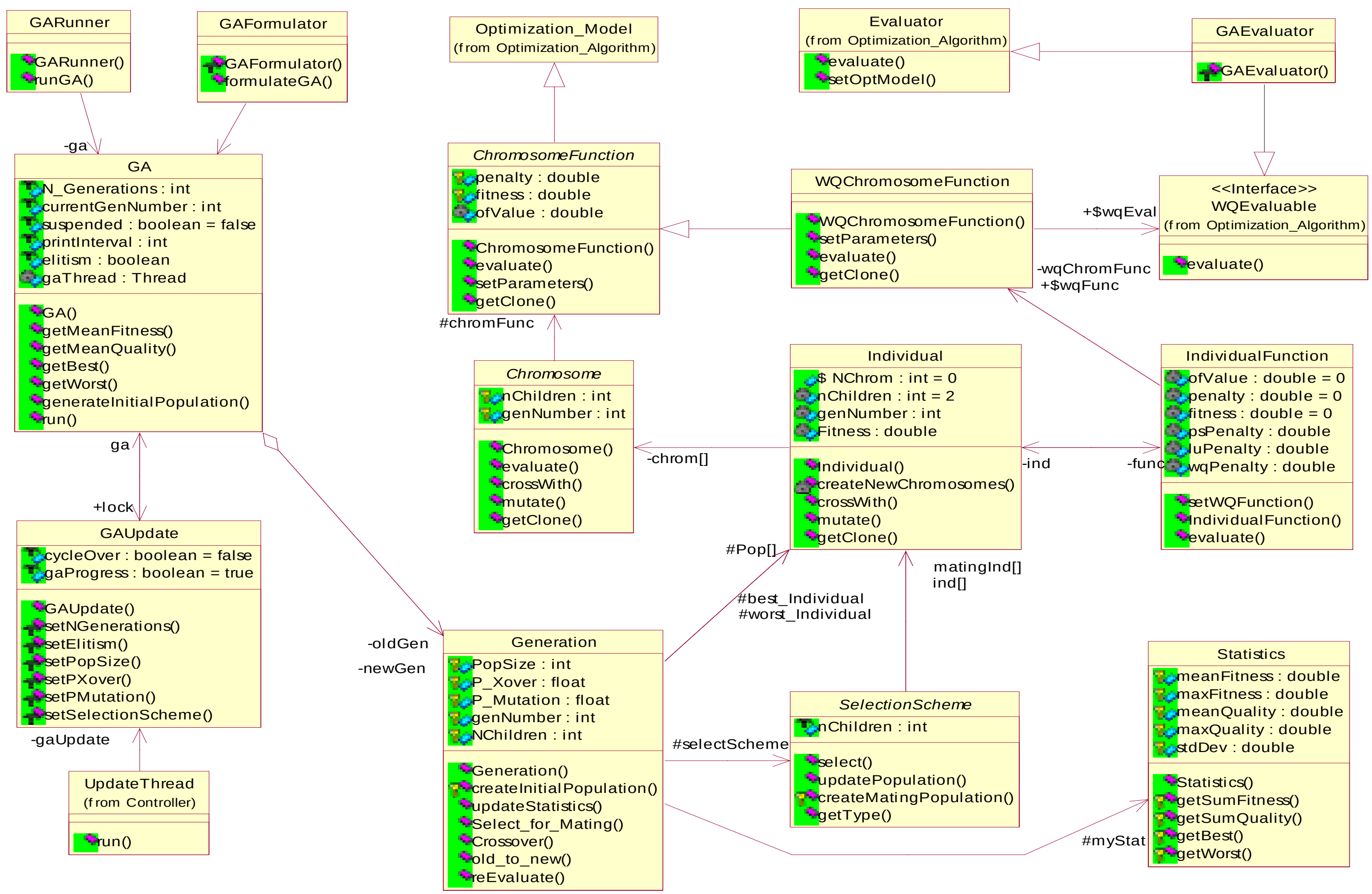


Figure 4.14 Class Diagram for child *category* `GeneticAlgorithm` of `Optimization_Algorithm`

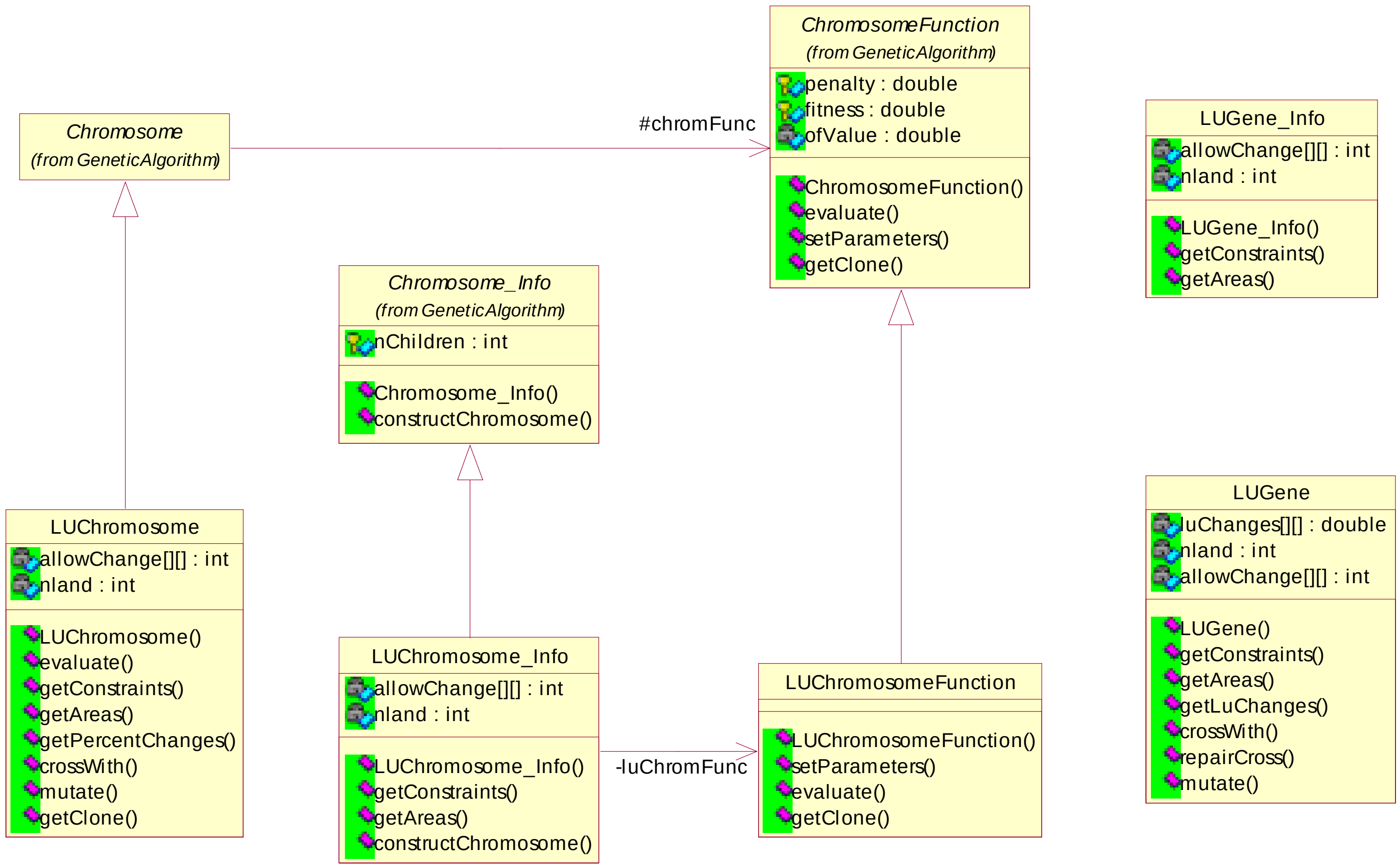


Figure 4.15 Class Diagram for the child *category* `LU` of `GeneticAlgorithm`

Figure 4.16 Class Diagram for the child *Category* `PS` of `GeneticAlgorithm`

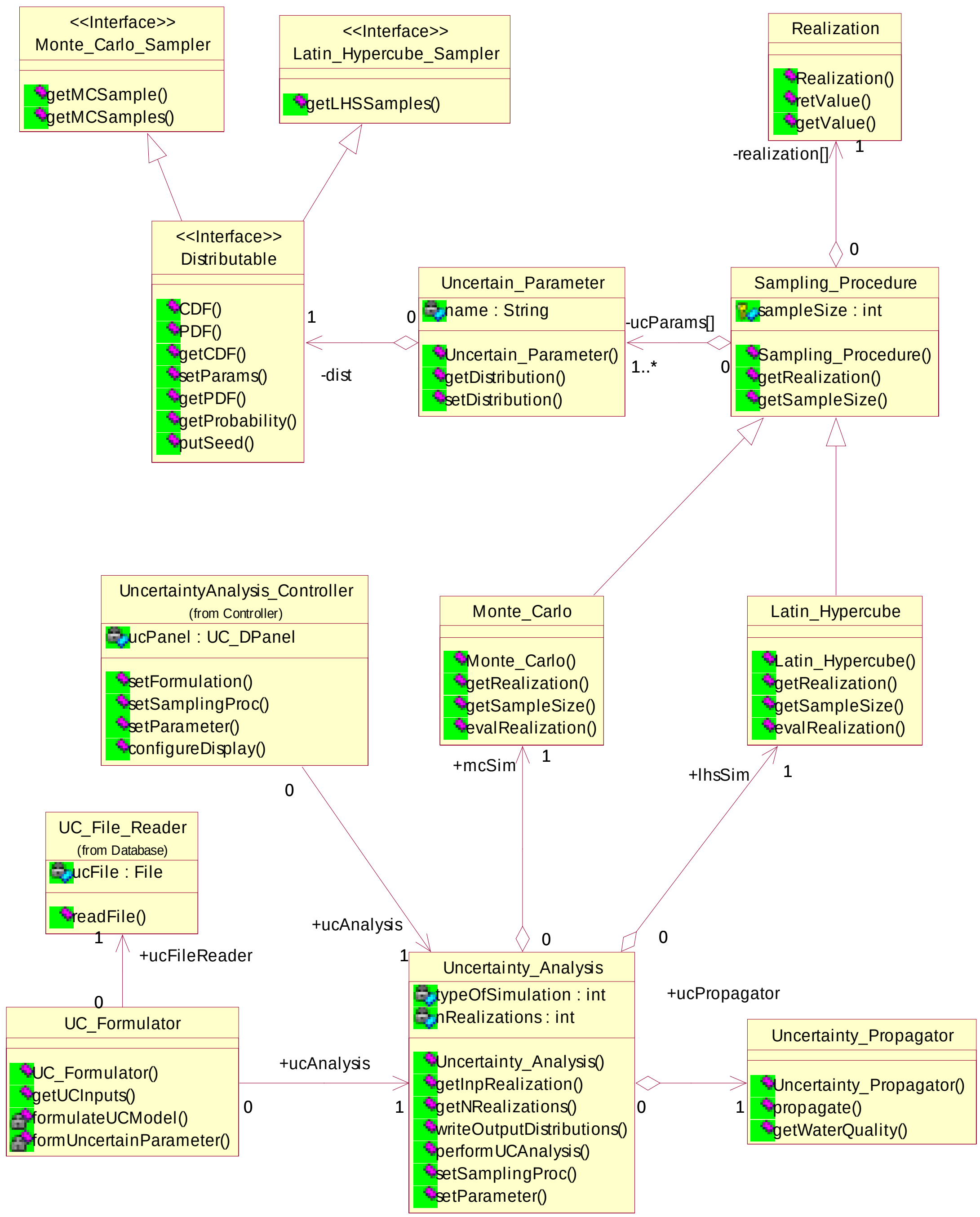


Figure 4.17 Class Diagram for the *category* `Uncertainty`

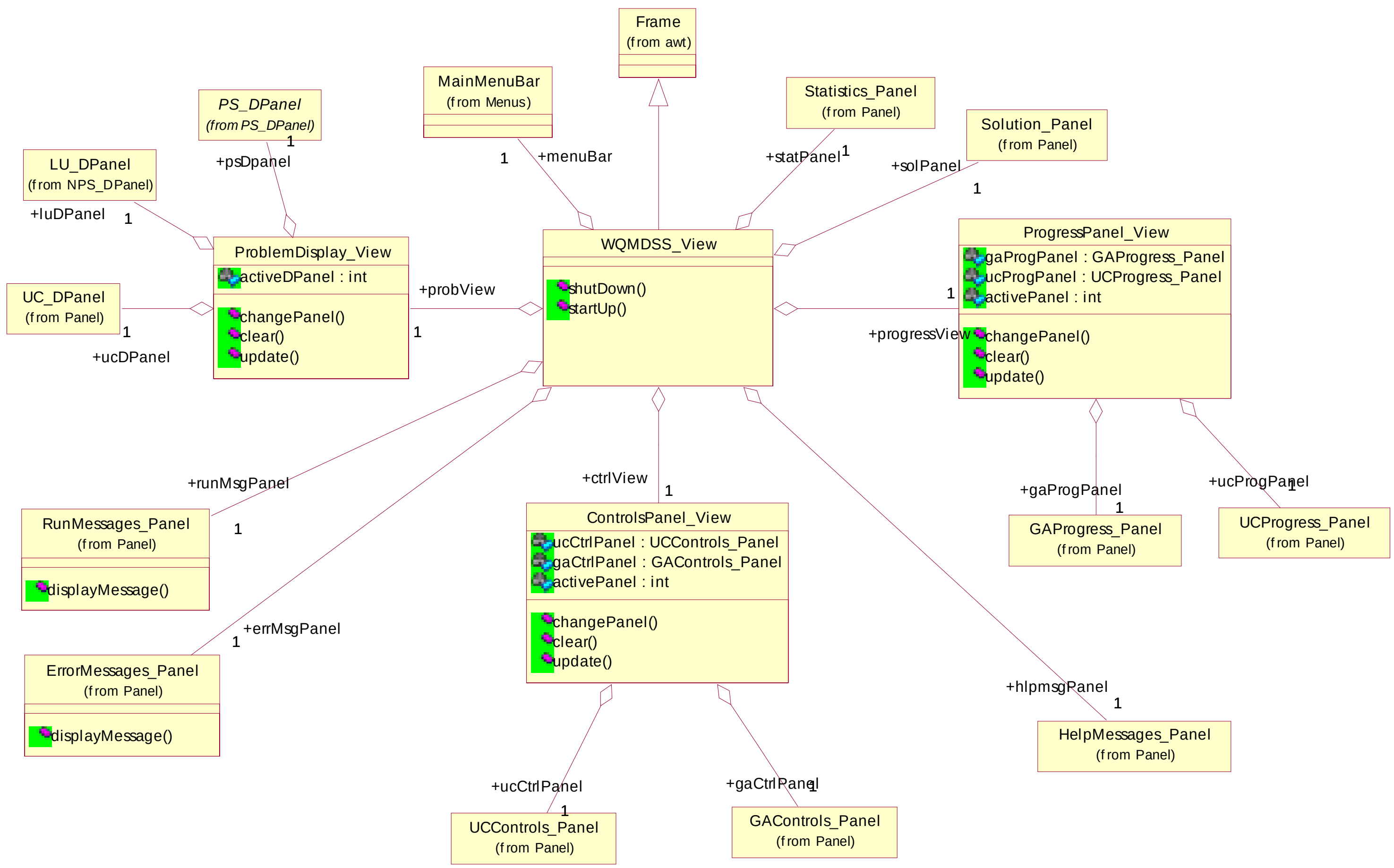


Figure 4.18 Class Diagram for the *category* `View`

The *class diagrams* shown in Figures 1.4 through 1.17 relate to the *package* structure shown in Figure 4.19.

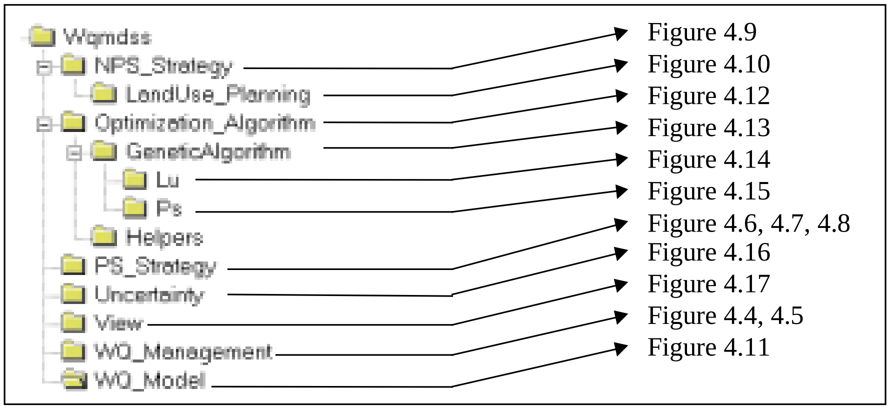


Figure 4.19 Package Structure

Section 4.10.1 through Section 0 describe the principal components of the *class* diagrams and relate the decision support methods and algorithms discussed in Chapter 2 with the software components.

4.10 System Description

The following notation is used in the discussion.

- The `Courier New` font is used for the name of a *class* or a *category*.
- The name of a *class* or a *category* that represents an entity or object is placed in parentheses, immediately following the entity or object. For e.g. the phrase, '…and the associated environmental impact (`Environmental_Impact`)…' implies that the *class* `Environmental_Impact` represents the entity environmental impact.
- A *coordinator class* coordinates the operations of its various *attributes* and is responsible for passing messages from other objects to its *attributes*.
- A *container class* serves as a data structure grouping similar objects.
- The *formulator* (`ABC_Formulator`) *classes* are design constructs and are responsible for creating the instances of *class* `ABC` by parsing information from input files or extracting required information from other objects.

4.10.1 Water Quality Management – Category WQ_Management

The *classes* in this *category* collectively contribute towards the:

- Formulation of watershed water quality management (WWQM) control strategies. This is facilitated by the *class* `WQ_Management` that acts as the *coordinator class* for the point source management (`PS_Management`) and non-point source management (`NPS_Management`) *categories*.
- Simulation of water quality indicators by linking with the *class* `WQ_Model_Management`.
- Analysis of control strategies by computing decision variables, cost, emissions and environmental impact. This is facilitated by the *classes* `CStrategy_Analyzer` and `Analysis_Model`.
- Optimization of control strategies by linking the representation of the point and non-point control strategies to the genetic algorithm *classes*. This is facilitated by the *class* `GA_Initializer`.

The *class* `WQ_Management` coordinates all of the above activities and is the root *class* of this *category*. The WWQM solution is represented by the *class* `WQM_Solution`.

This *category* contains a set of *decision variable classes* that represent the decision variables of the WWQM problem such as cost (`Cost`), emission (`Emission`) and environmental impact (`Environmental_Impact`). The `Cost_Function`, `Emission_Function` and `Environmental_Impact_Function` are *classes* that enable the formulation of summation functions that sum these decision variables over a set of objects. This provides the flexibility of grouping together a dissimilar set of objects such as point sources and non-point sources to sum over the desired decision variables. The implementation of the interface `Evaluable` by the *decision variable classes* imposes on these *classes* the operation `getValue()`. This imposition facilitates the formulation of weighted functions of the *decision variable classes* that are required for representing the objective function in a multi-objective problem formulation.

4.10.2 Point Source Management – *Category* PS_Management

The *classes* in this *category* represent real world emissions, emission controls, point sources and the point source control strategies that are required to control the emissions.

Figure 4.9 illustrates the emission control, the associated control cost and the emission *classes*. The point source emissions (`PS_Emission`)are controlled by applying a control (`Control`). Controls on point source dischargers to streams can be typically *class*ified as continuous (`Continuous_Control`) and discrete control (`Discrete_Control`). A continuous control has an efficiency $\eta(0,1)$ that is a continuous real variable. The cost function for this control is represented by a cost curve (`Treatment_Cost_Curve`) that can be linear (`Linear_Treatment_Cost_Curve`) or can be a mathematical function (`Function_Treatment_Cost_Curve`). A discrete control (`Discrete_Control`) consists of a set of control technologies (`Control_Technology`) that have a fixed efficiency and cost. Most real world point source models can be adequately represented by this set of *classes*.

Figure 4.7 depicts the *classes* that represent the point source (`Point_Source`), the control strategy (`PS_CStrategy`), the grouping scheme (`Grouping_Scheme`) and the point source management formulation (`PS_Management`). Point sources can be classified into groups, and various control strategies can be applied to different groups. The *class* `PS_Management` acts as the *coordinator class* and *container class* for control strategy formulations, and is the root of this *category*.

Figure 4.8 illustrates the various types of point source control strategies that can be implemented. These are broadly classified as control strategy analysis (`PS_CStrategy_Analysis`) and control strategy optimization (`PS_CStrategy_Optimization`).

Analysis consists of defining either directly or indirectly the emission values and simulating the water quality indicators to assess the environmental impact. The emission reductions can be specified either directly (`Command_And_Control`) indirectly by specifying effluent charges (`Effluent_Charges`) as discussed in Section 3.2.4.1.

Point source control strategy optimization is facilitated by the *class* `PS_Cstrategy_Optimization`. Users can define the objective function and optionally the constraints for the point source optimization model (`PS_Optimization_Model`). Users can specify the objective function and the constraint left hand side functions to be a cost function, emission function or an environmental impact function. The `PS_Optimization_Model` *class* adequately represents most optimization problem formulations including the ambient least cost (ALC) and emissions least cost (ELC) formulations discussed by Loughlin (1998).

The *class* `PS_Cost_Optimization` facilitates the solution of optimization problems discussed in Section 3.2.5.1 by linking with the appropriate *classes* in the *category* `Genetic_Algorithm`. The *class* `Trading_Permit_Program` facilitates the formulation of a transferable discharge permit (TDP) program model as discussed in Section 3.2.5.2. This TDP model is then solved by linking with the appropriate *classes* in the *category* `Genetic_Algorithm.PS`.

4.10.3 Non Point Source Management – *Category* NPS_Management

The principal *classes* in this *category* are `NPS_Management` and `NPS_CStrategy`. These *classes* mainly serve as *container* and *coordinator classes* for various types of non-point source control strategies such as reducing runoff by effective land use planning, buffer zoning streams and constructing system-wide detention ponds. The current state of system implementation supports non-point source management only by land use planning. This is because the watershed models are not designed to simulate directly the effect of other non-point source control strategies such as buffer zoning and detention ponds. With the incorporation of these additional strategies, the principal *classes* in this *category* will assume more significant roles.

The child *category* `LandUse_Planning` represents various land use planning formulations.

4.10.3.1 Land Use Planning – Category LandUse_Planning

The *classes* in this *category* represent the formulation of a land use analysis or land use planning problem as discussed in Section 3.2.4.2.

Land use analysis (`LU_Analysis`) consists of defining the incremental changes in land use plans (`LU_Plan`) and simulating the change in the water quality indicators. This enables users to perform a 'what-if' analysis for different scenarios of land uses. This enables the refinement of benchmark solutions.

Land use planning optimization discussed in Section 3.2.5.3 consists of defining an objective function and/or constraints. The objective function and the left hand side constraint functions can be land use functions (`LU_Function`) that are weighted functions of land use changes or environmental impact functions. Usually, for a land use planning problem, the constraints are specified as upper bounds on the change of a particular land use acreage. This is facilitated by the *class* `LU_Constraints`. The land use allowable change matrix (`LU_Allowable_Change_Matrix`) is a user specified binary matrix that indicates if one type of a land use can be changed to another.

The land use constraints and the allowable change matrices can be formulated at the watershed and the sub-watershed levels. These are then assembled into an instance of the land use optimization model (`LU_Optimization_Model`) that is solved by linking with the appropriate *classes* in the *category* `GeneticAlgorithm.LU.`

The *class* `LU_Management` is the *container* and *coordinator class* for the land use analysis and land-use planning formulations.

4.10.4 Water Quality Model – *Category* WQ_Model

The *classes* in this *category* collectively contribute towards running the water quality model for values of decision variables and model parameters given by the optimization and uncertainty algorithms and returning appropriate water quality indicators. This set of tasks is carried out in the following manner.

1. The *class* `UCI_Controller` is instructed by either the:
    - Control strategy analyzer (`WQ_Management.Cstrategy_Analyzer`) with user defined values of point source emissions, land use plans etc, or
    - Evaluator (`Optimization_Algorithm.Evaluator`) with the decision variable values. If the genetic algorithm is used as the optimization algorithm, these decision variable values represent the genetic information decoding.

- The uncertainty propagator (`Uncertainty.Uncertainty_Propagator`) with the uncertain parameter realizations.

2. The `UCI_Controller` reads the users input file (UCI), instructs the corresponding UCI controllers to change the input file content and commits the changed information to the input file.
3. The `WQ_Model` runner runs the water quality model (HSPF), waits for the run to complete and instructs the output parser to formulate the water quality indicators represented by the *class* `WQ_Impact` and returns these to the calling algorithm.

The water quality optimization model (`WQ_Optimization_Model`) is the users' formulation of the objective function and optionally the constraints. The objective function and the left hand side functions of constraints may be environmental impact functions.

4.10.5 Optimization Algorithm – *Category* Optimization_Algorithm

A typical mathematical programming problem consists of the definitions of the decision variables, the objective function and constraints that are functions of the decision variables. Figure 4.13 shows the *classes* that represent collectively a generic mathematical programming problem given by:

$$\begin{aligned}
&MAX/MIN \quad f(X)\\
&subject\ to:\\
&X \in F_{R^N}\\
&where:\\
&X = vector\ of\ decision\ variables\\
&f = objective\ function\\
&F_{R^N} = feasible\ search\ space
\end{aligned} \tag{4.1}$$

The optimization model (`Optimization Model`) consists of an objective function (`ObjectiveFunction`) and a set of constraints `SetOfConstraints`. The objective function (`ObjectiveFunction`) consists of a direction and the function that is optimized (`Functionable`). A constraint (`Constraint`) is composed of a left-hand side function (`Functionable`), a sign (`Operator`), a bound (`Object`) and a penalty function (`Penalty_Function`). `Functionable` can be any object that implements an operation `getValue()`. Thus, any *subclasses* of the *class* `Function` such as

`Cost_Function`, `Emission_Function` etc. as well as any mathematical function (`JelFunction`) of the decision variables can be used directly in the objective function or the constraints.

The abstract *class* `Function` defines a summation function over the decision variables (`DecisionVars`).

4.10.5.1 The Java Expressions Library (JEL)

To augment the evaluation process with the evaluation of generic mathematical expressions of objects that are instances of different *classes*, a Java Expressions Library (JEL) developed by Metlov (1998) is used. JEL enables the evaluation of the user-defined functions. JEL compiles mathematical expressions directly to Java byte-codes, allowing their fast evaluation. JEL supports all the Java primitive types as well as object types. Functions in JEL are methods of the Java objects. No front ends for functions have to be written. It is possible to use Java objects directly, exporting their functions to JEL. Appendix A discusses an example of the manner in which JEL can be used in the evaluation process.

4.10.5.2 Genetic Algorithm – Category GeneticAlgorithm

Designing appropriate data structures to represent the hierarchy of genetic information content is of central importance to the GA search procedure. Typically, the *aggregation hierarchy* can be represented adequately by a *gene* at the lowest level. A chromosome (`Chromosome`) is a vector of similar types of genes. An individual (`Individual`) is a *container* for a set of chromosomes. A *population* comprises of a set of individuals and is the basic evolving unit. Individuals in a population undergo selection, crossover and mutation to produce the next generation of individuals. The binary tournament and roulette-wheel selection schemes are supported. The crossover and mutation operators are implemented by *subclasses* of `Chromosome`. The genetic algorithm (GA) is the *coordinator class* that evolves the generations, implements elitism etc.

The abstract *class* `ChromosomeFunction` represents the optimization model formulation. The genes of the `Chromosome` represent the decision variables of the optimization model. *Subclasses* of this *class* obtain the information regarding the objective function values, constraint violations etc. from the decision variable values. The *class* `WQChromosomeFunction` represents the water quality optimization model. Each instance of this *class* needs to be evaluated for objective function value and constraint violations. The water quality decision variable values that are required for this evaluation are obtained from a run of the water quality model. The *class* `IndividualFunction` represents the complete optimization model formulation for the watershed water quality management problem.

Point Source GA Classes – Category PS

The *classes* in this *category* mimic a subset of the *class* hierarchy in the *category* `PS_Management`. The *class* `PSChromosome` is similar to the *class* `PS_Cstrategy_Optimization` and is the basic evolving unit for the point source control strategy optimization. This *class* implements a uniform crossover. It crosses corresponding instances of point source genes (`PSGene`) which in turn cross corresponding instances of the emission genes (`EmissionGene`). The real mixing of genetic information occurs at the control gene (`ControlGene`) level.

The *class* `TradingPSChromosome` facilitates the simulation of a multilateral trading market as discussed in Section 3.2.5.2. The source status indicates the status of emitter, if a buyer (B) or a seller (S). The allowable trade matrix is a binary, $N \times N$, skew-symmetric matrix that represents the occurrence of a trade between emitters i and j. The trade matrix is a real-valued, $N \times N$, skew-symmetric matrix each element of which, t_{ij} , represents a trade between the emitters i and j. The crossover procedure implemented by this *class* is designed to maintain feasibility of the resulting child chromosomes.

Land Use Planning GA Classes– Category LU

The *classes* in this *category* mimic a subset of the *class* hierarchy in the *category* `NPS_Management.LandUse_Planning`. This set of *classes* was designed and implemented by another member of the research group, S.Kishan Chetan. The *class* `LUChromosome` is similar to the *class* `Wateshed_Planning` and is the basic evolving unit for the land use planning optimization problem. The crossover and mutation operators are designed such that feasibility is maintained.

4.10.6 Uncertainty Analysis – *Category* Uncertainty

The *classes* in this *category* represent the uncertainty analysis procedures discussed in Section 3.2.7. Most of these *classes* were implemented by another member of the research group, S. Kishan Chetan. An uncertain parameter (`Uncertain_Parameter`)is modeled as a probability distribution (`Distributable`). Table 4.2 shows the types of probability distributions that are supported. Some of these distributions were extended from the *class* library `or.drasys.prob` that was obtained from (ORObjects for Java). The sampling procedure (`Sampling_Procedure`) formulates a set of realizations (`Realization`) of these uncertain parameters. Monte Carlo and Latin Hypercube Sampling are the types of sampling procedures implemented. The *class* `Uncertainty_Analysis` is the *coordinator class* and is the root of this *category*. The *class* `Uncertainty_Propagator` propagates the uncertainty by linking with the *classes* `UCI_Controller` and `WQ_Model_Runner` in the *category* `WQ_Model`.

Table 4.2 Probability Distributions

Probability Distribution	**Extended Form**
Normal	drasys.or.prob.NormalDistribution
Poission	drasys.or.prob.PoissionDistribution
Uniform	drasys.or.prob.UniformDistribution
Exponential	drasys.or.prob.ExponentialDistribution
Triangular	-
Weibull	-
LogNormal	-

4.10.7 Graphical User Interface Display – *Category* View

The set of *classes* in this *category* are linked together to form the user interface to the DSS core model *classes* discussed above. The development of these *classes* is assigned to the build *category* B2 in the Requirements Trace Matrix (RTM) shown in Table 2.1 and is part of the ongoing research.

4.11 Hotspots and Plug-Points for the DSS Framework

The set of *classes* developed to represent the various control strategies, optimization model structure, genetic algorithm and the uncertainty analysis algorithms offer considerable flexibility for customization of this framework. The customization is done for incorporating additional or different utilities specified by the user.

The *classes* `WQ_Management`, `PS_Management` and `NPS_Management` are the coordinators that handle different control strategies. Additional control strategies can easily be added to the existing framework. For e.g., a different point source control strategy can be constructed by extending the *classes* `PS_CStrategy_Analysis` or `PS_CStrategy_Optimization` in the *category* `PS_Strategy`. Non-point control strategies such as riparian buffer zoning and regional detention pond can be added by developing child *categories* of the *category* `NPS_Management` that are similar to `LandUse_Planning`.

Currently, genetic algorithms are used as the optimization procedures. Other heuristic optimization techniques such as simulated annealing (SA) can be integrated into the DSS framework by developing a child *category* containing the SA *classes* to the *category* `Optimization_Algorithm`.

Currently, the DSS provides the capability of providing stochastic sampling by using Monte Carlo and Latin Hypercube sampling procedures. Other sampling procedures can be included by extending the *class* `Sampling_Procedure` in the *category* `Uncertainty`. Additional probability distributions can be included by implementing the interface `Distributable` in the *category* `Uncertainty`.

The watershed model HSPF is currently used for simulating the water quality. Other watershed models such as Soil and Water Assessment Tool (SWAT) and Storm Water Assessment Model (SWMM) can be easily integrated into this framework by replacing the *class* `UCI_Controller` with an appropriate *classes* and implementing the necessary input and output parsers.

All the *categories* of the DSS except the *categories* `NPS_Management` and `WQ_Model` are generic and can be used for the development of decision support frameworks for other applications. For e.g., the *categories* `PS_Strategy`, `Uncertainty`, `Optimization_Algorithm`, `GeneticAlgorithm` can be used in the development of a decision support framework to aid environmental decision making that addresses cross-media issues in an integrated manner.

References

1. Booch, G., Rumbaugh, J. and Jacobsen, I., (1997). *Unified Modeling Language for Object-Oriented Development*, V 1.1, Rational Software Corporation, Santa Clara, CA.
2. Booch, G., (1995). *Managing the Object-Oriented Project*, Addison-Wesley.
3. DRA Systems, (1997), *Operations Research Java Objects*, URL: http://OpsResearch.com/OR-Objects/index.html.
4. Fowler, M. and Scott, K., (1997). *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley.
5. Gosling, J., Joy, B., Steele, G, (1989). *The Java Language Specification*, Addition-Wesley Publishing Co., Inc., Reading, MA.
6. Horstmann, C.S., and Cornell, G., (1999). *Core Java Volume I-Fundamentals*, Sun Microsystems Press.
7. Jacobsen, I., Christerson, M., Jonsson, P., (1994). *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley.
8. Metlov, C., 1998. *Java Expressions Library (JEL) Reference Manual*, URL: http://galaxy.fzu.cz/JEL.
9. Michalewicz, Z., (1996). *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag.
10. Michalewicz, Z., (1995). *Genetic Algorithms, Numerical Optimization and Constraints*, Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan-Kaufman, pp.151-159.
11. Object Management Group (1997). *UML 1.1 Documentation*. URL: http://www.rational.com/.
12. Rizzoli, A.E., Davis, J.R., and Abel, D.J., (1998). *Model and data integration and re-use in environmental decision support systems*, Decision Support Systems, 24, pp. 127-144.
13. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., (1991). *Object Oriented Modeling and Design*, Prentice Hall, Inc.
14. Texel, P.P., and Williams, C.B., (1997). *Use Cases Combined With Booch/OMT/UML: Process and Products*, Prentice Hall PTR.

5 Summary

A collection of genetic algorithm (GA) tools is developed for generic use. A subset of the Unified Modeling Language (UML) is used for object oriented design and the implementation of the *classes* for a simple steady state GA was carried out using Java^{TM}. This set of *classes* represents various data structures, representations, GA operators such as crossover and mutation and various selection schemes. A set of object *classes* to represent a generic optimization model structure is also developed. This set of *classes* enables the construction of a generic mathematical programming problem by the specification of the objective function and the constraints. The Java Expressions Library (JEL) is used to assist in the evaluation of user defined functions. A penalty function approach is used to handle constraint violations for solving constrained optimization problems using GA methodology. Different forms of penalty functions can be selected to penalize the constraint violations. The GA fitness function is a user defined function of the objective function and the constraint penalties.

A graphical interface is developed that enables a user to formulate a any mathematical function optimization problem in a simple algebraic format and solve it by choosing from a combination of various decision variable representations, GA operators, penalty functions etc. The decision variables, their representations, objective functions, constraints and fitness function that characterize the optimization problem are specified interactively by the user. The problem formulation can be saved to and loaded from specifically formatted problem files. The system is capable of checking the optimization formulation for errors and reporting these errors to the user. The interface also has an online help system that explains various functionalities and operations. The GA parameters and operators such as the crossover, mutation operators and probabilities, selection schemes, elitism, population size can be dynamically controlled during the GA run. This feature can be used to gain an understanding of the effect that the GA operators have on the GA search. The progress of the GA run can be monitored by viewing dynamically, the characteristics of the best solution, the generations statistics, the convergence of fitness and objective function values and the constraint violations.

The GA object *classes*, the optimization model *classes* and the user interface comprise together the Generic GA Based Optimizer (GeGaOpt). The use of the GeGAOpt is demonstrated for the solution of various unconstrained as well as constrained optimization problems. The GA object *classes* and the optimization model *classes* can be customized easily for application to a wide range of problems and be used in education and research.

A decision support framework is developed to enhance the capabilities of existing watershed management framework (BASINS) by including a systematic search for 'good' environmental control strategies and procedures for the estimation of uncertainty in achieving water quality targets. This framework enables users to formulate various point source control strategies such as command and control (CAC), effluent charges, transferable discharge permit (TDP) programs and non-point control strategies such as land use planning. It also allows the user to perform uncertainty analysis of the model parameters by various stochastic sampling procedures such as Monte Carlo and Latin Hypercube sampling. The integration of HSPF into this framework enables a user to easily perform 'what-if' analysis to aid iterative decision making and enables the HSPF run information to be included in the users' definition of the control strategy. As traditional optimization procedures impose severe restrictions in using complex nonlinear environmental processes within a systematic search, genetic algorithms (GAs), general, probabilistic, heuristic search procedures are used for global optimization.

The primary objectives of the decision support framework, the various control strategy formulations, analysis, optimization models, genetic algorithms, uncertainty analysis procedures form the basis of the requirements for the design. The design of this system is developed using the software engineering approach such as object oriented analysis (OOA) and object oriented design (OOD). Unified Modeling Language (UML) was used for the design. The core model *classes* of the framework are implemented using the Java™ programming environment. All the core *classes* that are required to represent the control strategies, optimization and uncertainty analysis algorithms, have been implemented in the current implementation. The graphical user interface (GUI) *classes* and the controller *classes* that link the GUI *classes* to the model *classes* are being developed. The design of user interfaces to integration with BASINS to build a state of the art watershed management decision support system is an ongoing effort.

Appendix A

A.1 The Java Expressions Library (JEL)

Java Expressions Library (JEL), developed by Metlov (1998) is used in the development of GeGAOpt and the DSS. JEL enables the evaluation of the user-defined functions. The key feature of JEL is that it is a compiler. To write the interpreter for expressions is not a very difficult task but it results in huge performance losses because Java is an interpreted language. JEL compiles mathematical expressions directly to Java byte-codes, allowing their fast evaluation. JEL supports all the Java primitive types as well as object types. Functions in JEL are methods of the Java objects. No front ends for functions have to be written. It is possible to use Java objects directly, exporting their functions to JEL through `gnu.jel.Library` *class*. During evaluation, expressions are converted into the Java byte-codes on the fly. Then, they are loaded into the Java virtual machine and returned to the caller as an instance of the *class* that is a *subclass* of the *class* `gnu.jel.CompiledExpression.`

If the Java virtual machine has JIT compiler, then the expressions can be transparently compiled into the native machine code, resulting in performance higher that for most C written interpreters. JEL is easily portable across platforms and does not require recompilation.

JEL is available under the GNU General Public License. This license guarantees freedom to share and change free software. The detailed terms and conditions of this license can be found at http://www4.ncsu.edu/~avparand/GeneticAlgorithm/license.txt.

A.2 Using JEL

JEL can be instructed to use specific libraries of functions. For e.g., the following Java code enables the use of functions of the class `java.lang.Math` and of the class whose instance is the object `variables.`

```
import gnu.jel.Evaluator;
import gnu.jel.CompiledExpression;
import gnu.jel.Library;
import gnu.jel.CompilationException;
import java.math.*;
import java.util.*;

public abstract class AnyFunctionEvaluator {
        String expression;
        CompiledExpression expr_c;
        Library library;
        Object[] variableObjects;

        void setUpLibrary(Object variables) {
                //Construct Static Library
                Class[] staticLib=new Class[1];
                try {
                        staticLib[0] = Class.forName("java.lang.Math");
                }
                catch(ClassNotFoundException e) {};
                //Construct Dynamic Library
                Class[] dynamicLib=new Class[1];
                dynamicLib[0] = variables.getClass();

                //Construct the variables instance
                variableObjects=new Object[1];
                variableObjects[0] = variables;

                //Set up library
                library = new Library(staticLib, dynamicLib);
        }

        public CompiledExpression getCompiledExpression(String expr) {
                expression = expr;
                compileExpression();
                return expr_c;
        }

        /*Compile the expression*/
        public String compileExpression() {
                try {
                        expr_c = Evaluator.compile(expression, library);
                        return "compiled";
                }
                catch (CompilationException ce) {//Compilation Error message }
            }
        }

        /*Evaluate the Expression*/
        public double evaluate() {
                Number result = null;
                if (expr_c !=null) {
                        try {
                                result = (Number) expr_c.evaluate(variableObjects);
                        }
                        catch (Throwable e) {//Error message;}
                return ((Double) result).doubleValue();
        }

}//end of class
```

A.3 Limitations of using JEL

JEL uses a 'classloader', a privileged Java operation, to evaluate to compile the expressions. The implementation of this operation requires that JEL be used as a application. The Java security manager prohibits applets from successfully implementing privileged operations. Therefore, the use of JEL as an applet, requires that the source code be trusted. This means that web users of the GeGAOpt will have to grant permissions to the GeGAOpt code in the either of the following ways:

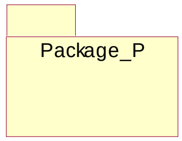
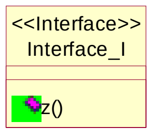
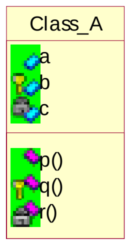
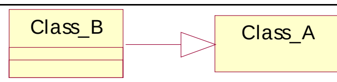
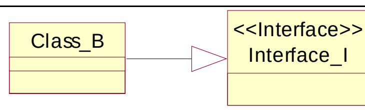
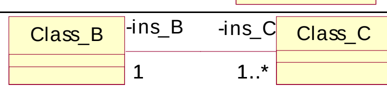
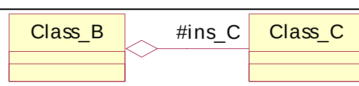
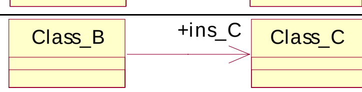
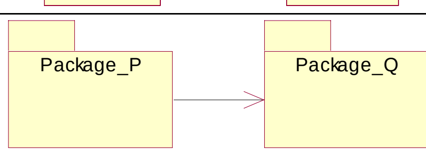
- By downloading the *.java.policy* and *.keystore* files from the GeGAOpt webpage at URL: http://www4.ncsu.edu/~avparand/GeneticAlgorithm/download
- By granting the code from the URL: http://www4.ncsu.edu/~avparand/GeneticAlgorithm/classes full permissions. These permissions are specified in the users' *.java.policy* file.

A detailed discussion of the Java security implications is outside the scope of this report. Interested readers should refer to the references at the end of this Appendix.

References

1. Metlov, C., 1998. *Java Expressions Library (JEL) Reference Manual*, URL: http://galaxy.fzu.cz/JEL.
2. `An example of using trusted applets.` URL: http://java.sun.com/security/signExample12/
3. The Java Plugin. URL: http://java.sun.com/products/plugin/1.2/
4. JDK 1.2 security model URL: http://www.javasoft.com/docs/books/tutorial/security1.2/index.html
5. Java Tutorials. URL: http://www.javasoft.com/docs/books/tutorial/

Appendix B

No	Visual Formalism	Represents	Notes
1		A *package* Also a *category*	
2		An *interface*	
3		A *class*	a, b, c are *attributes* of this class. p, q, r are *operations* of this class. **Visibility** a, b, c are public, protected and private *attributes* respectively.
4		An *inheritance relationship*	Class_B extends Class_A
5		An *implements relationship*	Class_B implements Interface_I
6		An *association*	ins_C is an instance of Class_C ins_B is an instance of Class_B '1' and '1..*' are the multiplicities. These imply that Class_C points to only one instance of Class_B, and Class_B points to multiple instances of Class_C
7		An *aggregation relationship*	Class_C is an aggregate of Class_B
8		A *uni-directional association*	Class_B points to Class_C by reference.
9		A *dependency*	Package_P depends on or needs visibility to Package_Q.
10	+, #, - notation in the associations	Indicates *access*	+ indicates a public access # indicates a protected access - indicates a private access