Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

A FRAMEWORK FOR DISTRIBUTED SIMULATION OPTIMIZATION

Björn Gehlsen
Bernd Page

Department for Informatics
University of Hamburg
Vogt-Kölln-Str. 30
D-22527 Hamburg, GERMANY

ABSTRACT

The system presented bridges the gap between three different research areas: discrete event simulation, heuristic optimization methods and distributed systems technology. Its goal is to provide a framework which supports an efficient implementation of simulation optimization projects, including heuristic optimum seeking procedures and parallel execution of experiments. It is written completely in Java and only uses components that are publicly available, including software libraries from academic institutions or the Java API from Sun Microsystems. The framework is well applicable for education and further research.

1 INTRODUCTION

Simulation projects often support the analysis of complex systems. Usually, some scenarios to be evaluated are determined before the simulation runs are started. When a simulation project aims at finding a good or even optimal configuration with respect to some quality measure, the need for an integration of simulation tools with optimization methods arises. One drawback of simulation optimization is the huge computational effort usually generated by such tasks. It results from the need to execute many simulation experiments, each typically causing a considerable machine load.

This problem is tackled by both using effective and fast optimization strategies and operating several simulation engines for experiment execution in parallel. The framework DISMO (DIstributed SiMulation Optimization), which is described in this article, shows a noncommercial approach to distributed simulation optimization. Open source developments from academic institutions are used for simulation and optimization. This allows for optimal integration of the framework's components. Additionally, it is of particular value as some commercial products require a separate license for every machine involved in a distributed simulation optimization project.

The paper is organized as follows: The next section introduces DESMO-J, a simulator developed at the University of Hamburg. In Section 3, the experimental frame and some foundations of simulation optimization are discussed. A concept for using Genetic Algorithms and Java's Remote Method Invocation (RMI) for simulation optimization purposes is developed in Sections 4 and 5 respectively. This leads to a system architecture which integrates the required tasks. It is presented in Section 6. Finally, we offer some conclusions and give a short outlook on further research.

2 A JAVA-BASED FRAMEWORK FOR DISCRETE EVENT SIMULATION

A framework for Discrete Event Simulation and MOdeling (DESMO) was developed at the University of Hamburg in the late 1980s. It has been continuously evolved over the last decade. Starting with Modula-2, the framework was ported to various programming languages including Smalltalk and C++. The latest version is DESMO-J, presented by Lechler and Page (1999). It is based on a fully object oriented architecture and is completely implemented in Java.

Current work on DESMO-J includes

- the implementation of dedicated classes for the simulation of production systems,
- an implementation of the object oriented DESMO architecture in Delphi and,
- the integration with a material networks approach for industrial environmental management systems.

As every DESMO-J model is a Java program, features of Java also apply to DESMO-J. The framework as well as implemented models are platform independent. Experiments were run successfully under Solaris, Linux, Windows NT/98/ME and MacOS.

Java is easy to learn and widely used; thus a modeler is most likely not concerned with learning a new pro-

gramming language. So fast and flexible model development compensates for Java's competitive disadvantage with respect to execution speed. Utilities like `javadoc` allow the generation of model documentation from the model builder's comments in the source code.

Furthermore, introspection as well as remote method invocation (RMI) is supported by the standard Java packages `java.lang.reflect` and `java.rmi` respectively. It will be shown later how these features can be used with DISMO for collecting information on model classes or to instantiate and execute experiments on remote machines. DESMO-J is used as a simulator, i.e. a system executing (remote) models, in the DISMO framework. In contrary to many commercial tools, DESMO-J does not require separate licences for remote servers as it is published under the GNU public license.

The source code and a Java archive can be found on the DESMO-J website (DESMO-J 2000) together with a complete API documentation and an extensive tutorial. For more details on DESMO-J the reader is referred to Page et al. (2000).

3 THE EXPERIMENTAL FRAME

A "specification of the conditions under which the system is observed or experimented with" is known as an *experimental frame* (Zeigler et al. 2000). It "operationally formulates the objectives that motivate a modeling and simulation project", which can roughly be split into the three phases of model building, experiment executions and output analysis.

Unfortunately, building a model which adequately represents a real or virtual system of interest still requires considerable effort. This task includes an analysis of the system, the model's implementation as well as its verification and validation. Realistic model runs additionally require a collection of adequate data. The work of Bachmann (2000) aims at easing the process of model building significantly by composing models from distributed and reusable modules which communicate on a CORBA-based platform independent architecture. In contrary, DISMO concentrates on the experimental phase assuming that an appropriate model is available to the simulation analyst. It could be chosen from a model base or built from existing modules, if available; but very often it has to be built newly to meet the project's requirements completely. Once a model has been built, these efforts are to be justified by profitable results obtained during multiple model runs.

The experimental phase of a simulation project or study usually consists of many model runs or experiments with different values set for the model's parameters. For statistical reasons, an experiment itself can be composed of many replications, varying only seed values for random number generators but keeping application specific parameter values fixed. Results of replication runs need to be aggregated to a single configuration's evaluation.

3.1 Model Parameterization

To start the experimental phase, DISMO lets the analyst initialize a chosen model by setting some values depending on the goals of the simulation study. These settings cannot be provided in advance by a model builder although default values might be given. They include the specification of which input variables to vary, which output measures to observe and how long to simulate.

As the number of possible parameter vectors grows exponentially with the number of variable input parameters, searching this space can be eased by keeping the latter small. It helps to consider which parameters do not need to be changed during the project's simulation runs. Some of the input parameters may be uncontrollable in the real system anyway. Thus their variation could not be implemented later and might not be worth consideration. In a different context it just might be the intention to examine the consequences of changing some (apparently) uncontrollable parameters and to examine whether a big effort might pay off.

Usually the model runs produce much more data than necessary. A choice of which responses are relevant and worth monitoring should be made before the experiments are started as well as how to aggregate or summarize model output. It might be necessary to specify start and termination points for single runs or to create batch means from long runs depending on the model and the project.

3.2 Replications of Model Runs

With every experiment, a single configuration of the real system is evaluated. It is represented by a vector of input parameter values. So the model can be viewed as a complex function mapping a specified system configuration to corresponding results.

Most simulation models contain stochastic elements, e.g. probability distributions. They may take random numbers as input values and deliver random output corresponding to the distribution's parameters specified in the model. Stochastic input values lead to uncertain, noisy output. Thus, one is no longer interested in a single evaluation of an input vector but the corresponding means, standard deviations, and confidence intervals of the results. This requires multiple replications of each configuration to be run in order to average out the output's stochastic behavior.

DISMO combines many replications to a model run evaluating a certain parameter vector. Each replication evaluates the same parameter values but differs from others in the initialization of their random number streams. The number of replications per run is left adjustable to the analyst.

The output is then aggregated and assessed by an objective function returning a scalar value. This guarantees that outputs of different model runs are comparable with respect to the quality defined within the project. For the time being DISMO simplifies the task of output analysis by simply averaging the objective values corresponding to the replication's outputs. A survey of more precise methods as well as further references on output analysis are presented by Kelton (1997 and 1999). DISMO hides replications from the user, who may only want to quantify the number of replications to observe for each configuration.

3.3 Experimental Design

The experimental design determines which parameter values to choose for the model runs at issue. The number of possible input parameter vectors grows exponentially with the number of input variables. Thus, an exhaustive search of the entire solution space is infeasible within reasonable time, at least for problems with practical relevance.

The system configurations to be evaluated should be chosen carefully and with the objectives of the simulation project in mind to protect a simulation analyst from unsystematic and ineffective "hit-or-miss" sequences of simulation runs. Chapter 12 of Law and Kelton (2000) lists some strategies to produce a mature plan of experiments. Their goal is to find a small but right fraction of all possible experiments to be run.

Executing model runs (experiments) according to the precedent experimental design generates corresponding output data which has to be aggregated and interpreted later. Then, useful information, which supports decision making, can be obtained from it, hopefully.

3.4 Simulation Optimization

A classical experimental design determines the system configurations to be evaluated *before* the simulation runs are started. Thus, knowledge about the structure or the behavior of the system, which is obtained during the experiments, does not influence the sequence of experiments. Unlike classical experimental design, optimization methods provide new and improved input parameter values more systematically. They can even take results of former model runs into consideration.

In contrast to evaluating a set of previously given configurations, a simulation project may aim at optimizing the simulated system, i.e. gathering certain values for the model's input parameters, which produce model outputs that maximize or minimize a corresponding performance measure. This prerequisite demands optimization tools for simulation projects. They are capable of finding one or more elements of a typically large solution space or search space which optimize a given objective function. Conceptually DISMO is designed to use any optimization method. The current prototype uses Genetic Algorithms. This heuristic optimization method suitable for use with simulation models will be presented in Section 4.

As shown in Figure 1, simulation and optimization are arranged cyclically. Starting with a given or randomly generated set of simulation input values, simulation and optimization alternate until some termination condition is met. An application specific objective function is used to rate the output corresponding to the simulation input. From a simulation analyst's point of view, the optimization method is wrapped around the simulation model; it receives the model's aggregated results, evaluates the output and generates new input values for the model.

From an optimization point of view, the model is seen as a black box evaluating a complex function which cannot be given in a closed form. It transforms the input vectors, each representing a different scenario, to outputs which are then compared using their objective function's values.

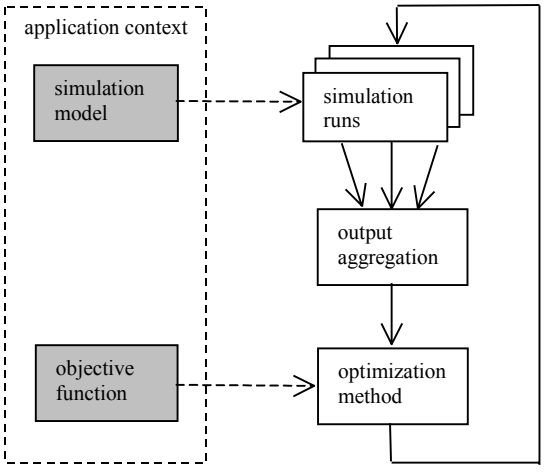

Figure 1: Simulation optimization cycle. Shaded objects are application specific and to be provided by the simulation analyst.

The optimization method itself can be customized by setting some technical parameters. In case of heuristic methods these parameters mainly specify the heuristic's accuracy. For Genetic Algorithms they include the population size, the number of generations observed and probabilities for applying the different genetic operators. Considering a simulation optimization project, the classical experimental frame is to be extended by the adjustment of these parameters.

4 GENETIC ALGORITHMS FOR SIMULATION OPTIMIZATION

In simulation models, analytical information about the structure of the search space is not available. Particularly, there are no derivatives. They could be used to find extrema of the objective function which could then serve as

indicators for optimal system configurations. Thus, optimization methods used in simulation projects have to do without this information. Genetic Algorithms (GA) belong to the class of direct search strategies. In a direct search every considered solution is rated using the objective function values only. Thus, no closed form of the problem and no further analytical information is required to direct the search process towards good or preferably optimal elements of the search space. The execution of a model run yields the evaluation data.

GA simulate the principles of biological evolution for the purpose of optimization. As a heuristic optimization method, GA do not guarantee to find optimum solutions, but they are looking for elements in a given search space which are good in terms of a specified measure. As only a fraction of the solution space is visited, GA finish after a reasonable time. Often, good solutions are found even in large spaces for which an exhaustive search would be infeasible anyway.

GA provide a general process model for heuristic optimization. We use GA in simulation projects to find input parameter values which result in a good or profitable behavior of the observed system in terms of the objective measure. The intention of the following subsections is to give an idea of GA being well suitable to guide the experimental design of a simulation project. More detailed literature on GA include Davis (1991) and Goldberg (1989). In the DISMO framework, the Java Distributed Evolutionary Algorithms Library (JDEAL), which is described by Costa et al. (2001), is used for implementing GA.

4.1 The Basic Genetic Algorithm

A Genetic Algorithm (GA) is an iterative, population-based, direct search strategy. The search space is usually given as the solution space of the application. In a simulation project, the solution space is the set of all controllable input parameter vectors for the model used, which correspond to feasible configurations of the real system under consideration.

A GA, i.e. its genetic operators do not work on the problem's solutions but rather on representatives. A suitable encoding maps solutions (phenotypes) to representatives (genotypes), which are often realized as linear bit-strings. Arrays of integers or floating-point numbers are other realizations of genotypes. By using representatives, the genetic operators (selection, crossover, mutation) which manipulate the genotypes can be defined and implemented problem independently.

During every iteration (generation) a GA is working on a set (population) of representatives of solutions (individuals). On the analogy of nature, in every generation the fittest members have the highest probability to produce offspring using their favorable characteristics. In every single generation of a GA, the individuals of the current population are *evaluated* and ranked according to their fitness. Then some of the individuals are *selected* to produce offspring by a selection operator which prefers individuals with higher fitness to guide the algorithm towards promising regions of the search space. Selected individuals are *reproduced* using a crossover operator (2 ancestors) and/or a mutation operator (1 ancestor) with a given probability. Finally, a new generation's population is built and (some of) the old individuals are *replaced* by the new offspring.

A number of technical parameters is used to control the behavior of a GA. They include population size, maximum number of generations, and probability measures for applying the different genetic operators. They adjust the balance between exploration of the whole space and exploitation of certain promising regions. Thus, these parameters determine the desired accuracy of the heuristic algorithm which usually conflicts with fast termination.

4.2 Customization of Genetic Algorithms for Use Within Simulation Projects

Generally, GA are domain independent and require no prior knowledge of the application context. Nevertheless, two main components of GA have to be specified depending on the application in order to be more effective than random search: the encoding of solutions (a mapping from the decision space to the individual space) and the objective function (Szczerbicka et al. 1998). So a GA can be customized for parameter optimization of simulation models by implementing these components specifically.

The objective function connects knowledge on the application domain and the encoding. The encoding should be chosen regarding the application context, e.g. similar solutions to the observed system should be represented by similar individuals. The objective function reflects the subjective ratings governing the simulation project and so effects the calculation of an individual's fitness. Thus, in DISMO the objective is provided with the model or implemented by the simulation analyst with respect to the goals of the current project before the experiment runs are started.

Principally, a GA is application independent. Michalewicz (1996) poses some guiding questions for the implementation of GA concerning the feasibility of interim solutions, which are worth consideration. A GA's genetic operators may produce individuals which are infeasible, e.g. whose corresponding solutions are not valid in the application context. As these solutions' offspring may be feasible again, new paths through infeasible regions of the search space are discovered this way and may lead to feasible regions which would remain undiscovered otherwise. In a GA for the parameter optimization of simulation models, infeasible solutions should be kept off from evaluation by simulation as this would waste valuable computation

time. Genetic operators can also be implemented to generally avoid the generation of infeasible solutions.

4.3 Parallel Genetic Algorithms

The population-based nature of GA offers a straightforward approach for parallelism. The individual's evaluations are categorically independent of each other. When applying GA within a simulation project, an individual represents a vector of concrete input parameter values and an individual's evaluation corresponds to a run or a set of replication runs of the underlying model with its parameter values. Thus paralleling independent and time consuming tasks promises to be a substantial benefit to the overall system performance.

In the literature, different approaches of parallelism in GA are reviewed. Global population models consider the whole population and parallel crossover and evaluation. Coarse grain models or island models build distinct subpopulations which are observed separately on different processors, and allow migration of individuals in a limited number only. Last, fine grain models, or cellular GA, which can be shown to be a subclass of cellular automata assign a separate processor to each individual and restrict genetic operation to neighboring individuals (Whitley 2000).

The framework described aims at supporting models built by different persons who might not consider during implementation that later the model runs will be executed remotely. Therefore, in DISMO, the evaluation of individuals, i.e. the simulation runs, are executed on a remote server, but one global population is controlled on a single client machine. The corresponding class (`Experiment-Manager`) is described in Section 6.

5 REMOTE EXECUTION OF SIMULATION RUNS

Simulation experiments are complex tasks which require considerable resources even on modern computer systems. When multiple simulation runs are essential for a simulation project, the delegation of experiment execution to a number of remote machines is very paying. As communication overhead can be neglected among independent model runs, a near-linear speedup can be achieved.

5.1 Encapsulated Experiments

For the purpose of experiment delegation, a coordinating machine needs to give its servers a precise instruction of their duty. A complete description of a model run including input parameter values and a reference to the model class is encapsulated in the class `SimTask` (see Section 6).

In each iteration, the simulation client constructs `SimTask` objects for every individual of the current population as well as their replications. It then passes them to the remote machines and receives the corresponding simulation results after the remote model run has finished.

5.2 Deployment of Simulation Tasks

In each generation of the optimization process, the population represents a set of independent experiments. Therefore, a set of remote machines is prepared to accept requests for executing model runs. A simple server program is located on all available remote machines. Upon activation, this server process registers itself as the 'DISMO-Service' with the registry of its machine and can then be reached from clients supplying the servers with simulation tasks to be executed.

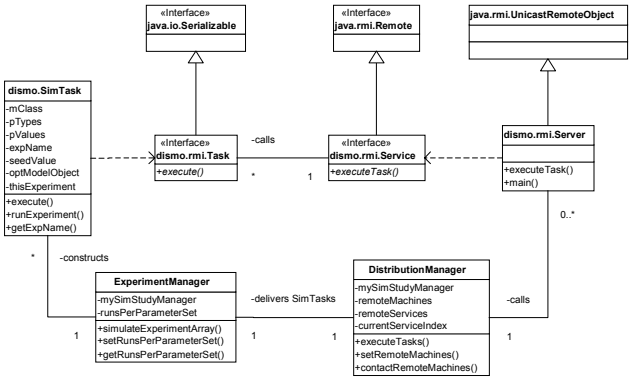

Figure 2: UML diagram of RMI related classes of DISMO

A local machine (simulation client) coordinates the construction of model run descriptions which are passed to the remote servers as a parameter of a remote method call. Favourably, every single run is allocated to its own remote machine. Even parallel execution of independent simulation runs on a single machine is possible, as DISMO organizes every single run into a separate Java thread.

5.3 Remote Method Invocation

In the DISMO system, Java Remote Method Invocation (RMI) is used to spread the current population of the GA. RMI is a core Java API and class library. It supports the implementation of remote objects. Methods on remote objects can be called by local objects as if they were residing on the local machine. In the same way, arguments can be passed on to the remote servers and return values are received (Figure 2).

To invoke methods on remote objects, a reference to the remote object is required. These so called remote references are stored in a registry, a small program running on the server's machine. The registry can be asked for the remote reference by the name the remote object is registered with. More technical details on RMI can be found in Har-

old (2000) as well as in the Java API documentation of the package `java.rmi` (Sun Microsystems 2001).

It is planned to include a Lightweight Directory Access Server (LDAP) to control the registered server machines and to make them known to clients requesting any DISMO service. Unlike in the current version, also tasks for generating new parameter values and fitness evaluation could be executed remotely.

6 SYSTEM ARCHITECTURE

The framework DISMO presented in this contribution aimes at optimizing systems which are evaluated through distributed runs of a discrete simulation model. Therefore, the framework involves a simulator, optimization methods and a distribution mechanism, all controlled by a simulation study manager (Figure 3). It assumes that an appropriate model and an objective function are supplied by the simulation analyst.

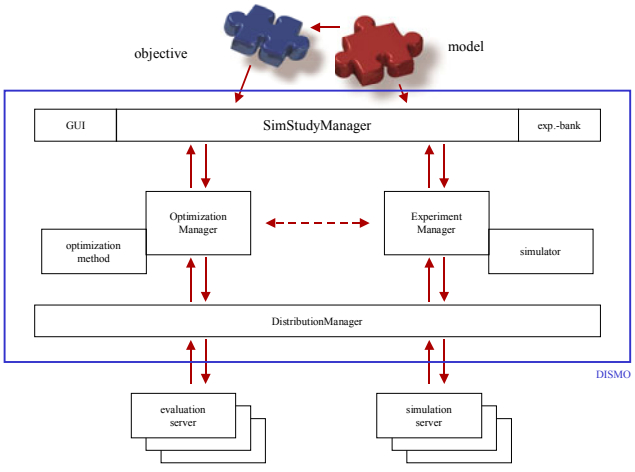

Figure 3: DISMO system architecture

6.1 Simulation Tasks

Conceptually, DISMO separates a simulation model from single experiment runs. The notion of a simulation task is introduced to encapsulate concrete parameter values and a model specification. It is implemented as a Java class `SimTask` and provides a method `execute()`, that can be activated on a remote machine. For that purpose a `SimTask` object is passed to a remote server as an argument of a remote method call.

6.2 Model Requirements

The execution of model runs is to be initialized with different parameter values which have to be provided independently from the model. Therefore, the model is supposed to implement the interface `Executable` which allows introspection of the model for a calling program (`SimStudyManager`). So the types of the model's input variables can be exported as well as default values or feasibility intervals for it's values, where appropriate. No concrete values for parameters (except defaults) should be implemented in the model itself.

The interface `Executable` enforces a model to provide methods which inform the `SimStudyManager` about properties of the model, particularly about

- the model's name including a short description,
- the name, type, default value, lower and upper bound of the model's parameters,
- whether a parameter's value is variable or constant,
- the model's result types,
- the model's result values.

These details are required to initialize the `Experiment-` and `OptimizationManager` as well as to check compatibility with the objective function which implements a method `evaluate()` taking the model's aggregated result values as input parameters.

6.3 Distributed Simulation Optimization with DISMO

Before any experiment is executed, the simulation study manager (`SimStudyManager`) lets the analyst define a model and the objective function to use during the project. Additionally it collects some required technical parameters (e.g. population size, probabilities for genetic operators, maximal iteration number and number of replications per simulation run) as well as model specific parameters. Finally it constructs an `ExperimentManager`, an `OptimizationManager` and a `DistributionManager` for the current simulation project.

The `OptimizationManager` starts to generate the first simulation input parameters. The `ExperimentManager` constructs `SimTask` objects and passes them to the `DistributionManager`. The `DistributionManager` maintains a list of the available remote services each of which represents a server process on a remote machine. It deploys `SimTask` objects as arguments of an RMI call to remote machines for execution. After completion the simulation results are returned to the `OptimizationManager` where evaluation, selection, recombination and replacement result in a new set of parameter values which will be used as simulation input for the next iteration.

7 CONCLUSIONS AND OUTLOOK

The DISMO Framework successfully implements the functionality of automatic instantiation and execution of simulation experiments. Optimization methods are used to find good values for the simulation model's parameters systematically. DISMO also allows the resulting load to be paralleled on

available machines across a network, so an almost linear speed-up can be achieved by using a LAN of workstations which often have some free capacity. Remote execution of simulation experiments are realized by using standard Java components, from the `java.lang.reflect` and `java.rmi` API, available from Sun Microsystems (Sun Microsystems 2001).

DISMO is an academic framework. Future extensions include a visualization component. For real world applications of the framework a container terminal simulation in cooperation with the port of Hamburg is under consideration.

ACKNOWLEDGMENTS

Ruth Meyer deserves special thanks for interesting discussions and valuable ideas. She also made several suggestions to improve the accuracy and clarity of this paper. The research presented is partially based on the work of several students involved in the development of DESMO, particularly Tim Lechler, Olaf Neidhardt and Sönke Claassen, who recently developed, evolved and documented DESMO-J.

REFERENCES

Bachmann, R. 2000. HLA und CORBA: Partner oder Konkurrenten? In *Simulationstechnik, 14. Symposium in Hamburg*, (*in German*), ed. D. Möller, 141-146. Erlangen: European Publishing House.

Costa, J., N. Lopes, and P. Silva. 2001. JDEAL. The Java Distributed Evolutionary Algorithms Library [online]. Available online via <`http://laseeb.ist.utl.pt/sw/jdeal/`> [accessed March 16, 2001].

Davis, L. D. 1991. Handbook of Genetic Algorithms. New York: Van Nostrand Reinhold.

DESMO-J. 2000. The DESMO-J homepage [online]. Available online via <`http://www.desmoj.de`> [accessed July 4, 2001].

Goldberg, D. E. 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Reading. Addison Wesley.

Harold, E. R. 2000. Java Network Programming, 2nd Edition. Sebastopol, CA: O'Reilly&Associates.

Kelton, W.D. 1997. Statistical analysis of simulation output. In *Proceedings of the 1997 Winter Simulation Conferenc*e, ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, 23–30.

Kelton, W.D. 1999. Designing Simulation Experiments. In *Proceedings of the 1999 Winter Simulation Conferenc*e, ed. P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, 33–38.

Law, A. M., W. D. Kelton. 2000. Simulation Modeling and Analysis, 3rd ed. Boston: McGraw Hill.

Lechler, T. and B. Page. 1999. DESMO-J : An Object Oriented Discrete Simulation Framework in Java. In *Proceedings of the 11th European Simulation Symposium*, ed. G. Horton, D. Möller, and U. Rüde, 46-50. Erlangen: SCS Publishing House.

Michalewicz, Z. 1996. Evolutionary Computation and Heuristics. In *Meta-Heuristics: Theory & Applications*, ed. I. H. Osman, and J. P. Kelly, 37-52. Norwell: Kluver Academic Publishers.

Page, B., T. Lechler, and S. Claassen. 2000. Objektorientierte Simulation mit dem Framework DESMO-J (*in German*). Hamburg: Libri Books on Demand.

Sun Microsystems. 2001. The Source for Java Technology. [online]. Available online via <`http://java.sun.com`> [accessed March 21, 2001].

Szczerbicka, H., M. Becker, M. Syrjakow. 1998. Genetic Algorithms: A Tool for Modelling, Simulation, And Optimization of Complex Systems. In *Cybernetics and Systems: An International Journal*, Vol. 29, 639-659.

Whitley, D. 2000. A Genetic Algorithm Tutorial. [online]. Available online via <`http://samizdat.mines.edu/ga_tutorial`> [accessed September 4, 2000].

Zeigler, B., H. Praehofer, and T. G. Kim. 2000. *Theory of Modeling and Simulation, 2nd Edition*. San Diego: Academic Press.

AUTHOR BIOGRAPHIES

BJÖRN GEHLSEN is a scientific assistant and Ph.D. candidate at the Department for Informatics at the University of Hamburg from where he received his M.S. degree in Computer Science. He focused on operations research and heuristic optimization methods for NP-hard problems and worked on different research projects on traffic simulation and on concepts for accessing distributed databases. His current research includes the integration of simulation tools with optimization methods, using distributed system technology. He is a member of the ACM, the German Informatics Society (Gesellschaft für Informatik) and the German speaking simulation society ASIM. His e-mail address is <`gehlsen@informatik.uni-hamburg.de`>.

BERND PAGE is a professor for applied computer science at the University of Hamburg where he leads the simulation group. He received his Ph.D. from the Technical University of Berlin and also holds a M.S. from Stanford University. Before his appointment at the Department of Computer Science at the University of Hamburg in 1984 he was working as a scientific associate in the Environmental Information System Group at the German Federal Environmental Agency in Berlin. His research interests include discrete event simulation software and programming techniques as well as Environmental Informatics. Dr. Page is the author of a well recognized German textbook on discrete event system simulation and more recently of a book on object oriented simulation in Java. His email address is <`page@informatik.uni-hamburg.de`>.