

Typeset with N_DThesis version 2.14 (2000/09/08)
on May 9, 2003
for
Xiaorong Xiang
entitled

AGENT-BASED SCIENTIFIC APPLICATIONS AND COLLABORATION
USING JAVA

This class conforms to the University of Notre Dame style guidelines established Fall 2000. However it is still possible to generate a non-conformant document if the published instructions are not followed! Be sure to refer to the published Graduate School guidelines as well.

This summary page can be disabled by specifying the `nosummary` option to the class invocation. (i.e., `\documentclass[nosummary]{ndthesis}`)

**THIS PAGE IS NOT PART OF THE THESIS, BUT
SHOULD BE TURNED IN TO THE PROOFREADER!**

N_DThesis documentation can be found at these locations:

<http://www.nd.edu/~afsunix/faq/tetexdoc/latex/ndthesis/>
<http://www.gsu.nd.edu/Committees/ITC/ndthesis.pdf>
http://www.gsu.nd.edu/Committees/ITC/sample_ndthesis.tar.gz

General L^AT_EX documentation and info:

On-line docs:

ND installation <http://www.nd.edu/~afsunix/faq/tetexdoc/>
T_EX User's Group <http://www.tug.org/>

Books:

A Guide... for Beg. & Adv. Users by Kopka/Daly
L^AT_EX User's Guide ... by Lamport
The L^AT_EX Companion by Goossens/Mittelbach/Samarin

Packages: (check on-line docs)

rotating sideways tables and figures
longtable multi-page tables
graphicx using Postscript and other figures

AGENT-BASED SCIENTIFIC APPLICATIONS AND COLLABORATION
USING JAVA

A Thesis

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science

by

Xiaorong Xiang, B.S.

Dr. Kevin Bowyer, Director

Department of Computer Science and Engineering

Notre Dame, Indiana

May 2003

ACKNOWLEDGEMENTS

The thesis presents my work at the Computer Science and Engineering Department of the University of Notre Dame. I am very grateful to Dr. Gregory R. Madey, who gave me encouragement and crucial guidance to my research. I would like to thank Dr. Kevin Bowyer for his kind support. I would also like to thank Dr. Matthias Scheutz for serving on my defense committee.

I would like to thank two members of the NOM research group, Yingping Huang and Eric Chanowich, for their contributions of this project. I would also like to thank Dr. Steve Cabaniss for his formulation of the NOM problem.

I would like to thank Dr. Patricia Maurice and Leilani Arthurs for testing our simulation model and their valuable feedbacks. I would also like to thank Leilani Arthurs for proofreading my thesis.

I would like to express my gratitude to my parents, my husband and my son for their encouragement, emotional support, and love.

This research has been supported in part by information technology research (ITR) program of the NSF.

AGENT-BASED SCIENTIFIC APPLICATIONS AND COLLABORATION
USING JAVA

Abstract

by

Xiaorong Xiang

Natural organic matter (NOM), a heterogeneous mixture of molecules, plays a crucial role in the evolution of soils, the transport of pollutants, and the change of global weather. The evolution of NOM over time is an important research area in biology, geochemistry, ecology, soil science, and water resource. Due to its complexity and the structural heterogeneity, new simulation approaches are needed to help to better understand the structure and the evolution of NOM. We present a new stochastic model which explicitly treats NOM as a large number of discrete heterogeneous molecules. The NOM, micro-organisms, and their environment are taken together as a complex system, and simulated using an agent-based modeling approach. The global properties of NOM evolution over time can be studied by simulating the physical and chemical reactions between individual agents with temporal and spatial properties.

Unlike the previous stand-alone simulation models, the NOM simulation model serves as an example of E-science, in which we do science on the Web by combining recent information technologies with a computational approach. An intelligent Web-based interface is developed to allow scientists to access the remote simulation model from a standard Web browser. The Web-based interface enables scientists to remotely provide parameters for their simulations, start and stop the simulations,

and view the results.

The initial users of the NOM simulation model includes a geographically separated group of NSF sponsored scientists from different research areas. A prototype of a NOM "collaboratory" is built to promote collaboration among these scientists and allow them to share their data and information across distributed sites. A XML-based Markup Language, NOML, is provided to build the XML-based Web components and facilitate Web services development in the future.

Performance is a prime concern for implementing scientific applications using the Java programming language. In this thesis, several aspects of the NOM simulation model have been analyzed for improving performance and scalability. These aspects include runtime optimization, database access, object usage, and parallel and distributed computing. This research demonstrates that high-performance Java applications can be achieved by profiling and proper design.

This is for my parents, my husband, and my son.

CONTENTS

ACKNOWLEDGEMENTS	i
TABLES	vi
FIGURES	vii
CHAPTER 1: INTRODUCTION	1
1.1 Introduction to E-science	2
1.2 What is a complex system?	2
1.3 Introduction to agent-based modeling	3
1.4 What is Natural Organic Matter (NOM)?	5
1.5 Main contributions of the thesis	5
1.6 Organization of the thesis	7
CHAPTER 2: RELATED WORKS AND TECHNOLOGIES	8
2.1 Simulation models	9
2.1.1 Deterministic models vs. Stochastic models	9
2.1.2 Stochastic model	10
2.2 An agent-based stochastic model	11
2.3 A Web-based scientific application	12
2.4 Agent-based simulation toolkits	13
2.5 Technologies for building scientific application model	15
2.5.1 Introduction to Java programming language and J2EE architecture	15
2.5.2 Oracle database overview	18
2.5.3 Application server overview	19
2.6 Summary	20
CHAPTER 3: A SIMULATION MODEL FOR A COMPLEX SYSTEM	22
3.1 Introduction	22
3.2 NOM simulation model overview	23
3.2.1 Access the NOM model from web	25
3.3 Core simulation engine description	25

3.3.1	Simulation algorithm	26
3.3.2	Reaction probabilities calculation	28
3.3.3	Molecule objects	29
3.3.4	Reactions and Processes	30
3.3.5	Environmental parameters	32
3.3.6	Molecular Properties	32
3.3.7	Pseudo-random number generator	33
3.3.8	Program description	33
3.3.9	Input and Output	34
3.4	Intelligent agent components	35
3.4.1	Model-view-controller Architecture	37
3.4.2	Database design	38
3.4.3	Web Interface Layout	40
3.4.4	Intelligent agent components	41
3.5	Data analysis	44
3.6	Conclusion	44

CHAPTER 4: EXPLORING PERFORMANCE IMPROVEMENT FOR JAVA-BASED SCIENTIFIC APPLICATIONS	46
4.1 Introduction	46
4.2 Profiling tool	48
4.3 Data Structure	48
4.4 Object reuse	51
4.5 Database connection and database query	54
4.6 Parallel data output with Java threads	58
4.7 Choosing JVM	59
4.8 High performance compilers	60
4.9 Scalability	62
4.9.1 Parallel implementations	64
4.9.2 Performance results	66
4.10 Conclusion	68

CHAPTER 5: A COLLABORATORY BUILT UPON J2EE PLATFORM FOR SUPPORTING SCIENTIFIC COLLABORATIONS	71
5.1 Introduction	71
5.2 The NOM Collaboratory overview	73
5.3 XML-based NOM Markup Language (NOML)	76
5.4 Components design and capabilities	80
5.4.1 Search engine	80
5.4.2 NOML file uploader	81
5.4.3 Molecule editor	83
5.4.4 Molecule validator	84
5.4.5 Chat room and Discussion board	84
5.4.6 File sharing	85
5.5 Conclusion	85

CHAPTER 6: CONCLUSION AND FUTURE WORK	87
6.1 Conclusion	87
6.2 Future work	89
APPENDIX A: GLOSSARY	91
APPENDIX B: NOML	92
APPENDIX C: Screen capture and benchmark results	96
C.1 OptimizeIt	96
C.2 Java Grande benchmark	96
C.3 Design for Java threads version and MPJ version	98
C.4 Sequential model results	99
APPENDIX D: Molecular structures and chemical reactions	101
D.1 Molecular structures	101
D.2 Chemical reactions	101
BIBLIOGRAPHY	105

TABLES

3.1	Chemical reaction types	31
4.1	Summary of the performance improvements	69
C.1	Benchmark results for GCJ compiler and Java Hotspot virtual machine.	98

FIGURES

2.1	The J2EE environment (adapted from Nicholas (2000) [1])	18
3.1	The architecture of NOM simulation Model	24
3.2	A flow chart of the algorithm for the NOM simulation model	27
3.3	Probabilities	29
3.4	Class Diagram	34
3.5	Input and Output	35
3.6	The implementation model for the Web interface	37
3.7	Diagram for database design	39
3.8	Interface configuration steps	40
4.1	Performance comparison of different operations for <code>ArrayList</code> and <code>LinkedList</code> , the simulation runs for 500 time steps. Left: grid size is 400 X 300. Right: grid size is 800 X 300	50
4.2	Comparison of data insertions using five approaches in NOM simula- tion model	57
4.3	Overlap the computation and I/O using Java threads. Left figure shows that the number of data insertion over the time steps. Right figure shows that the speedup of using the second thread for data output	59
4.4	Performance comparison for a NOM application run in Sun Client VM and Sun Server VM with different grid sizes.	61
4.5	Compare the performance between sequential programming model and Java thread model in the NOM application (one thread vs. two threads on a single dual CPU computer).	66
4.6	Comparison of the performance between sequential programming model and MPJ model that runs on 4 machines for 1500 time steps.	67

4.7	Comparison of the performance between sequential programming model and MPJ model on 4 machines without synchronization and molecule exchange.	68
5.1	Components of the NOM collaboratory environment for supporting cooperation between remotely located scientists	75
5.2	Tree structured view of three types of documents in NOML format. Note: not all the elements are shown in the graph.	79
5.3	The design model for NOML file uploader	82
5.4	The benzene ring structure	83
C.1	OptimizeIt provides a tree structure view of the CPU usage while the application is running, and identify the HotSpots.	96
C.2	OptimizeIt provides the number of each type of objects that are allocated in the heap and the deallocation information; it also shows the object size.	97
C.3	OptimizeIt provides the graph that shows the heap usage, the activation of garbage collector, the thread activity, and the class load information.	97
C.4	The design for parallelism of NOM simulation model using Java threads.	99
C.5	The design for parallelism of NOM simulation model using MPJ.	100
C.6	The behavior of sequential model when time step and grid size increase.	100
D.1	The structure of cellulose	101
D.2	The structure of lignin	101
D.3	The description of protein	102
D.4	Ester Condensation	102
D.5	Ester Hydrolysis	102
D.6	Amide Hydrolysis	103
D.7	Microbial uptake	103
D.8	Dehydration	103
D.9	Strong C=C oxidation	103
D.10	Mild C=C oxidation	104
D.11	Alcohol (C-O-H) oxidation	104

D.12 Aldehyde C=O oxidation	104
D.13 Decarboxylation	104

CHAPTER 1

INTRODUCTION

Traditionally, the two main ways that scientific study is advanced are by theoretical and experimental approaches. In 1953, however, a third and more revolutionary approach to doing science emerged – the computational approach. The concept of a “computer experiment” was introduced by Fermi and his colleagues in 1953 and computers began being used to study physical phenomena. Employing a computational approach allows scientists to explore some scientific phenomena which may be too large and expensive to be handled with an experimental approach, or too difficult and complex to be analyzed by a theoretical approach [2].

Nowadays, advanced Internet and computer technologies provide scientific study the advantages of a computational approach over traditional theoretical and experimental approaches. The computer simulation is one of the methods in the computational approach. Computer simulations are intended to model real world phenomena in a variety of scientific systems in the biological, physical, geochemical, ecological, and social sciences. They provide scientists a flexible testbed for theoretical hypothesis because models can be tested with any number of input parameters under different situations. Additionally, computer simulations allow scientists to view the evolution of a system over millions of years or over seconds. Computer simulations also offer scientists a way to model phenomena, such as stock market changes and robot navigation in space, which are impossible or expensive to study using an

experimental approach.

1.1 Introduction to E-science

The advent of the Internet and advanced digital computer technology has produced a new generation of Web-based applications [3]. Web-based applications use the Web as a new platform to do science by combining recent information technologies and a computational approach. Nowadays, the widespread availability of the high-speed network and middle-ware technologies, such as Sun Microsystem's J2EE (Java 2 Platform, Enterprise Edition) [4] and Microsoft's .Net [5] make it possible to use Web and network as a platform for scientific applications. This new infrastructure, called E-Science in this thesis, has been used to solve problems such as distributed physical or astronomy data analysis [6], and remote access of the information source and simulation [7][8]. Web-based applications are intended to facilitate the use of the computational resources located in different physical sites, thereby allowing users at different locations to easily access information and communicate with each other.

1.2 What is a complex system?

A complex system is defined as a large-scale decentralized system with "emergent properties", few constraints, and simplifying assumptions. A complex system consists of multiple heterogeneous agents. Each agent is an individual component that pursues individual strategies resulting in self-organization without central regulation. The characteristics of individual agents can change as the agents adapt to their environment or interact with each other [9].

The dynamic changes of a complex system over time are usually nonlinear and sometimes even chaotic. It is impossible to analyze and understand a complex sys-

tem using traditional mathematical or theoretical methods. Computer simulations, however, can model the heterogeneous features and interactions of the agents in a complex system. The global phenomena of a complex system often can be studied by simulating the dynamic behavior of individual components and their interactions in the system. Analysis of the dynamics and emergent behavior of these simulation models help us understand how the actual system works.

The study of complex systems is widely applied in social, biological, physical, and computer sciences. For example, the center for the Study of Complex System [9] at the University of Michigan and the Santa Fe Institute [10] focus on the study of complex systems.

1.3 Introduction to agent-based modeling

Agent-based modeling (ABM), also known as individual-based modeling (IBM), is a method used to track the actions of multiple agents that can be defined as objects with some type of autonomous behavior [11]. This approach is used to simulate decentralized systems where no one part of any system is in apparent control. Reynolds (1987) [12] modeled flocks of birds as a large number of autonomous agents. Each bird was assigned a few simple rules to maintain a safe distance from other birds and to maintain a certain speed of flight. The results of Reynolds's simulation showed that the behavior of the modeled flock was consistent with the natural behavior of bird flocks. By using the ABM approach, the higher level behaviors of a system, called "emergent behaviors" of the system, can be discovered without being explicitly coded into the simulation. The technique of building and using ABMs is an essential tool for understanding complex systems and is increasingly applied to ecological, economic, and sociological issues.

An alternative approach to modeling a system is equation-based modeling (EBM),

a process for solving a set of differential equations. Van Dyke Parunak (1998) [13] working in the domain of supply networks, presented an agent-based model and an equation-based model for modeling the supply network, and concluded that an agent-based model is more suitable for modeling complex systems that are composed of interacting components and exhibit a wide range of dynamic behavior.

EBMs can describe already known global properties of a system, but often can neither explain the origin of those properties nor track the behavior of individual components. ABMs typically consist of an environment in which interactions occur and some number of individuals are defined in terms of their behaviors and characteristic parameters. ABMs and EBMs are significantly different with respect to which characteristics they focus on. Agent-based models focus on the characteristics of each individual and track them through time. Equation-based models, on the other hand, focus on the characteristics of the population, which are averaged, and simulate changes in the averaged population characteristics [13].

In some agent-based models, the individuals are associated with a location in geometrical space, and some individuals (animals, cars) can move around their environment while others (plants, computers) have no mobility at all. The geometrical space in these models can be described as either continuous space, represented by partial differential equations, or as discrete grid space, represented by integer values [14].

The objectives of agent-based models include observing individual agents with various attributes in a system, describing the heterogeneous aspects of an environment with spatial and temporal information, providing a mechanism for interactions between individual agents, and predicting phenomena at higher levels based on the actions of individual agents, in a bottom-up approach.

1.4 What is Natural Organic Matter (NOM)?

Natural organic matter (NOM) is a mixture of molecular compounds with different types of structure, composition, functional group concentration, molecular weight, and reactivity. NOM comes from animal and plant material in the natural environment. It exists everywhere in the world, from terrestrial ecosystems to aquatic environments. NOM plays a crucial role in ecological and bio-geochemical processes such as the evolution of soils, the transport of pollutants, and the global biochemical and geochemical cycling of elements [15]. The evolution of NOM over time from precursor molecules to mineralization is an important research area in a wide range of disciplines, including biology, geochemistry, ecology, soil science and water resources. NOM, a prevalent constituent of natural waters, is highly reactive with mineral surfaces [15]. While NOM is transported through soil pores by water, it can be adsorbed onto or desorbed from mineral surfaces. Sorption of NOM is an important consideration in the treatment of drinking water.

NOM is present in all surface and soil waters. The amount and the composition of NOM differs with climate and physical location, as well as a number of other environmental factors in different places. NOM represents a significant fraction of the solute present in fresh water and influences all chemical and biological processes in the aquatic environment [16].

1.5 Main contributions of the thesis

In previous models for NOM study, EPIC (1985)[17], Daisy (1990)[18], Brown (1999)[19] and Gu (1995) [20], NOM is treated as a single “organic carbon” entity, and the “average” value of properties is used to represent the complex NOM mixture. Cabaniss (2002) [15] presents several examples showing that the use of “average” values to represent the complexity of NOM is problematic. The use of average

values in modeling NOM can result in discrepancies when the NOM model results are compared with the results from the laboratory studies.

In this thesis, we present a model which explicitly treats NOM as a large number of discrete heterogeneous molecules. The NOM, micro-organisms, and their environment are taken together as a complex system, and simulated with an agent-based model. Individual components (molecules), modeled as individual agents, are given a set of simple rules on how to move through soil pores and how to interact with each other. The changes in individual components or a group of agents, which start with the same properties over time, can be tracked. The global properties (including distributions of physical, chemical, and biological properties) of NOM evolution over time can be predicted by simulating the physical and chemical reactions between individual agents with temporal and spatial properties.

The main contributions of this thesis are summarized as follows:

- Build a NOM simulator using agent-based modeling approach. The NOM transforms over time in response to biological and non-biological reactions including adsorption, desorption, and physical transport. NOM transformations are simulated using the Java programming language [21] and the Repast [22] and Swarm [23] [24] toolkits.
- Provide an intelligent web interface so that scientists can access the NOM simulator using a standard Web browser from remote sites and to assist scientists with entering reasonable parameters in their simulations. The different configurations represent the different environmental factors or different soil samples.
- The NOM project involves scientists from different locations and involved in different areas of research. A web-based scientific NOM collaboratory has been

built in order to support communication and cooperation between scientists involved in NOM research. The XML-based (Extensible Markup Language) NOM Markup language (NOML) is developed to support these component-based Web services.

- Building a scientific simulation with high performance, scalability, and stability is crucial for a scientific simulation written in Java programming language. From a software engineering perspective, several approaches to improve the performance and scalability of the NOM model are explored.

The objective of the NOM project is to provide scientists a tool which can be applied to aquatic ecosystem studies, soil science, crop science, environmental protection, remediation of surface and sub-surface waters, and global climate change prediction [15]. Application of the NOM model is also discussed as a case study for doing science on the Web.

1.6 Organization of the thesis

This thesis consists of six chapters in total. Chapter 1 provides an introduction to the computational approach for scientific study, an introduction to natural organic matter, and an overview of this thesis. In chapter 2, works related to the study of NOM and current computer technologies used in simulation modeling are discussed. The core simulation engine and a Web-based intelligent interface is discussed in chapter 3. In chapter 4, several approaches for improving the performance and scalability of the NOM simulation model are described. The prototype of the NOM collaboratory, which used to support collaboration between scientists, is presented in chapter 5. Some conclusions are drawn and future work is described in chapter 6.

CHAPTER 2

RELATED WORKS AND TECHNOLOGIES

The evolution of NOM over time from its biological precursor compounds plays a crucial role in the bio-geochemical process and the predictive environmental modeling. Understanding both the structural heterogeneity and the evolution of NOM can help scientists to predict the outcomes of environmental processes. However, due to the complexity and the structural heterogeneity of the NOM mixture, several existing models for NOM study are not suitable for scientists to better understand the structure and the evolution of NOM. Predicting the consequences of these environmental processes requires a clear understanding how a single NOM sample behaves and how the NOM mixture evolves in time and space.

Cabaniss [15] suggests that current models of NOM study can be classified into two categories: carbon cycling models, based on average properties of various organic carbon pools, and molecular models that employ connectivity maps or electron density. Cabaniss states that:

These models are either too simplistic to represent the heterogeneous structure of NOM and its complex behavior in the environment, or too compute intensive to be useful for large-scale environment simulations [15].

The evolution of NOM including transport, adsorption, desorption, and chemical reactions, is a dynamic process that consists of the successive states of the NOM complex system. This process can be modeled using a stochastic model or a deterministic model.

2.1 Simulation models

Simulation models can be classified in their progression of time, their state space and the evolution of the system [25][26].

Although using a continuous time domain is a natural choice for representing temporal relationships, it is preferable to model a system in terms of discrete time steps due to the following reasons: 1) a discrete time model is simpler than a continuous time model; 2) the computer can only perform calculations at discrete points; and 3) experimental methods measure data at discrete time points.

State space refers to the collection of all the possible states of an given system. It can be continuous or discrete.

For a deterministic process, the state of the system at one time completely specifies the system for all times. The evolution of the system over time is often represented by ordinary differential equations (ODE). In a stochastic process, the state of the system is represented by a set of values with a certain probability distribution, such that the evolution of the system is dependent on a series of probabilistic discrete events. Due to these differences, deterministic models are more easily implemented for mathematical analysis than stochastic models.

2.1.1 Deterministic models vs. Stochastic models

Deterministic models tend to use the continuous time and a deterministic process to simulate the state changes of a system. The evolution of these models is represented as explicit, first order, ordinary differential equations. Several simulators have been developed to simulate biochemical reactions using a deterministic model. Scamp (1993) [27] used non-standard biochemical languages to specify a model in a series of command files. These files are used to construct differential equations, then these equations are solved using numerical integration. MIST (1995) [28] uses

a graphical user interface to allow the user to enter the chemical equations, then convert the input into differential equations and solve them.

A stochastic model is a probabilistic model of a system that evolves randomly in time and space. It can use a continuous or discrete time domain. Physically, chemical reactions are stochastic events that involve a discrete number of particles with a spatial distribution. Deterministic models assume that the system is uniform and that elements in the system represent an average nature and behaviors. As such, deterministic models are not sufficient for modeling a chemical system with a discrete or stochastic nature, nor are they sufficient for predicting the concentrations of chemical species.

Thus, a stochastic model is more realistic and suitable for simulating chemical reactions in many biological systems with discrete and heterogeneous properties. When the dynamics of a system is too complex, the stochastic model dominates because the deterministic representation is impractical. NOM is such a kind of system because it consists of a spatial distribution of discrete molecules with heterogeneous properties.

2.1.2 Stochastic model

Gillespie (1977) [29] developed a discrete, stochastic algorithm to simulate chemical reactions efficiently. This algorithm was widely used by other researchers to analyze biochemical kinetics. In Gillespie's model, the simulation time was quantized into small time steps with variable length. In each time step, one random number is used to choose which reaction will occur according to the set of weighted reaction probabilities while the other random number is used to determine the length of the time step. The chemical populations are altered according to the reaction and the process is repeated.

Although Gillespie’s model is suitable for simulating many systems, it is still limited for simulating a large proportion of biological systems. Firth and Bray [25][30] developed a simulator called StochSim written in standard C++ to predict cell signaling pathways. In their model, one or two molecules are selected from the set of molecules randomly at each time step. Another random number is then generated to see if a reaction occurs, and the system is updated immediately after each reaction occurs.

Gillespie’s model maintains the total number of molecules of each chemical species instead of representation for individual particles. It is, therefore, not possible to associate positional information with each particle in the system. The StochSim simulator provides a more realistic representation of molecular interactions by treating every molecule in the system as a unique object. StochSim is efficient at simulating small volume systems with a large number of reactions.

2.2 An agent-based stochastic model

The complexity of NOM must be represented by a large number of molecules in order to give meaningful property distributions. Although the omission of spatial heterogeneity greatly facilitates modeling and reduces the computational time of simulations, the spatial distribution of molecules plays an important role in chemical and physical reactions. The simulation model should incorporate spatial movement of NOM with water flow down into the soil, ground water and open ocean. The simulation model should also simulate chemical reactions between two molecules by choosing a nearest neighbor molecule instead of randomly choosing a molecule from the system (as in the StochSim simulator).

In chapter 3, we describe a novel approach to simulate the dynamic interaction of individual entities, interactions between NOM and micro-organism or their en-

vironment, and the mobility of individual molecules in NOM. These components possess spatial mobility and group together to form the properties of the whole system. The faster CPU and larger memory of the modern computer make modeling and simulation of a real system as an agent-based model more feasible. Modeling the dynamics of a complex system composed of interacting individuals with their internal structures has the following features: self organization, decentralized control, emergent behavior, and spatial mobility. That is, the strategies of individual agents ensure that a common goal can be achieved without central regulation, and interactions between agents can produce a stable system that displays new global behavior of the system. The agent paradigm is well suited for simulating the NOM complex system.

Thus, we developed an agent-based stochastic model of NOM evolution that explicitly includes structural and functional heterogeneity. The agent-based model of NOM evolution represents individual molecules as agents with a specified elemental and functional group composition, size, and reactivity. Temporal evolution of NOM is simulated using Monte Carlo algorithms [31] in which specific probabilities are assigned to particular transformations. The reactivity of the resulting NOM over time is predicted based on the distributions of molecular properties.

2.3 A Web-based scientific application

Unlike other simulators used in the study of NOM thus far, the NOM simulation model is a Web-based scientific application model. Web-based applications are mainly focused on end users and collaboration. Compared with the traditional stand-alone application model, there are several advantages of using the Web as a new platform for scientific study. Scientists do not need to download, build, and install the application packages or related software on their own computers

which can sometimes be a tedious task, especially for applications written in C or C++ that need to be compiled to the native code. Scientists can share the expensive computational resources or instruments, such as large scale databases, which are not readily affordable for small groups. Web-based applications offer scientists the opportunity to share their data and information with others in the research community by providing a suite of collaboration tools.

In addition to providing a core simulation engine, the NOM application model also provides an intelligent Web interface to assist users in the configuration of their simulations in a reasonable and easy way. These Web interfaces provide maximum flexibility for scientists to define new molecule representations and to include or delete various molecular transformations which meet the requirements for researchers with different interests in the area of NOM study. A set of intelligent components is provided to help users find similar simulations, to predict the simulation time, to restart a particular simulation, and so on. We use the most recent database technologies to build data analysis tools for faster access to the data results. Scientists can view the real-time data reports while their simulations are running through a standard Web browser.

A set of XML-based web service components have been developed and integrated in order to support the collaboration between scientists in the NOM research community.

2.4 Agent-based simulation toolkits

Swarm[23] [24] is a software package for simulating complex systems that was developed at the Santa Fe Institute. It is a set of libraries that facilitate the implementation of agent-based models. The basic architecture of Swarm is designed to simulate a collection of agents that concurrently interact with each other according

to a schedule of events. Swarm provides a variety of scheduling mechanisms that provide control over the execution of model actions. Swarm was originally written in the Objective C language. The Java API of Swarm is provided via a standard Java interface called Java Native Interface (JNI). JNI is a way to take native code libraries like Swarm and make them available in Java.

One of the primary benefits of Swarm is that it provides a high level of observability. A control panel allows a running model to be stopped, restarted, or executed step by step. Swarm provides a “probe” function for users to visualize the current state of the running model and probe every object in the model. An animation window shows the location of individual agents in a space. Line graphs and histograms are used to illustrate changes in collections of model objects that occur during the simulation. Additionally, simulation data, such as population size over time, can be reported periodically as the model executes.

Swarm also provides powerful random number generators and distributions which are important for Monte Carlo computer simulation.

Swarm is a powerful and flexible tool for simulating complex systems using an agent-based modeling approach in a variety of disciplines. There are excellent support mailing lists and a plethora of generally available Swarm, Java, and Objective C code. There is also an annual meeting of the Swarm Users Group called SwarmFest where researchers from diverse disciplines present their experience with multi-agent modeling and the Swarm simulation system.

RePast [22] is a Swarm-like agent-based simulation toolkit written in pure Java language. It was developed by the University of Chicago and Argonne National Laboratory. Most of the functions of RePast are borrowed from Swarm. Both the RePast and Swarm package are used in the NOM simulation model.

2.5 Technologies for building scientific application model

The NOM simulation model is a Web-based scientific application model. Several recent information technologies that are used to support the implementation of the NOM scientific simulation model are introduced below.

2.5.1 Introduction to Java programming language and J2EE architecture

Object-oriented programming languages are well-suited for implementing agent-based models. ABMs can represent populations of several types of agents which are heterogeneous at several levels. Each category of agent type, such as cellulose, lignin, and proteins in NOM, has completely different structure and behavior rules. Individuals of the same type have the same rules and parameter values with different state variable values, such as the location. Object-oriented programming languages support “inheritance” for the easy differentiation of these levels in code.

Java, an objected-oriented language, was used for implementation due to its platform independence, class reuse-ability, and Internet capabilities. Java’s built-in threads make it easier to implement multi-threaded simulations for performance improvement. The performance gap between Java and other languages, like C, C++, and Fortran, has been largely reduced due to the development of the Just-in-Time compiler (JIT) technologies, performance improvements of Java Virtual Machine (JVM), and the emergence of high performance compilers that compile the Java source to native code for particular architectures.

With the evolution of Internet technology, there is growing interest in using the Web as a new platform for scientific applications, especially after Sun Microsystem introduced the J2EE (Java 2 Platform, Enterprise Edition) technology. J2EE provides developers with the development tools and runtime capabilities to build applications that meet security, scalability, and maintenance requirements.

The latest version of J2EE as of the authorship date of this thesis is 1.4 Beta. The J2EE version used by our simulation model is J2EE 1.3. The major components in J2EE are listed as follows:

Servlets are programs that run on a web server, acting as a middle layer between a request coming from a HTTP client and a database or application on the HTTP server. Although Servlets play the same role as traditional CGI (Common Gateway Interface) programs, Servlets have several advantages over traditional CGI and CGI-like technologies [32]. Multiple servlets can share data and easily maintain information from request to request. A servlet receives a request from a client and sends the response to the client using the HTTP protocol. HTTP is a “stateless protocol”: each time a client retrieves a web page, it opens a separate connection to the web server, and the server does not maintain contextual information of the client. There are three typical solutions: cookies, URL-rewriting, and hidden form fields. Due to the real and perceived privacy concerns, some users disable cookies on their Web browser. By using URL-rewriting, the server-side program needs to do more processing. The hidden field can be used to store information about the session, but only works if every page is dynamically generated. J2EE provides a session tracking API which is a high-level interface built on the top of cookies and URL-rewriting. The servlet programmer can create a new session object, store the information into a session, and retrieve the information from a session using the HttpSession API. All these tasks can be accomplished easily and transparently. Servlets build control and navigation logic into J2EE applications, typically following a Model-View-Controller design pattern in conjunction with JSPs.

JavaServer Pages (JSPs) present dynamically generated content as part of an XHTML document sent to the client in response to a request. Although a JSP page looks more like a regular HTML file than a servlet, a JSP page is automatically con-

verted to a normal servlet the first time the page is requested. JSP does not provide any capabilities that could not in principle be accomplished with a servlet. JSPs, however, separate the presentation from the content and this enables the Web page designers to build a page layout while the servlet programmer inserts the dynamic content. JSPs use XML-like tags and scriptlets, written in the Java programming language, to encapsulate the application logic that generates the content for the page. Additionally, JSPs programmers can reuse JavaBeans and create custom tag libraries which enable web-page designers who are not familiar with Java to enhance web pages with powerful dynamic content and processing capabilities.

Java Naming and Directory Interface (JNDI) provides component location transparency in a clustered J2EE environment.

Java Database Connectivity (JDBC) handles communication between a Java program and database, with all database input/output via SQL. The JDBC standard was defined by Sun Microsystems. Individual providers can implement and extend the standard with their own JDBC drivers. Separation of the API from particular drivers enables developers to change the underlying database without modifying the Java code that accesses the database.

JavaBeans are a set of classes and programming conventions that constitute a component development model for the Java language. JavaBeans provide the programmers with tremendous flexibility by allowing programmers to reuse and integrate existing components.

Aside from the components that we used in our system, J2EE also includes the following components: Enterprise JavaBeans (EJBs), Java Connectivity Architecture (JCA), Java Message Service (JMS) and Java Management Extensions (JMX),

The J2EE platform manages the infrastructure and supports Web services to enable the development of a secure, robust, and interoperatable scientific Web ap-

plication. The J2EE platform is designed to provide both server-side and client-side support for developing multi-tier applications. Such applications are typically configured as a client tier to provide the user interface, one or more middle-tier modules that provide client services and application logic, and backend enterprise information systems for data management. Figure 2.1 shows the typical J2EE environment [1].

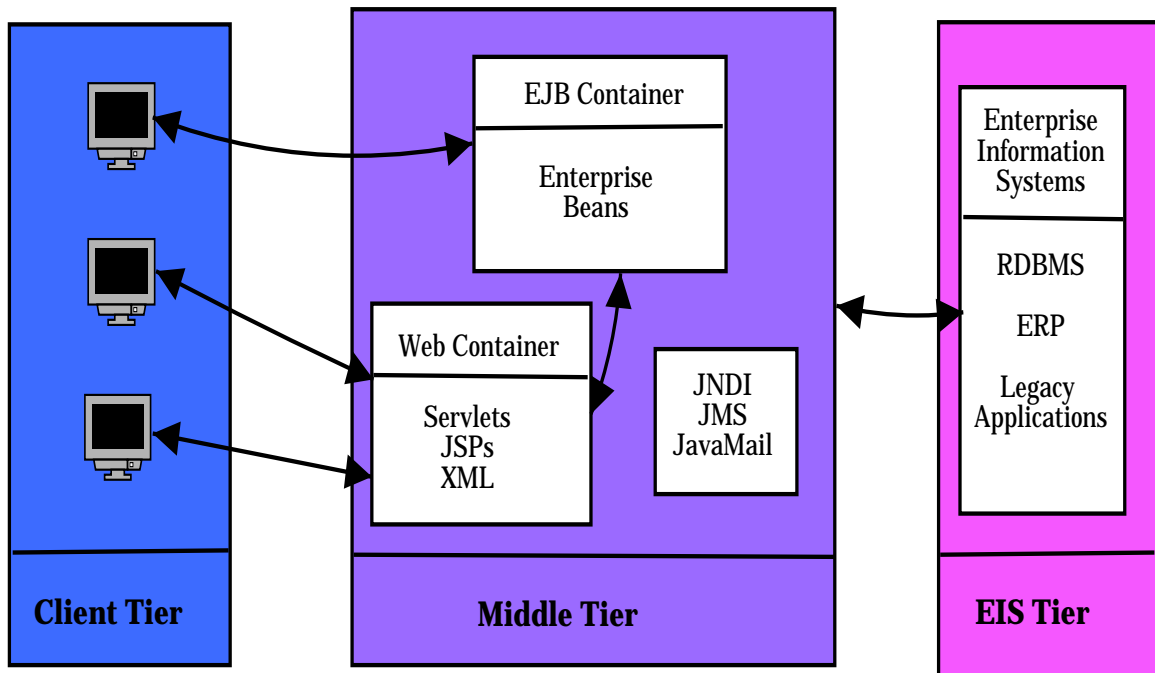


Figure 2.1. The J2EE environment (adapted from Nicholas (2000) [1])

2.5.2 Oracle database overview

For Web applications, a database plays the vital role of storing and accessing the simulation data. In the NOM system, data inputs from the Web, including user information and simulation parameter setup, are stored into the Oracle database. After a simulation is invoked, the simulation results are stored in the Oracle database while the simulation is still running. The NOM system model requires a database

that is capable of handling large amounts of data and information. The Oracle database was chosen because it is the most scalable, full featured, and popular object-relational database available. The Oracle database provides many powerful tools, such as Oracle PL/SQL language, SQL, and RDBMS for Data Warehouse and Data Mining.

The Oracle9i package provides Oracle9i Reports Developer and Oracle9iAS Reports services. The Report Developer helps the J2EE developer rapidly create web and paper reports against the Oracle database. Oracle9iAS Reports Services publishes these reports in any format (including HTML, XML, JSP) and can send the reports to any destination including a Web browser and an Oracle9iAS Portal. They are the major tools which we used to build the Data Warehouse, do data-mining, and on-line transaction processing (OLTP).

2.5.3 Application server overview

An application server typically consists of an EJB container, a servlet, and a JSPs container. An application server provides services such as JNDI directories, database connection pooling, integration with distributed systems, and resource management.

There are many application servers available: IBM Websphere Application Server¹, Weblogic from BEA², JBoss³, iPlanet (which is now a division of Sun and is a core component of the Sun Open Net Environment (Sun ONE)⁴), Apache TomCat⁵, Oracle9i application server (OC4J), and Orion application server⁶. Among those products, we choose OC4J (Oracle9iAS container for J2EE). It's a fast, lightweight,

¹<http://www-3.ibm.com/software/webservers/appserv/>

²<http://edocs.bea.com/>

³<http://www.jboss.org/overview.jsp>

⁴<http://www.sun.com/software/>

⁵<http://jakarta.apache.org/tomcat/>

⁶<http://www.orionserver.com/>

highly scalable, and easy-to-use J2EE environment. OC4J is written entirely in Java and executes on the standard Java Development Kit (JDK) virtual machine. It provides all the containers that J2EE specifies. OC4J is based on the technology licensed from Ironflare Corporation, which developed the Orion application server.

2.6 Summary

In this chapter, we compared the NOM simulation model with other existing models for NOM study. Our NOM simulation model is the first model to both describe organic carbon transfer quantitatively and to accommodate the structural and functional heterogeneity of NOM using the agent-based modeling approach. The NOM simulator simulates various molecular transformations due to chemical reactions and the physical transport of NOM. These transformations are separate modules built into the NOM simulator so that scientists can either include or omit various molecular transformations, depending on their research interests, while they configure their simulations. For example, one group of scientists in the Center for Environmental Science & Technology at the University of Notre Dame is only interested in the simulation of NOM sorption at this time.

Although there are several advantages for implementing scientific applications using the Java programming language, the performance of such applications needs to be improved. We analyzed the NOM simulation system from the software engineering perspective and explore the potential for performance improvement, the scalability and the reliability of scientific applications written in Java. Aspects of performance and scalability that we considered include object creation, memory usage, data output, and parallel and distributed computing.

The NOM application model is the first Web-based application model in the study of NOM. The NOM model can support collaboration and communication

between scientists in the NOM research community. This collaboration and communication is facilitated by the model's intelligent interface, core simulation engine, data analysis tools, and collaboratory built upon J2EE technology. We describe these components in detail in the following chapters.

CHAPTER 3

A SIMULATION MODEL FOR A COMPLEX SYSTEM

3.1 Introduction

NOM (a mixture of natural organic molecules), micro-organisms, and their environment form a complex system. The global phenomenon of a complex system can often be observed by simulating the dynamic behavior of individual components and their interactions in the system. Complex systems often have emergent properties. The evolution of NOM over time from precursor molecules (such as cellulose, lignin, and protein) to eventual mineralization involves various molecular transformations. These transformations involve chemical reactions, adsorption, aggregation and physical transport in soil, ground, or surface waters.

The NOM simulation system is an agent-based stochastic model with an intelligent Web interface that can model NOM, mineral surfaces, and microbial interactions near the surface of the soil. In the stochastic model of the evolution of NOM in discrete time and space, NOM is presented as a large number of discrete molecules with varying chemical and physical properties. Individual molecules can be transported through the soil medium via water flow, adsorb on the soil particle surfaces, and react with other molecules, micro-organisms, and the environment. The chemical and physical attributes, as well as behaviors of molecules are simulated using the Java programming language, and the Swarm and RePast agent-based modeling libraries. By simulating the behaviors of individual molecules in the system, the dis-

tributions of physical, chemical, and biological properties of NOM can be predicted. Additionally, such simulations can provide scientists in biology, geochemistry, and ecology valuable information for their studies.

The NOM simulation system is designed as a distributed, Web-based system using the Java 2 Enterprise Edition (J2EE) platform. This Web-based system allows users to configure their own simulations and invoke the core simulation engine from a Web browser. The system also offers the capability to let users visualize the simulation results adapted to their requirements through the Web browser. The Web interface is written in Java Server Pages, Java Servlets, and Java Beans and communicates with a database via Java Database Connection (JDBC). Several intelligent agent components, defined in the interface, assist users in configuring their simulations more easily and more reasonably.

The purpose of the NOM simulation system is to provide scientists a testbed for their theoretical analysis and experimental results for NOM study. After the simulation system is evaluated, we expect that the system will help scientists better understand the NOM complex system by providing them with information for predicting the properties of the NOM system over time.

In section 2, an overview of our NOM simulation system is given. The core simulation engine of this model is described in section 3 in detail. Intelligent agent components for configuration assistance are presented in section 4. We briefly describe the data analysis in section 5 and draw a conclusion in section 6.

3.2 NOM simulation model overview

The NOM simulation model includes three components: an intelligent Web interface that assists users in configuring simulation parameters; a core simulation engine that does the computations; and a data analysis package that allows users

to view their simulation results through a standard Web browser. We employ the latest Internet technology and scalable Web-based database management system to improve the reliability of the simulations and to facilitate analysis of the large dataset results.

The NOM simulation model is built upon the J2EE architecture running on a distributed cluster. This cluster has multiple dual processor PCs running Redhat Linux 8.0 and Windows 2000 operating systems. These machines in the cluster include a HTTP server, application servers, database servers, a reports server, and a data mining server. The architecture is showed as Figure 3.1.

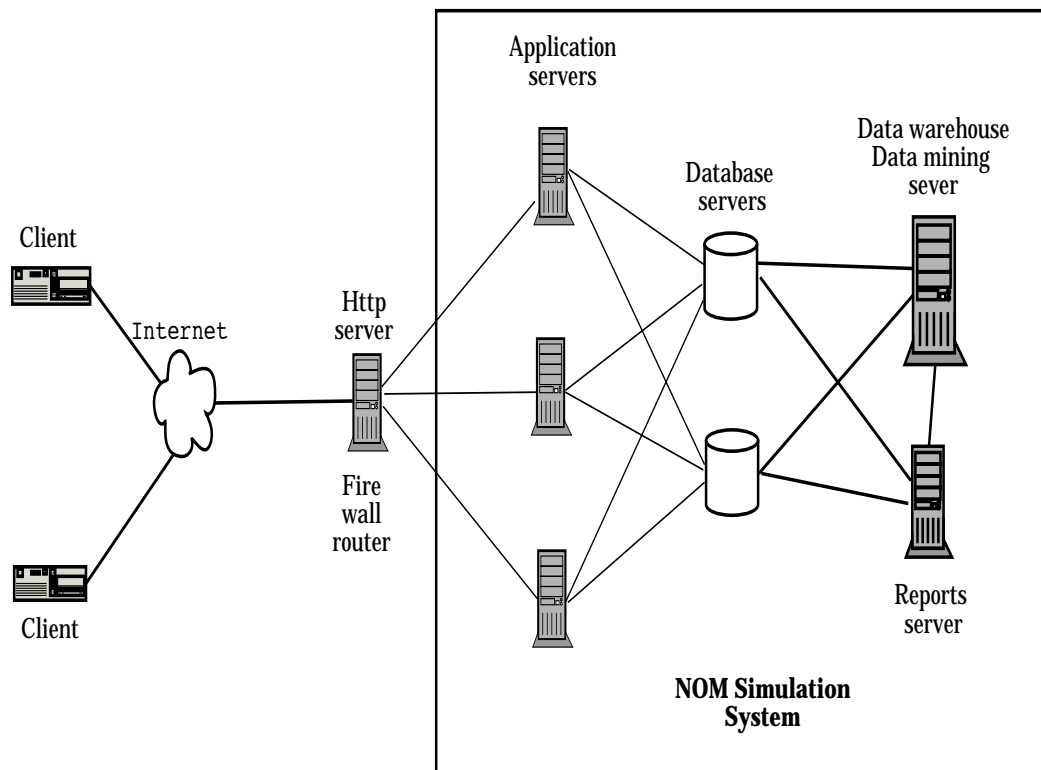


Figure 3.1. The architecture of NOM simulation Model

The number of servers can be scaled to meet our requirements for better performance.

3.2.1 Access the NOM model from web

Users can access the NOM simulation model from a standard Web browser. The work flow is designed as follows:

1. Users login, if the user is a new user, fill out the information form first then sign up.

2. Users (1) choose one of the environment sets from a list that contains all the available settings in the database or (2) define their own simulation environment parameters using an “Environment editor”. Users can also decide whether to make the setup public or not by marking a check box.

3. Users (1) choose one or more molecule types from an existing molecular type list or (2) define new molecular types using a “Molecular editor”. Users are also asked to define the distribution (relative percentages) of the molecule types that they have selected.

4. After the above setup, an intelligent agent gives users a predicted running time for their unique setups. Users can change their individual setups if necessary. An agent also helps users find already completed simulations that have similarities with their own setup, if any exist.

5. Users can invoke the simulation or discard their setup.

6. While the simulation is running, users can view the real-time reports for their simulations from the Web.

7. Users are sent an Email which notifies them that their simulations are completed.

3.3 Core simulation engine description

The NOM core simulation engine is a stochastic, agent-based simulation model built entirely using Java, JDBC, and the Swarm and RePast libraries. The sim-

ulation engine can run either in batch mode or in graphical use interface (GUI) mode.

3.3.1 Simulation algorithm

In the NOM simulation model designed in this research project, individual molecules are represented as agents with their own complex structures. These molecules move along with the water flow and react with each other in a 2D discrete grid, i.e. a rectangular lattice composed of multiple cells. Each molecule can occupy at most one cell and each cell can host at most one molecule. During the execution of the simulation, each molecule may move to another location, experience adsorption to or desorption from a particular site, and chemically react with other molecules. The basic algorithm for the simulation engine is illustrated in Figure 3.2. It includes the following major steps:

- Precursor molecule creation: Molecules are created by obtaining values for all quantities in the molecule’s structure. The quantities are described in detail in the “Molecule Object” section.
- Movement and sorption: As molecules move along with the flow, they can be adsorbed to surfaces or desorbed from them when some random events occur.
- Reaction: Molecules are allowed to react for a predetermined length of time under fixed external conditions and molecule structures are recorded at defined time increments.
- Property calculation: In order to better understand and monitor the global properties of NOM over time, large amounts of information for the system needs to be stored in the database for calculation and analysis.

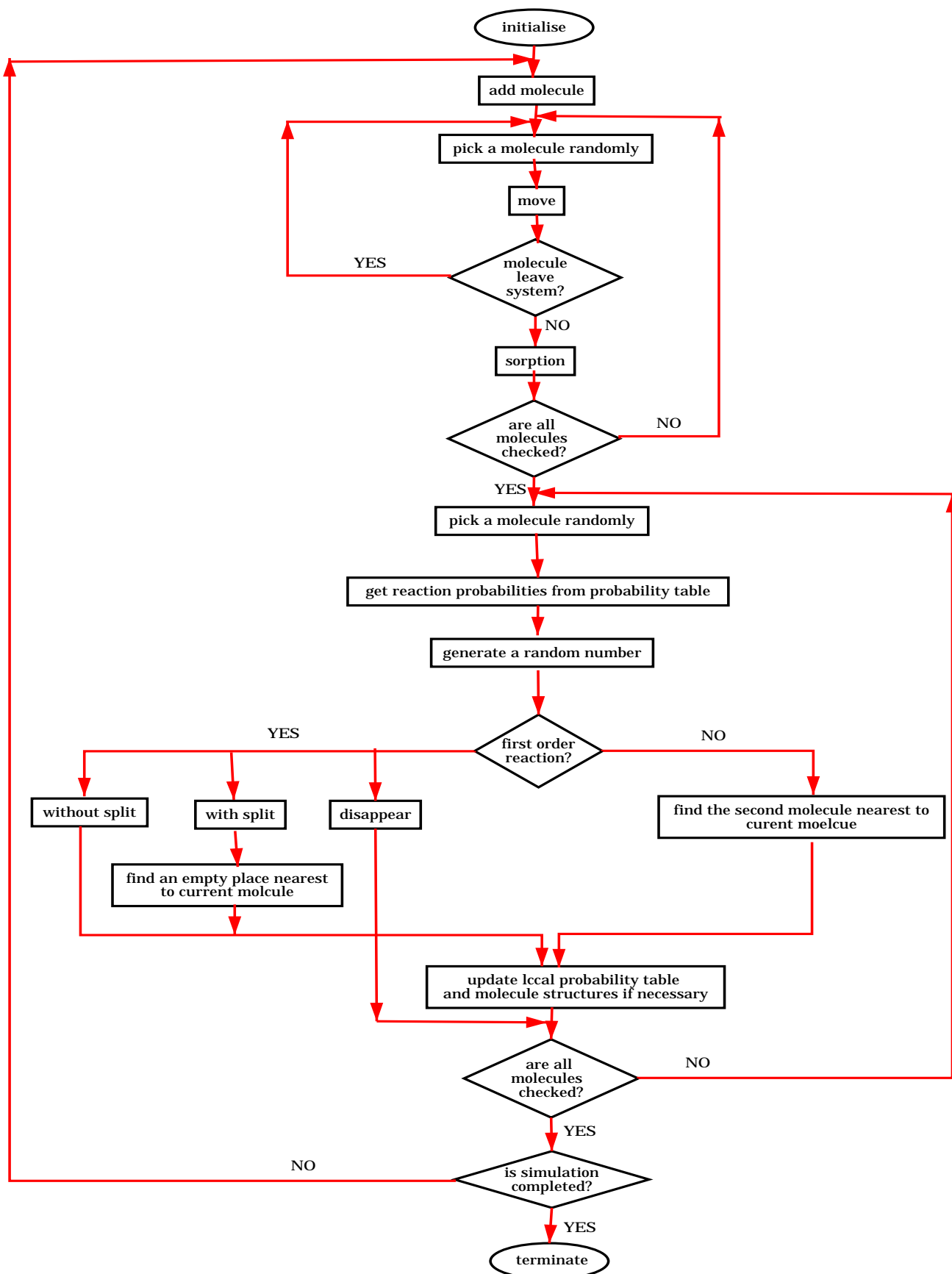


Figure 3.2. A flow chart of the algorithm for the NOM simulation model

At the beginning of a simulation, parameters including environment variables, molecular types, and their distribution are read from the database into the simulation server. The total number of molecules in the simulation can be determined by specifying the discrete lattice size and the molecule density. The NOM complex system includes different types of molecules.

The simulation time, defined by the user, is divided into a very large number of equal, discrete, and independent steps called “time steps.” In each time step, each molecule in the system is chosen in random order. A random number is generated to see which reaction will occur. To determine which reaction will be taking place in a given time step, we calculate the probability of each reaction according to the environment variables and the structure of the chosen molecule. This random number is used to compare with the precalculated reaction probabilities associated with the chosen molecule. The molecule structure and properties associated with the molecule are updated after each physical or chemical reaction occurs.

3.3.2 Reaction probabilities calculation

The probability for each reaction type is determined by the molecular structure of the particular molecule and the environment parameters. The formulas for calculation are defined by Cabaniss (2002)[15]. The probabilities associated with each molecule have been calculated and stored into a local look up table in each molecule object. In order to reduce the computational time, a two dimensional global probability table is used to store the probabilities of the different reaction types for each molecule type. If a particular molecule has the same structure as the molecule type in the probability table, then the probability can be retrieved from the global probability table instead of being calculated whenever a molecular structure changes.

In the simulation, the sum of all the reaction probabilities is controlled so that it is less than 1 percent. It is determined by choosing a reasonable length of time steps, ΔT . The interval, $[0, 1]$, is partitioned into 11 subintervals as shown in Figure 3.3. The length of the first interval is equal to the probability of the first reaction type; the length of the second interval is equal to the probability of the second reaction type, and so on. The length of the last interval is the probability in which no reaction will occur. The random number from the interval $[0, 1]$ resides in one of these intervals, and it will decide which chemical reaction will occur if there is one.

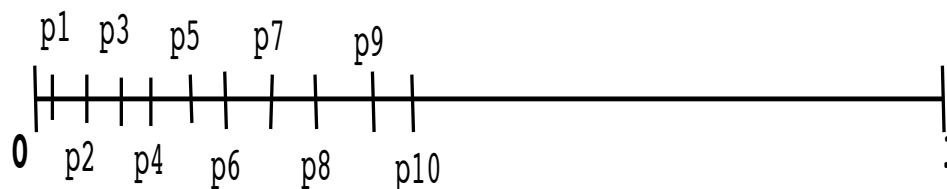


Figure 3.3. Probabilities

3.3.3 Molecule objects

NOM molecules are hypothesized to be derived from macromolecules such as proteins, polynucleotides, tannins, lignin, or micro-molecules such as phospholipids, sugars, amino acids. To avoid dealing with each possible macromolecule, we work with “representative” molecules, which have typical structures and elemental compositions. These representatives can be used to obtain reasonable similarity to the actual macromolecular precursors. There are three typical representatives: cellulose, lignin and proteins. Users can also define their own molecule types using the “Molecule Editor” provided by our Web interface.

Given that a true molecular representation is impractical, we need an intermediate level molecular representation which is more specific than simply “percent

carbon” but less detailed than a molecular connectivity map. The data used for constructing representations of molecules includes the following components:

- Elemental formula, i.e. the number of C, H, O, N, S, P atoms in the molecule. The molecular weight can be easily calculated from the elemental formula.
- Functional group count, i.e. carboxylic acid, alcohol, ester groups. There are a total of 19 possible functional groups in each molecule structure.
- A record of the molecular “origin”, i.e. a given molecule’s start location, type of molecule, time of molecule entering the system. This allows for the calculation of separate “turnover times” and apparent ages for individual molecules or fractions. The location of each molecule in the simulation is represented by x, y coordinates of a 2-dimensional lattice.

3.3.4 Reactions and Processes

NOM evolves either by physical reactions or chemical reactions. The key distinction between these two is whether the molecular attributes change or whether the position/state of the molecule changes. The physical reactions modeled in our simulation are sorption to and from surfaces, as molecules are transported by water through soil pores. Sorption is not a transformation of molecular structure; however, it will affect the probability of other reactions by changing the environment of a given molecule. The simulation stops when the number of “time steps” defined by the user is completed.

Table 3.1 presents the ten types of chemical reactions that have been modeled in our simulation, including first-order and second-order chemical reactions.

These ten chemical reactions are separated into four categories:

- Second order reactions are defined as follows: Two molecules A and B are

Table 3.1. Chemical reaction types

Reaction Name	Reaction Type
Ester condensation	Second order
Ester hydrolysis	First order with split
Amide hydrolysis	First order with split
Microbial uptake	First order with molecule disappear
Dehydration	First order with split
Strong C=C oxidation	First order with split (50%) of the time
Mild C=C oxidation	First order without split
Alcohol(C-O-H) oxidation	First order without split
Aldehyde C=O oxidation	First order without split
Decarboxylation	First order without split

summed to form a new molecule C; the new molecule C replaces one of the predecessor molecules while the other predecessor molecule disappears from the system.

During each time step, molecule A is chosen. If the generated random number indicates that a second order reaction should occur, a second molecule B, which is nearest the location of A, is found using a spiral algorithm for the grid. Molecule B should have non-zero alcohol groups. These two molecules combine to form Molecule C. The probabilities are calculated for molecule C and the structure for molecule C is determined. Molecule C replaces the position of A and molecule B leaves the system.

- First order reactions with a split are defined as follows: The predecessor molecule A is split into two successor molecules B and C, molecule B occupies the position of molecule A, and one of the empty cells nearest molecule B is filled with molecule C.
- First order reactions without a split are defined as follows: The transformation only changes the structure of the predecessor molecule A.

- First order reactions with the disappearance of a molecule are defined as follows: The predecessor molecule A disappears from the system. This reaction occurs when the molecule is small enough that it can be enveloped by a micro-organism like fungi or bacteria.

As illustrated in Figure 3.2, in each time step, for each molecule, a uniform random number is generated. It is this number that determines whether a chemical reaction will occur and if one does occur, it determines the reaction type. After a reaction takes place, the probability table for the current molecule is updated. The molecule structure is changed if necessary and the new probability table is assigned to newly formed molecules, if there are any.

3.3.5 Environmental parameters

Environmental parameters in the simulation setup include both simulation control parameters such as the simulation time step, the grid size, and several natural environmental factors such as pH, temperature, microbial density, fungal density, oxygen density, light intensity, and so on. These parameters affect the reaction probabilities and the NOM evolution. Likewise, NOM evolution can also affect the natural environment.

3.3.6 Molecular Properties

Some functional and analytical properties of NOM are calculated and predicted because they are useful and interesting to scientists. Some properties are easy to calculate like molecular weight, molecular charge, and charge density. On the other hand, others involve non-trivial computations. The reactivity of the resulting NOM over time can be predicted based on the distributions of molecular properties.

3.3.7 Pseudo-random number generator

The pseudo-random number generator plays a crucial role in the stochastic simulation model. In the NOM simulation model, random numbers determine which chemical reactions occur at each discrete time step in the simulation. At the initial state of the simulation, random numbers are used to place individual molecules into the 2D space. In every time step, molecules in a collection are examined in random order. Although random number generators have been used for computer simulations for decades, no generator is “safe” for all applications. Fortunately, the Swarm package provides us a high quality random number generator which was used in this model. The Swarm package also provides a “shuffle” function and we used it to randomize the order of molecules in a collection at every time step.

3.3.8 Program description

The core simulation engine consists of 9 Java classes. Each class contains a collection of attributes and methods. Figure 3.4 shows a UML diagram of the relationships between some of the classes. The main class is *StartMolecule*. Associated with it are *ObserverSwarm* that runs the program in GUI mode and *ModelSwarm* that runs the program in batch mode. If the program is running in GUI mode, *Modelswarm* is an instance in *ObserverSwarm*. The *ModelSwarm* maintains a list of *Molecules*, each *Molecule* is associated with a *ProbabilityTable* and a *Reaction*. *ReadData* reads all the data associated with the current session from the database into memory in a simulation server by using JDBC.

Besides the classes that are listed above, there is one more class that does not show in the diagram, *Constant* class. *Constant* is an interface class which defines a set of constants that can be used in other class definitions by implementing the *Constant* class.

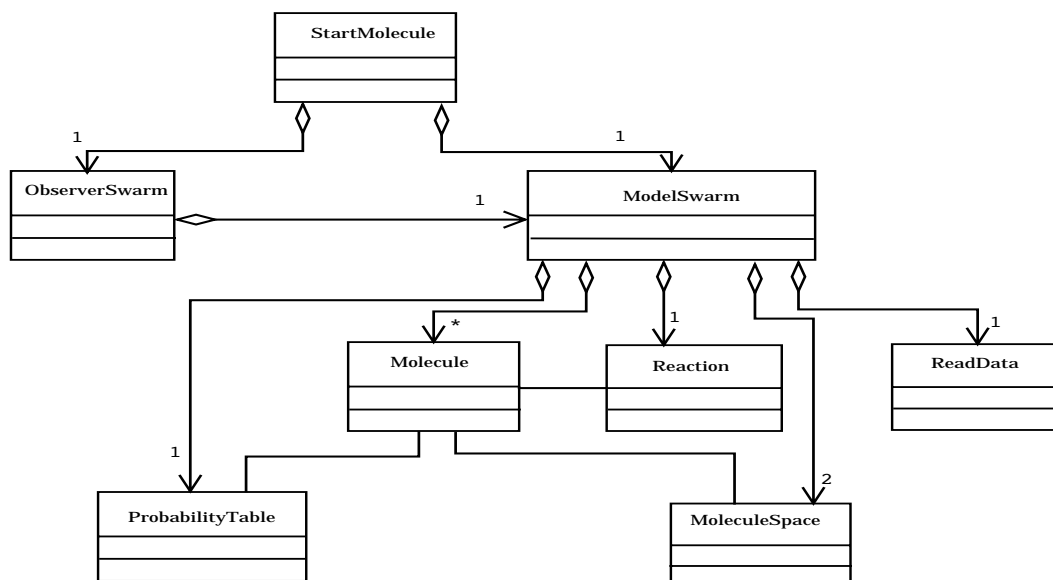


Figure 3.4. Class Diagram

3.3.9 Input and Output

All the inputs to the core simulation engine can be handled via three approaches as shown in Figure 3.5.

The Swarm library provides a set of API for reading in a Lisp format. The input parameters for a Swarm application can be stored in a .scm file by specifying the class name, field name, and value. This format is only used during the system development stage. The NOM model offers users two ways for entering parameters. These are XML-based data format and Web interface.

XML provides information about data in a standardized, human readable format. The XML format has been widely used in Web applications for exchanging data and information between users and applications. We defined an XML-based NOM Modeling Language (NOML) for describing parameter setups and molecule types. The NOML can be used to support collaboration between users by sharing their simulation setups, simulation results, and molecule types. We further describe

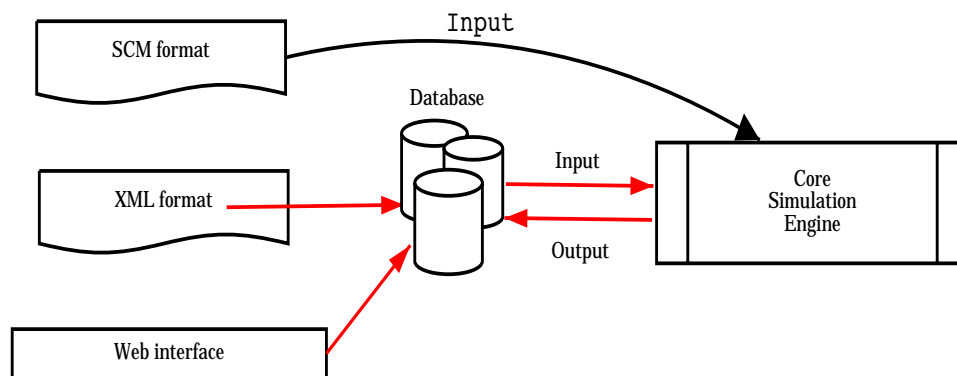


Figure 3.5. Input and Output

NOML in Chapter 5.

The Web interface provides remote users a way to easily configure their simulations. The Web interface is described in section 5.

In each time step, the data information for the reacted molecules is written into the database. The large amount of data takes time to write, especially during heavy network traffic. To alleviate this problem, a multi-threading program model for data output is developed and runs on the simulation servers with dual processors. Strategies for improving performance and their implementation are presented in Chapter 4.

3.4 Intelligent agent components

The Web interface provides remote users with a configuration tool. Users can specify the simulation parameters and invoke the simulation from a standard Web browser using HTTP protocol. The intelligent configuration interface can also guide users in setting up their simulations step-by-step. State-of-the-art Internet technologies, J2EE technology and an Oracle database, were used in the interface implementation.

Our NOM application is a thin-client, three-tiered application using J2EE technology.

A client tier interacts with end users and displays information from the server to end users. In general, HTML and Java applets in a client container implement this tier.

For the design of our Web interface, we use standard HTML, XML/XSL style sheets and JSPs instead of Java applets. Although applets can make the Web page look more attractive, not all Web browsers can support Java applets very well. Sometimes, users need to download the Java Runtime Environment (JRE) in order to run the Java applets correctly. Normally, the size of Web pages written in Java applets is bigger than that of Web pages written in HTML and JSPs and have longer response times, especially when the Internet connection is slow.

A Web tier accepts users' responses from the client tier and generates the presentation logic. We used JSP pages for presentation logic and Servlets for session management.

An application tier handles the core scientific logic of the application. EJB components in an EJB container and JavaBeans are used for implementing the application tier. In the NOM system, we used JavaBeans.

By implementing the thin-client application, the logic to access the remote objects and databases is removed from the client and moved to an application server. Moving the database processing to an application server can reduce the network traffic and provide higher performance.

Figure 3.6 shows the Web interface implementation model using J2EE technology.

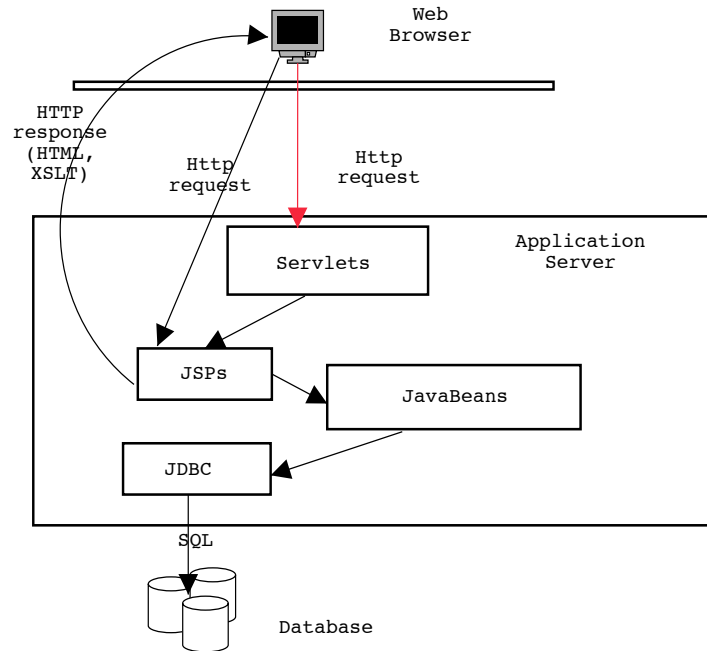


Figure 3.6. The implementation model for the Web interface

3.4.1 Model-view-controller Architecture

Our NOM interface configuration tool is implemented by following the Model-View-Controller (MVC) design paradigm. Several Web application frameworks are based on this architecture, such as Struts¹ and Webwork².

MVC originally appeared in Smalltalk-80 as a method for separating user interfaces from underlying application data. The MVC architecture divides applications into three layers (model, view and controller), and decouples their respective responsibilities. Each layer handles specific tasks and has specific responsibilities to other layers.

A model represents application data and application logic or operations that govern the access and modification of data. The model notifies the view when it

¹<http://jakarta.apache.org/struts/>

²<http://sourceforge.net/projects/opensymphony>

changes and provides the ability for the view to query the model about its state. It also provides the ability for the controller to access application functions encapsulated in the model.

A view accesses data from the model and specifies how that data should be presented. It forwards the user input to a controller and updates the data presentation when the model changes.

A controller dispatches user requests and selects views for presentation. It interprets user inputs and maps them into actions to be performed by the model. In a Web application, user inputs are HTTP GET and POST requests to the Web tier. A controller selects the next view to display based on the user interactions and the outcome of the model operations. Some applications have one controller for each set of related functions, while other applications use a separate controller for each client type.

The MVC design pattern separates design concerns (data persistence and behavior, presentation, and control), decreases code duplication, centralizes control and makes the application more easily maintainable. MVC also helps developers with different skills to focus on their core skills and collaborate through clearly defined interfaces. MVC is particularly well-suited for interactive Web applications where a user interacts with a Web site, with multiple iterations of screen page displays and multiple round-trips of requesting and displaying data [1].

3.4.2 Database design

We have designed an Oracle database for supporting the configuration interface and collaboration. There are six tables: USERS, ENVIRONMENT, MOLECULE-TYPE, SESSION_ENVIRONMENT, SESSION_MOLECULES, USER_SESSIONS. We declared *userid*, *environid*, *typeid*, and *sessionid* as *sequences*, which are unique

integer numbers generated by Oracle. One user can set up several different simulations called sessions. We can keep tracking each session for every user according to the *userid* and *sessionid*. For every simulation, environment variables, molecule types, and distribution can be stored and retrieved by the *environid* and *typeid*. Figure 3.7 shows the diagram of the relationship between those tables.

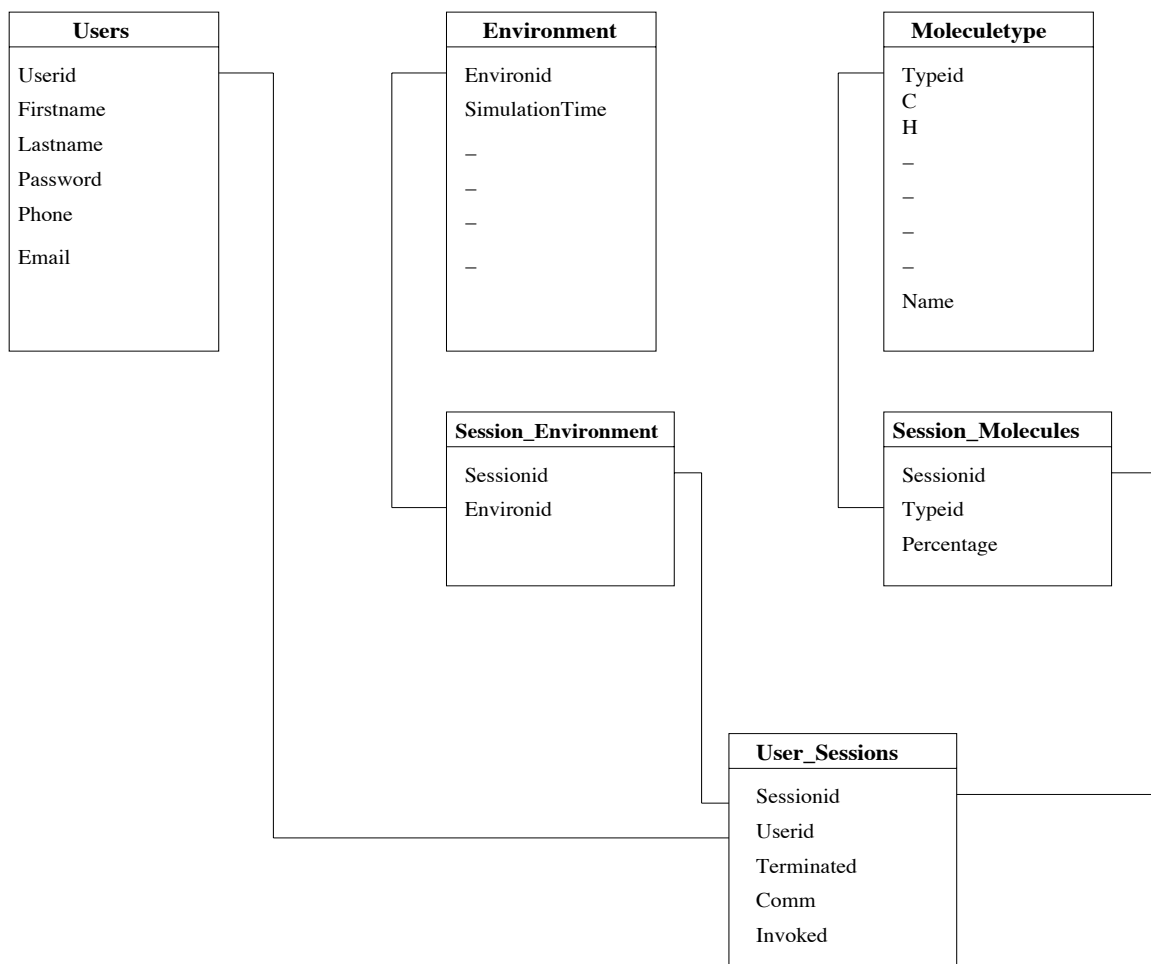


Figure 3.7. Diagram for database design

The Web interface uses those six tables as the backend to store user information and parameter setup.

3.4.3 Web Interface Layout

The Web interface is implemented mainly by using JavaBeans, JSPs, and Servlets following the MVC design pattern. JavaBeans are the “model” for implementing the application logic and storing and retrieving data from the database. Servlets are the “controller” which implements the session management. JSPs and HTMLs are the “viewer” for implementing the presentation logic. Figure 3.8 shows a work flow chart for the configuration of a simulation. The form validations are mostly done by using JavaScripts at the client side in order to reduce network traffic.

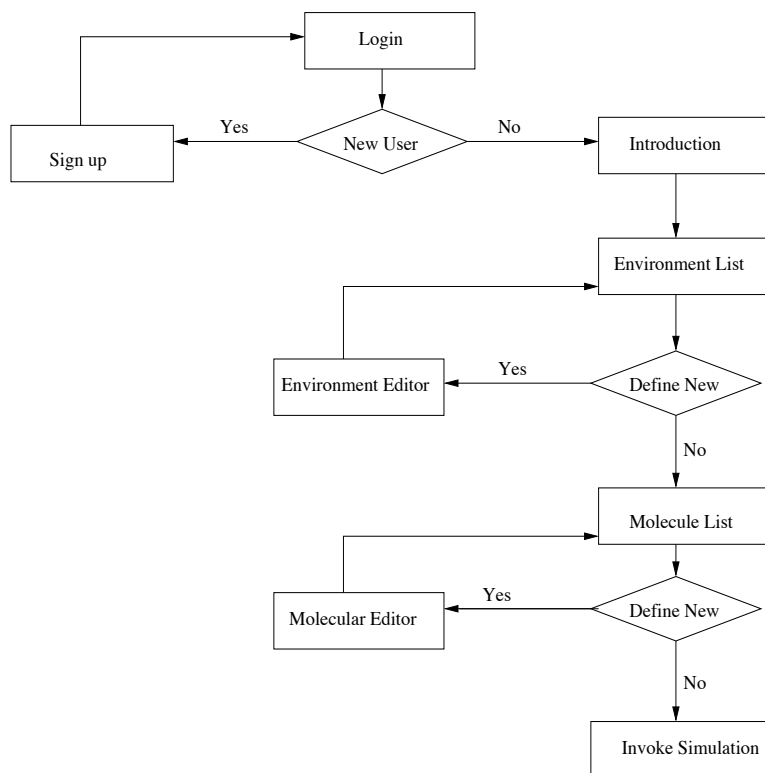


Figure 3.8. Interface configuration steps

3.4.4 Intelligent agent components

Intelligent agents in the Web interface can not only provide a novice user with guidance during the configuration process, but also provide the experienced user more information on how to better use the simulator. These intelligent agents are components based on JavaBean technology, and every component is a separate module. By following this design diagram, we can more easily maintain the code and reuse it elsewhere. A summary of these intelligent agents is described as follows:

- The sending email agent

The agent automatically sends an email to the user after the simulation is done. This email message gives the user information about the status of the simulation, either it was successfully completed or it was terminated due to any exception that was caused by hardware malfunction or software problem.

- The running time prediction agent

A user can ask this agent to statically predict how long it will take to run the simulation according to the user's configuration inputs before the user invokes the simulation. From our experience, the running time is primarily determined by the number of simulation time steps and the number of molecules in the simulation problem. According to the time returned by the agent, the user can decide if the simulation configuration should be adjusted.

The running time for an application is determined by the CPU usage at different times. This agent also provides the capability for dynamically predicting the running time while the simulation is running. The actual time when the simulation was invoked is stored in the *USER_SESSIONS* table and the running time steps is stored at each time step. The agent computes the average

time spent for each time step from the formula:

$$\text{Average time} = \frac{(\text{Current actual time} - \text{Start actual time})}{\text{current time step}}$$

The time still needed to finish the simulation can be calculated by:

$$\text{Time left} = \text{Average time} * (\text{Total time step} - \text{Current time step})$$

Users can submit a request that asks how much time remains before the finish of their simulation by simply clicking a button.

- The similar simulation finder

Since executing a simulation is a time-consuming task, especially when the grid size is big and the running time is long, we provide users the option to take advantage of the simulation's results that were done earlier by the user or by other users. A similar simulation finder agent helps users find finished simulations that have some similarities with the current setup in the database. Users can either retrieve the simulation results from the database without running their simulations or submit the simulation for execution according to their own preference.

A simulation setup includes three parts: environment parameter setup, molecular type selection, and definition of the distribution for each selected molecular type. The similarity between two simulation setups can be defined as follows:

- (1) If two simulation setups do not consist of the same combination of molecular types, then there is no similarity between them.
- (2) Otherwise, users can ask to find similar simulations according either to the molecular distributions or environment setup. The agent can return an

ordered list of similar simulations, with the most similar simulation listed first. Users can also view the parameter setup of similar simulations.

The basic algorithm for finding the similar simulations is described as follows:

After users issue a request to search for similar simulations according to distribution or environment setup, the agent retrieves the data sets in the database.

These data sets can be viewed as a number of points in a high-dimensional space with j attributes $(a_1, a_2, a_3 \dots a_j)$. The task is transformed to compute the distance from the current parameter setup to the other parameter setups.

Each attribute in these data sets is normalized to the $[0, 1]$ interval using the following equation.

$$\frac{a_{ij} - \min(a_i)}{\max(a_i) - \min(a_i)} \quad (3.1)$$

By employing the “Euclidean distance,” the distance from the current input data set to other data sets retrieved from the database can be computed. The data sets that have the smallest distance have the highest degree of similarity with the current setup.

- Automatic restarting agent

The agent helps users extend their simulations for longer time steps by restarting their simulation from the point where it stopped. While one simulation is running, the data information about reactions is stored in the database every time step for further data analysis. In addition to this, the state of the simulation system is stored in the database at different checkpoints. A table called *restarter* is created for storing the state. A series of SQL statements is executed through JDBC code. The insertion and updating of the state at each checkpoint are treated as a data transaction process to guarantee data

integrity.

3.5 Data analysis

The Web-based application model is developed for an end user to access the simulation system. One of the primary goals of this model is to provide an explanation of the simulation results, to help users better understand their data, and to predict the global properties of NOM over time. Large-scale scientific simulations take days to run and produce massive amounts of data. A user typically needs to run several variations and compare many sets of results with their experimental data. It is very difficult for users to do this with only the raw data. Gray (2002) [33] shows how important the new data mining algorithms are in helping scientists to access their on-line data more quickly. The Oracle9i tools are used to build the data warehousing and data mining and to incorporate them into the simulation model in order to organize, visualize, and analyze the huge amount of data. We enabled the display of simulation results independently from the processing. Users can examine the results from the Web site during and after the simulation.

3.6 Conclusion

In this chapter, the NOM simulation model is presented. The intelligent Web interface, core simulation engine design, and modeling are discussed in detail. Additionally, the data analysis, a critical part of the NOM model, is briefly described. Data analysis and data mining technologies and algorithms used in this model are discussed in Huang (2002)[34] in detail.

A set of intelligent components is developed to provide guidelines for users. The algorithms used to implement these components, however, need to be improved for efficiency as the data sets become larger and larger.

Although there are several advantages of choosing the Java language for the implementation of scientific simulations, the Java language still has shortcomings when compared with traditional languages. Although the J2EE platform is suitable for the Web-based scientific application model, performance still is an issue. Improving the performance tuning of the NOM application model is necessary in order to achieve faster turnover time.

CHAPTER 4

EXPLORING PERFORMANCE IMPROVEMENT FOR JAVA-BASED SCIENTIFIC APPLICATIONS

4.1 Introduction

C, C++ and FORTRAN have traditionally been used for modeling scientific applications. Since the Java programming language was introduced by Sun Microsystems in the mid-1990s, there has been growing interest for using it in scientific and engineering applications. The reason for this is that Java offers several features, which other traditional languages lack. Scientists use various platforms (such as Windows, Unix and MacOs) for their scientific studies. It is impossible to deploy an application written in C, C++, or FORTRAN languages from one platform to the other without rebuilding the application or changing the code. One of the most attractive features of Java is its portability, “write once, run anywhere.” Java runtime environment (JVM) provides an automatic garbage collection feature that reduces the burden of explicitly managing memory for programmer. The Java built-in threads implementation and Java’s Remote Method Invocation (RMI) mechanism make it easy for parallel computing and distributed computing.

Although there are many attractive features provided by the Java language and the J2EE architecture makes Java a potential language for scientific applications, performance remains a prime concern for program developers using Java. The portability and memory management implementation in JVM impose a penalty on the performance. Ashworth (1999) [35] discussed several issues related to the use of

Java for high performance scientific applications.

However, much research has been done to reduce the performance gap between Java and other programming languages. This research includes Just-in-Time (JIT) compilers that compile the byte code into native code on-the-fly just before execution, adaptive compiler technology (Sun's HotSpot VM), third-party optimizing compilers that compile the Java source code to the optimized bytecode, and high performance compilers that compile the Java source to native code for a particular architecture (IBM High-Performance Compiler for Java for RS6000 architecture). Bull (2001) [36] rewrote the Java Grande Benchmarks in C and FORTRAN. Bull also compared the performance between these languages in different Java runtime environments on different hardware platforms. The results demonstrate that the performance gap is quite small on some platforms.

The runtime environment optimization is an important aspect to study in order to achieve high performance. Determining and understanding the factors that affect the performance of scientific applications from a software engineering perspective and identifying and eliminating the bottlenecks that limit scalability at the software development stage are also necessary for high performance scientific applications.

In this chapter, several approaches for analyzing and improving the performance of a particular scientific application, the NOM simulation model, have been described. The NOM simulation model is intended to simulate NOM behavior over time in a complex system. Such behavior includes transport through soil pores, adsorption to or desorption from mineral surfaces, and interactions with each other. The simulation model was built using the Java programming language and the Swarm library. The NOM simulation model is a typical distributed, stochastic scientific application that uses an agent-based modeling approach. It generates a substantial data set that must be stored in a remote database, able to be manipu-

lated for producing useful information. The application simulates the behavior of a large number of molecules. An application of the NOM simulation model needs to run for a long time, often for days. Several aspects of the simulation model have been analyzed, including runtime optimization, database access, objects usage, and parallel and distributed computing.

The NOM simulation model possesses the characteristics which typical scientific applications have. These techniques and analytical approaches can generally be used in other scientific applications. We expect that our experiences can help other scientific application developers to find a suitable way of tuning and achieving higher performance for their applications.

4.2 Profiling tool

OptimizeIt is a Java J2EE performance tuning tool developed by Borland company. It is a commercial tool and the trial version can be downloaded from their Web site ¹. OptimizeIt can display the information about heap allocation, garbage collection, active threads, and class load in the form of graphs. The CPU sampling information is also displayed in a tree structure. These data can be exported to a formatted text file (HTML). Information about object allocation and deallocation can be viewed in a user selected order. OptimizeIt is a powerful tool that detects memory leaks and CPU performance bottlenecks in Java applications. The screen images shown in Appendix C reflect some of the features of OptimizeIt while it was used for profiling the NOM simulation model.

4.3 Data Structure

Selecting the appropriate data structure is important in a scientific application. Different data structures have significant impacts on the performance. No single

¹<http://www.borland.com/optimizeit/>

data structure is appropriate for all situations. The best way to decide which type of data structure to use in an application is to create some benchmarks that reflect how the structure is used in the application.

In the NOM simulation model, each molecule object needs to be held in a collection. Individual molecule objects are accessed randomly at each time step. They can also be added or removed from this collection. A `List` data structure is a suitable choice to store, retrieve, and manipulate molecule objects. Java 2 platform provides two types of `List` structures: `ArrayList` and `LinkedList`. The `ArrayList` is a random access list, and the `LinkedList` is a sequential access list. The same operation has very different performance characteristics for different types of `List`. Position access is an operation used to access objects through its index in the `List`. Position access is a linear-time operation in the sequential access list, and it is a constant-time operation in a random access list. However, the remove operation for the random access list is more expensive than for sequential access list [37].

An algorithm for manipulating random access lists can produce quadratic behavior when it is applied to sequential access lists. In the NOM simulation model, in order to randomize the order of accessing the molecules in a `List`, the `List` needs to be shuffled at the beginning of each time step. The `collections` class in Java 2 platform provides a `shuffle` method to randomize the order of objects in the `List`. The shuffle algorithm used in the implementation has a linear time complexity for random access lists and quadratic complexity for sequential access lists. In order to avoid the expensive operation that would result from shuffling a sequential access list, the `shuffle` method converts the list into an array before executing shuffling and converts the shuffled array back into the list.

Three benchmarks have been created in order to test the performance, using different `List` structures and operations. These benchmarks reflect exactly how the

data structures have been used in the NOM simulation model. In benchmark A, the `MoleculeList` is implemented using `ArrayList`, objects are accessed using `get()`, and `MoleculeList` is randomized using `shuffle()`. In benchmark B, `MoleculeList` is implemented using `LinkedList`, objects are accessed using `get()`, and `shuffle()` is used. In benchmark C, `MoleculeList` is implemented using `LinkedList`, objects are accessed using `iterator`, and `shuffle()` is used. In each benchmark, the add and remove operations are measured, and the overall performance is measured to reflect the NOM application behavior.

Figure 4.1 shows the relationship between different types of `List` with different operations. It also illustrates the behavior of different operations when the `List` size is doubled. In the NOM application, molecules are uniformly distributed on the grid. Doubling the grid size, therefore, will double the molecule number and the `List` size.

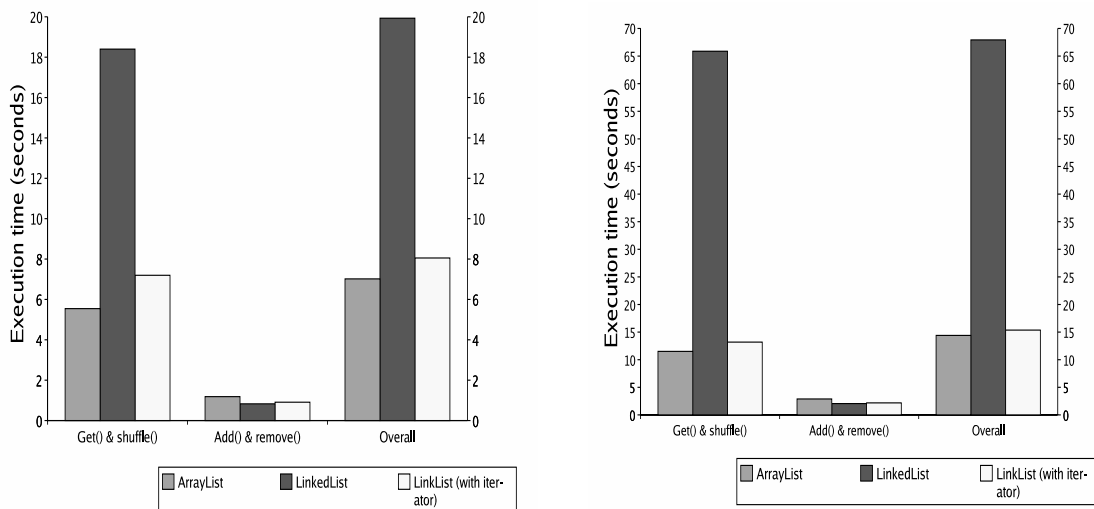


Figure 4.1. Performance comparison of different operations for `ArrayList` and `LinkedList`, the simulation runs for 500 time steps. Left: grid size is 400 X 300. Right: grid size is 800 X 300

The overall performance gain for `ArrayList` has been offset by the add and remove operations. By doubling the grid size, the time for randomly accessing the `LinkedList` increases more than 6 times. For this particular application, the `ArrayList` is the best choice for implementing the `MoleculeList`. It can get a maximum 2.8 speedup in this benchmark with grid size 400 X 300. When the grid size becomes large, the speedup increases.

4.4 Object reuse

Objects play a vital role in Object-Oriented programming, C++, and Java. The Java Virtual Machine (JVM) can automatically manage memory using the “garbage collection” mechanism. Java developers can allocate objects as necessary without considering deallocation, while C++ programmers must manually specify where an object in the heap is to be reclaimed by coding a “delete” statement. Excessively creating objects not only increases the memory footprint and CPU time for garbage collection, it also increases the possibility of memory leak.

Sosnoski (1999) [38] showed that the time for the object allocation in a Java program running on the JVM is 50 percent longer than one using the C++ code. This is caused by the overhead of adding internal information to help in the garbage collection process when allocating objects in heap.

Different JVMs, different versions of Sun JVM and IBM JVM, use various techniques to automatically discover objects when they are no longer being referred to and to recycle the memory periodically. Despite the automatic nature of the garbage collection process, a potential disadvantage is that it adds an overhead that can affect program performance, even in some JVMs such as Sun HotSpot JRE, where the garbage collection job runs in a separate thread.

Although the JVM is responsible for reclaiming the unused memory, Java pro-

grammers still need to put effort into making it clear to the JVM what objects are no longer being referenced [39]. For example, in some situations, a programmer needs to set the object into “null” manually in order to dereference the object. Although the memory leaks that are common in C++ are less likely to happen in Java, they can still occur due to poor design or simple coding errors.

An elegant way of reducing the overhead of objects created and destroyed and of improving the performance is object reuse. It can also reduce the probability of potential memory leaks. In order to reuse a certain type of object, several steps need to be followed:

- Isolating object

Due to the overhead of object reuse, such as managing the object pools, only objects that need to be created and destroyed frequently are compatible with this technology. For a large scale application, using a powerful profiling tool is necessary to help developers detect this kind of object. **Optimizelt** has been chosen in the development process of the NOM simulation model.

In the NOM simulation model, at each time step, a certain number of molecules can enter into the system or leave the system. The molecule objects need to be created and destroyed frequently and they are candidates for potential reuse.

- Optimizing object size

The size of objects not only has an effect on the memory footprint but also on the CPU time. It is worthwhile to estimate the size of a particular object and the number of instances for a given class in memory. A trade off can be made by either reducing the object size or keeping the precision.

- Reinitializing object

Although HotSpot VM radically improved the performance of object allocation and collection, object allocation and instantiation still has a significant cost, especially when the object size is small.

A micro-benchmark is created for testing the time of creation or reinitialization of the *Molecule* objects in the NOM simulation model. In this benchmark, 1000 *Molecule* objects are created, then reinitialized to their original state. The average creation time for these *Molecule* objects is 53.85 million seconds, while the average reinitializing time is 3.9 million seconds.

- Creating object pool

In order to reuse certain types of objects, these objects need to be kept in a collection in the memory, the so-called object pool. An object pool is used to store a free list of objects. Generally, an object pool can be implemented using `Vector`, `LinkedList`, `ArrayList`, or a raw array. Selecting a suitable type of data structure depends on the type of operations used for managing the pool.

In the NOM simulation model, the object pool has been implemented as a First-In-First-Out (FIFO) queue. When a new molecule object needs to be created, the method first needs to check the object pool. If there is an available object in the pool, the object is removed from the queue and reinitialized. If there is no object available, a new object is created. When a molecule object leaves the system, it is added at the end of the queue. The data structure chosen for the object pool implementation is `LinkedList`.

Object reuse is a simple and elegant way to conserve memory and enhance speed. By sharing and reusing objects, processes or threads are not slowed down by the instantiation and loading time of new objects, or the overhead of excessive garbage collection.

4.5 Database connection and database query

The database design and optimization plays a critical role in evaluating the performance and scalability of scientific applications. Most modern databases, called database management systems (DBMS), are built for speed and scalability. They are distributed systems that consist of a number of concurrently running processes, threads, and various machines that work together in an efficient way to deliver fast and scalable data. The basic purpose for a database is to store, manage, and access the data on one or several distributed machines.

For an application written in the Java programming language, communication to a database is accomplished through a Java Database Connectivity (JDBC) driver, with all database Input/Output via SQL (Structured Query Language). Database vendors provide their own JDBC drivers that conform to the common Java API defined by Sun Microsystems. Using JDBC allows developers to change database location, port, and database vendors with minimal changes in code.

Database query operations include data retrieval, insertion, updating, and deletion, as well as table creation and alteration. In a Web-based scientific application, data retrieval, insertion, updating, and deletion are the four most frequently used operations. JDBC offers several advanced techniques that allow the programmer to write high performance queries [40]. These techniques are presented to show the significant impact on performance and scalability.

- Connection Pooling

Establishing a connection to a database is a time-consuming process, especially for a short query. It is therefore reasonable to reuse the connection object that connects to the same database repeatedly. Connection pooling is a runtime object pool that can be used to cache connections. The connection pool is nor-

mally used in a Web-based application connected to a database that a number of clients frequently access. Database vendors and application server vendors provide connection pooling utilities for managing the connection between Web applications and databases in an efficient and reliable way.

- Prepared statements

Query processing is a process for resolving a SQL query. It can be broken down into three basic phases: query parsing, query plan generation and optimization, and plan execution. The query parsing phase is a syntax-checking process for the string-based SQL query that ensures the query statement is legal. If a SQL statement or a set of similar SQL statements need to be executed repeatedly, the cost for the parsing process can be reduced by caching the previously parsed queries. JDBC provides this function with a `PreparedStatement` object. Unlike the `Statement` object, which needs to be sent to a database for parsing each time, the `PreparedStatement` is compiled in advance and can be executed as many times as needed. The speedup of a query varies according to the type of query (`SELECT`, `INSERT`, `UPDATE`, `DELETE`) or the complexity of the query (more than one table involves).

- Batch updates

For a large scale scientific application, the application and the database normally reside on physically distributed machines. Substantial network latency can lead to very inefficient query execution. JDBC provides the batch update approach that can decrease the network latency effect by executing a number of queries in one network roundtrip. The query processing, however, is not necessarily faster when using the batch update approach instead of the `PreparedStatement`. The trade off comes in two forms, (1) batch updates can only

be combined with a **Statement** object and (2) the size of the data needs to be sent from the client to the server in one large roundtrip. The batch updates approach offers more benefits when the network connection is slow.

- Transaction management

In SQL terms, a transaction is a series of operations that must be executed as a single logical unit. When a connection is created using JDBC, the database is set in **autocommit** mode and each SQL statement is treated as a separate transaction. In order to allow two or more statements to be grouped into a transaction, the **autocommit** mode needs to be disabled using *Connection.setAutoCommit(false)*. Treating several operations as a transaction is a safe way to guarantee the integrity of a database. For example, if a bank customer wishes to transfer funds from a savings account to a checking account, the two update operations need to be sent. If these two calls are treated as two separate transactions, then one is successful and the other fails due to the network or other factors. This results in data that is not consistent in the database.

Explicitly commit a group of SQL statements is not only a safe approach, but also has large performance impact because the overhead of the commit operation is reduced. In the NOM model, all the data and information pertaining to the reacted molecules in the system are stored in the database at the end of each time step. The insertions for the data of all the reacted molecules at each time step are treated as one transaction to ensure data consistency in the system.

The NOM core simulation engine connects to the Oracle database using the JDBC thin driver. Figure 4.2 shows the performance comparison for data insertions

using five approaches. The simulation runs for 96 time steps, with a total of 1782 insertions.

The benchmark consists of five cases. In case 1, each insertion for each reacted molecule was treated as a single transaction with a `Statement` object. In case 2, each insertion for each reacted molecule was treated as a single transaction with a `PreparedStatement` object. In case 3, a group of insertions for every reacted molecule in one time step is treated as a single transaction with `Statement` object. In case 4, a group of insertions for every reacted molecule in one time step is treated as a single transaction with `PreparedStatement` object. In case 5, a batch updates approach is applied.

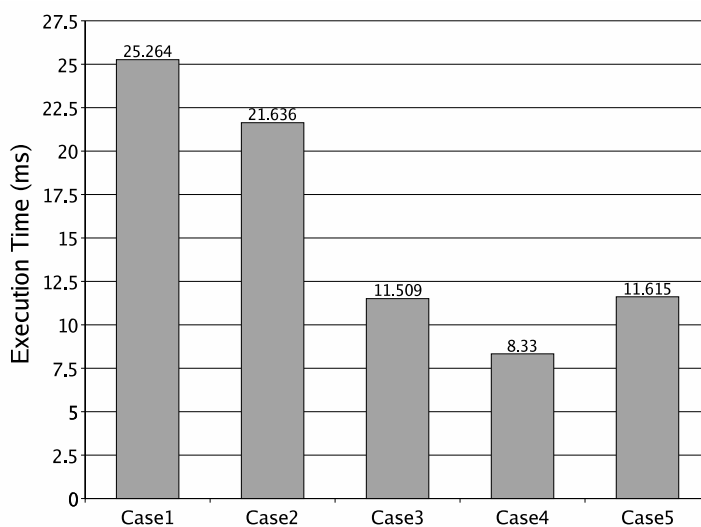


Figure 4.2. Comparison of data insertions using five approaches in NOM simulation model

Case 4, the `PreparedStatement` object with transaction management has the best performance in the NOM application. It offers 3.03 times speedup relative to case1 in this benchmark.

4.6 Parallel data output with Java threads

Querying a database is an I/O bound process, especially when the database server and the application server are on different machines. This is a common architecture design in large-scale scientific applications. For several independent queries, it is more efficient to make use of idle CPU cycles and have these queries executed in parallel instead of one after the other. For database queries that only involve the data read from a database, the parallelism can be easily accomplished by using the Java built-in threads and connection pooling technologies.

Large-scale scientific applications are not only I/O bound processes, but also CPU bound processes. If the application server has multiple processors, multithreading can be used to overlap the computation and communication. More specifically, parallelism can be achieved by overlapping the computation and the I/O. There are, however, trade offs between the simplicity of programming and the performance. When an application not only involves the data-read but also involves the data-write, several programming issues need to be considered to prevent the deadlock and the race conditions.

In the NOM simulation model, large amounts of data need to be written in the database at each time step. The average time for one record insertion using `PreparedStatement` object with transaction management is 4.7 milliseconds. In order to parallelize the data write to the database, a buffer (FIFO queue) is allocated and an extra thread is created. While the computational thread adds the object to the queue, the I/O thread removes the object from the queue and writes the data to the database. If there are no objects in the queue, the data-writing thread executes busy waiting. If the number of objects in the queue is equal to the queue size, the computation thread waits. In order to safely add to and remove from the queue, all the accesses to the queue are synchronized.

The NOM simulation model has been tested and run for various time steps, from 96 time steps to 579 time steps. Refer to 4.3.

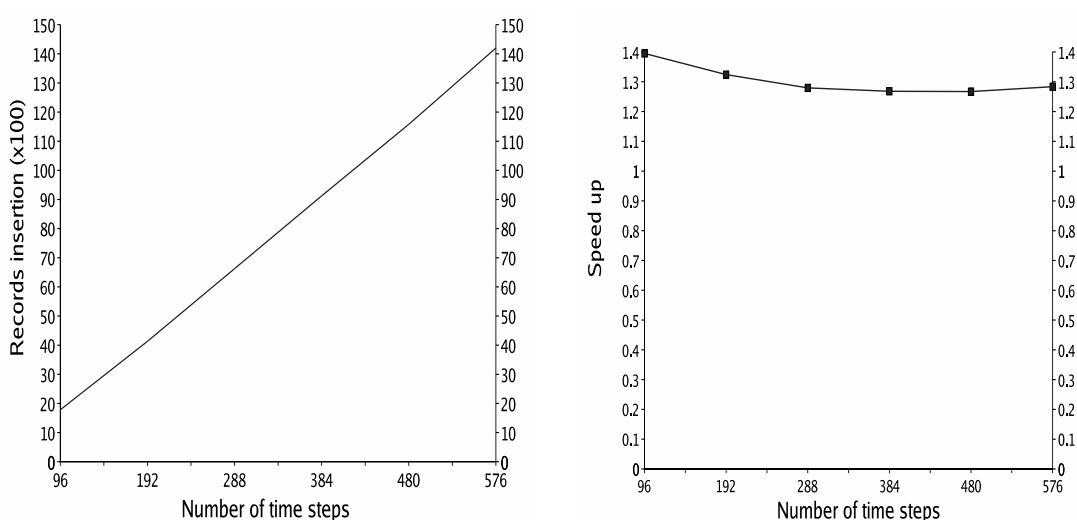


Figure 4.3. Overlap the computation and I/O using Java threads. Left figure shows that the number of data insertion over the time steps. Right figure shows that the speedup of using the second thread for data output

By using a separate thread for data output, there is average 1.3 speedup relative to the single threads model for the NOM application. In the NOM simulation model, the time for computation is more than the time for I/O at each time step. If the computation takes much less time than the I/O, we can consider using more I/O threads to execute data writing.

4.7 Choosing JVM

Various Java Virtual Machines (JVM), such as IBM JVM and Sun HotSpot JVM, have been implemented in conforming to the Java Virtual Machine Specification [41]. The runtime and the compiler are two main parts in the Sun HotSpot architecture. The compiler portion in a JVM translates the bytecode into native

machine instructions in order to improve execution speed. The runtime portion in a JVM includes a bytecode interpreter, memory management, garbage collection, and a low-level task handler [37].

Sun Microsystems implements two types of HotSpot JVM, Client VM and Server VM, to meet different requirements for different applications. Compared with server side programs, client side programs often require a smaller RAM footprint and have a faster start-up time. These two HotSpot JVM share the same runtime portion, but the main difference between them is found in their compiler technologies. The Server VM contains a highly advanced adaptive compiler that includes many of the optimization technologies which are used in C++ compilers [37].

The application of the NOM simulation model has been benchmarked using two different runtime modes of Sun JVM 1.4.1_01 on Redhat Linux 8.0 for 500 time steps and 1500 time steps. Figure 4.4 shows that as the grid size and the time steps increase, Server VM offers higher performance than Client VM. The results show that choosing different JVMs can produce significant differences in the performance.

Ladd (2003) [42] showed benchmark results for both Sun and IBM JVM with versions 1.3.1 and 1.4.1_01 on a Linux platform. According to Ladd, the JVM version 1.3.1 has a higher performance level than version 1.4.1 and the IBM JVM has a better performance level than Sun JVM. Choosing the appropriate JVM for a particular scientific application also involves the consideration of the hardware architecture and the operating system.

4.8 High performance compilers

Traditionally, a Java program is compiled into bytecode using a compiler and a Java Virtual Machine (JVM) is needed to read in and interpret the bytecode.

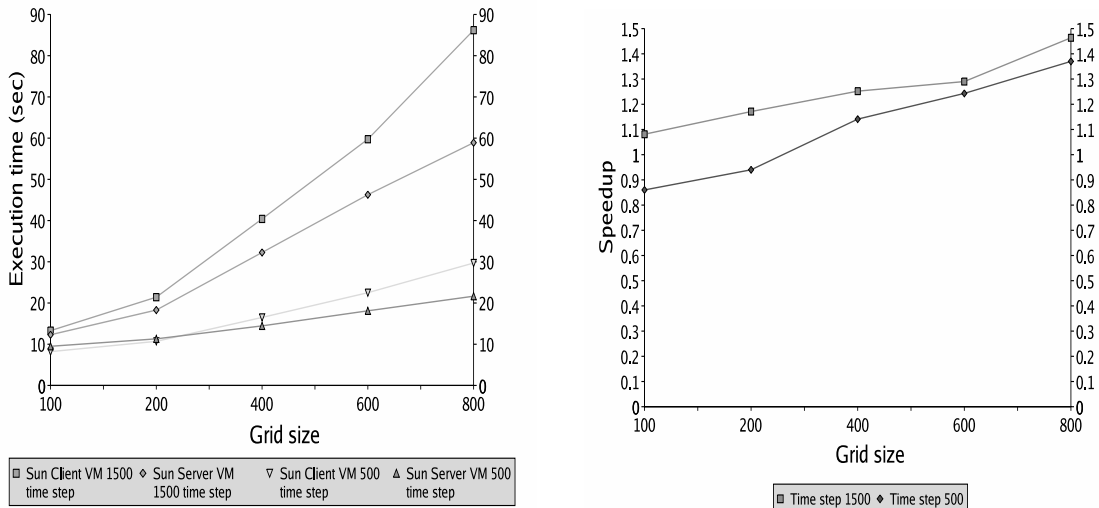


Figure 4.4. Performance comparison for a NOM application run in Sun Client VM and Sun Server VM with different grid sizes.

Jikes², a faster compiler developed by IBM, can translate Java source code into bytecode 10 times faster than the `javac` from Sun JDK 1.4.1_01. Although it does not necessarily generate a faster running code, it can speed up the development process for large applications.

GCJ³, the GNU Compiler for the Java language, can compile the Java source code into either the bytecode or the native code. GCJ has been integrated into GCC. Bothner (2003) [43] discussed the advantages, features, and limitations of GCJ in detail. Ladd (2003) [42] did several benchmarks using GCJ on the Linux platform and showed a performance gain of GCJ over other JVMs.

We did several experiments using GCJ with Java Grande benchmark suite. Table C.1 shows the benchmark results in Appendix C.

The results demonstrate that the performance of code compiled by using GCJ

²<http://www-124.ibm.com/developerworks/oss/jikes/>

³<http://gcc.gnu.org/java/>

compiler is better than the Sun's JVM for most of the applications. The GCJ compiler is a project under development and several limitations still exist. For example, GCJ can not compile the Java program with **swing**. Using GCJ, the Java application with Swarm library can be compiled into native code, but it does not improve the performance.

There are many other runtime environment optimizers and high performance compilers. AlphaWorks is a high performance compiler for Java from IBM alphaWorks. It can be used on OS/2, AIX and Windows NT platform. JOVE ⁴ also can only be used on Windows machines. TowerJ environment ⁵, developed by Tower Technology Corporation, is another example of high performance compilers. It is available for Solaris and Linux platforms.

4.9 Scalability

For large scale scientific applications written in Java, the scalability can be improved using the parallelism programming model. The parallelism model can run with a single JVM in a shared memory multiprocessor system or with multiple JVMs in a distributed memory system. The Java built-in threads mechanism is a convenient method for parallelism implementation in shared memory environments. However, for large scale applications that require large memory and CPU time, distributing the application on multiple JVMs in a distributed memory system with some message passing mechanisms for inter-VM communication is a suitable way to address the requirements.

The standard Java libraries, thread class, is appropriate for using in the parallel programming paradigm in a single JVM environment. Since most scientific applications are CPU bound, in order to avoid the context switching to achieve best

⁴<http://www.instantiations.com/jove/>

⁵<http://www.towerj.com>

performance, the number of threads should match the number of processors in the hardware architecture. Additionally, the thread creation and destroying should be avoided by creating and managing a thread pool.

OpenMP [44], an open standard for shared memory directives, defines directives for Fortran, C, and C++. OpenMP provides a portable and scalable model that offers a simple and flexible interface for developing parallel applications in shared memory systems. JOMP [45] provides a set of OpenMP-like directives and library routines for supporting shared memory parallel programming in Java. It uses Java threads as the underlying parallel model and is most useful for parallelizing scientific applications at the loop level.

For distributed computing, Java provides a communication mechanism using sockets and the RMI (Remote Method Invocation). Java RMI [46] is a message passing paradigm based on the RPC (Remote Procedure Call) mechanism. RMI is primarily intended for use in the client-server model instead of the peer-to-peer communication model. On the other hand, the explicit use of sockets is too low-level to be used to develop a parallel application [47].

In C, C++, and Fortran, an explicit message passing interface (MPI) [48] standard has been defined for supporting communication of an application in cluster environments. MPI is the most widely-used standard for inter-process communication in high-performance computing. MPICH [49], LAM MPI[50] are two successful examples of portable MPI implementations in traditional languages. Programming with MPI is relatively straightforward because it supports the single program multiple data (SPMD) model of parallel computing, wherein a group of processes cooperate by executing identical program images on local data values.

In 1998, a group of researchers of the Java Grande Forum worked on a specification MPI-like application programming interface for message passing in Java (MPJ)

[51]. The current implementations of the MPJ can be separated in two ways: as a wrapper to existing native MPI libraries or written in pure Java. NPAC's mpiJava [52] is an example of the wrapper approach using Java Native Interface (JNI) to execute a native call. The JMPI project [53] implements message passing with Java RMI and object serialization. The jmpj [54] is built upon the JPVM system. MPIJ [55] is a Java based implementation of MPI integrated with DOGMA (Distributed Object Group Metacomputing Architecture). The MPJ implementation in pure Java is usually slower than wrapper implementations for existing MPI libraries, but pure Java implementations are more reliable, stable, and secure [54]. Getov (2001) [47] did an experiment that used the IBM High Performance Compiler for Java (HPCJ), which generates native code for the RS6000 architecture, to evaluate the performance of MPJ on an IBM SP2 distributed-memory parallel machines. The results show that when using such a compiler, the MPJ communication components are as fast as those of the MPI.

4.9.1 Parallel implementations

The scalability of the NOM simulation model involves two aspects, the required total number of simulation time steps and the grid size. Two parallelism programming models are implemented for the NOM simulation model. The Java thread version is implemented using build-in Java threads and runs on a single JVM. The distributed memory model is implemented by using mpiJava library with LAM MPI.

In the sequential implementation model for simulating the NOM complex system, the behaviour of individual molecules are simulated using agent-based modeling approach. Molecules reside in a cell on a 2D grid and individual molecules can be transported through the soil medium via water flow. At each time step, molecules can move from one cell to the other when a random event occurs. The application

behavior along with the grid size and the time step is shown as C.6 in Appendix C.

Java threads version

In order to parallelize the programming model, the original grid has been equally separated into two subset grids (e.g., a 800X300 grid is separated into two 400X300 grids). Two threads are created, each thread has its own grid object to place molecules and a collection to hold molecules. In each time step, the computation for individual molecules is executed on the two threads concurrently.

When one molecule moves across the boundary, the molecule is removed from the grid and placed into a local buffer in the current thread. At the end of the time step, a **Barrier** is used to synchronize these two threads at this point. After all the threads reach this state, one of the threads executes the exchange operation by maintaining the state of two grids and two *MoleculeLists*. Threads have been synchronized before this thread finishes the setting of boundary condition. Figure C.4 in Appendix C shows the design.

MPJ version

In the distributed memory model, each processor runs its own copy of code. The **ModelSwarm** object contains a subset of the original grid. These subsets of the grid have equal size in order to ensure the computational balance of each node in the cluster. The basic design model is similar to the Java thread model. Each node in the cluster machine processes its own computation of the subset of the grid. When a molecule crosses the boundary, it is added into the local buffer. `MPI.COMM_WORLD.Barrier` is used to synchronize these processes at the each time step. Molecule objects that cross the boundary are sent to their neighbor grids using blocking send and receive modes, `MPI.COMM_WORLD.Sendrecv`, `MPI.COMM_WORLD.Send`, and

MPI.COMM_WORLD.Recv. Figure C.5 in Appendix C shows this design. The molecule list and grid on each machine are updated after all the sending and receiving are finished.

4.9.2 Performance results

In order to measure the performance of the parallel implementations, a Linux cluster has been built. The cluster consists of 4 PCs. Each PC has dual 650 MHz Intel processor running the RedHat Linux 8.0 Operating System. The Java codes are implemented and compiled using SUN's 1.4.1_01 Java Development Kit and executed on SUN's Java Virtual Machine.

Figure 4.5 presents the results for the sequential version and Java thread version for the NOM application that runs for 1500 time steps and various grid size. Both versions ran on a single dual Intel computer.

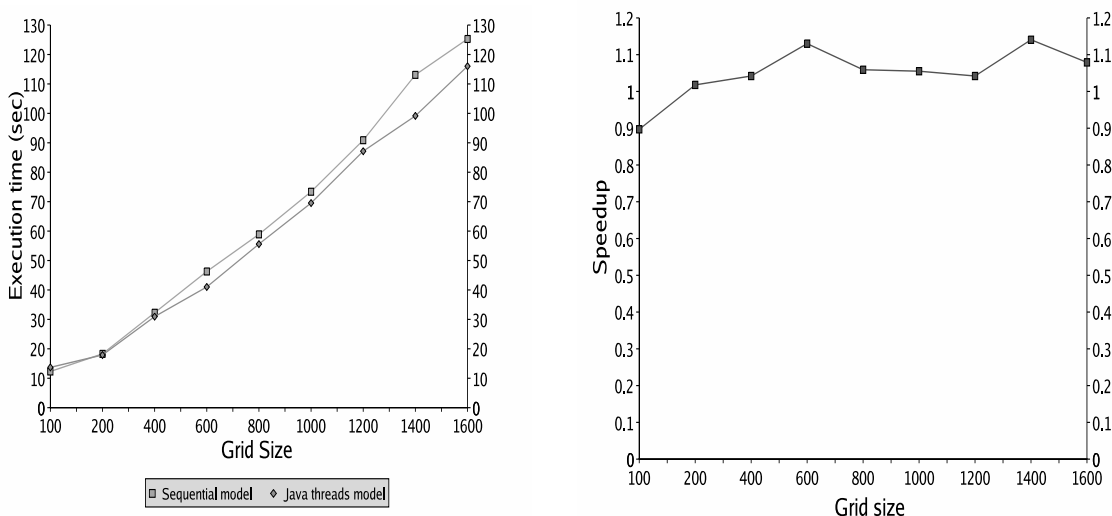


Figure 4.5. Compare the performance between sequential programming model and Java thread model in the NOM application (one thread vs. two threads on a single dual CPU computer).

The experiments for parallelism of the NOM simulation model using the distributed memory model are made on a cluster of four PCs. LAM MPI 6.5.9 and mpiJava have been used to build the execution environment. The NOM application runs on 2 and 4 machines.

Figure 4.6 shows that the performance comparison between the sequential programming model and the MPJ model that run on 4 nodes in the cluster. These two models both ran for 500 and 1500 time steps. This figure shows that the communication, the grid and *MoleculeList* maintenances offset the performance gained by distributing the job to different computers when the problem size is small. As the problem size grows (larger grid size or longer time steps), however, a performance improvement appears by distributing the job to. Compared to the ideal performance gain of a factor of 4, the efficiency is low.

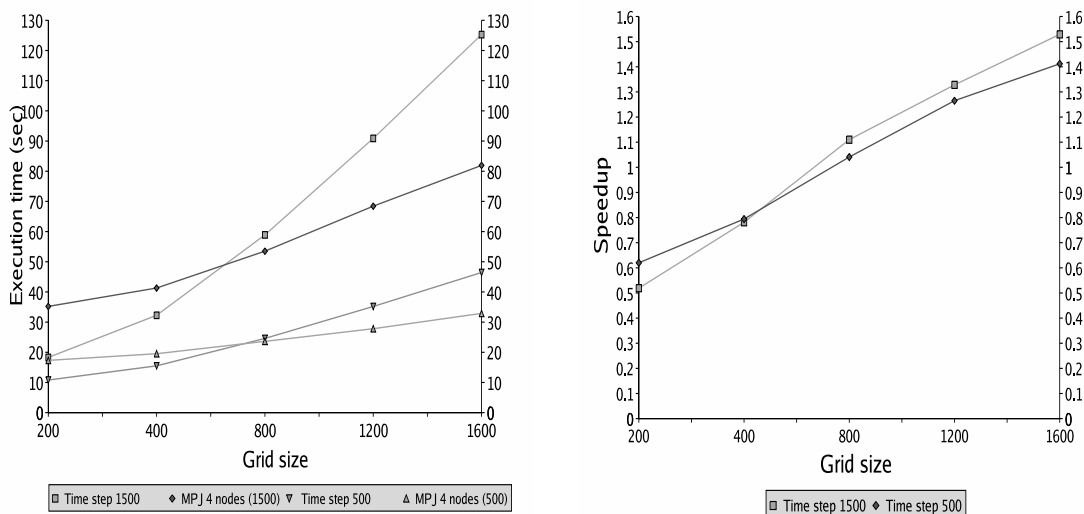


Figure 4.6. Comparison of the performance between sequential programming model and MPJ model that runs on 4 machines for 1500 time steps.

Figure 4.7 illustrates that the job has been distributed on four nodes in the

cluster machines and run for 500 time steps. There is no synchronization between nodes. Instead of sending the molecules which cross the boundary to other nodes, they are wrapped to the other side of this subset grid. This figure shows that as the grid size increases the speedup is closer to the ideal linear speedup of 4.

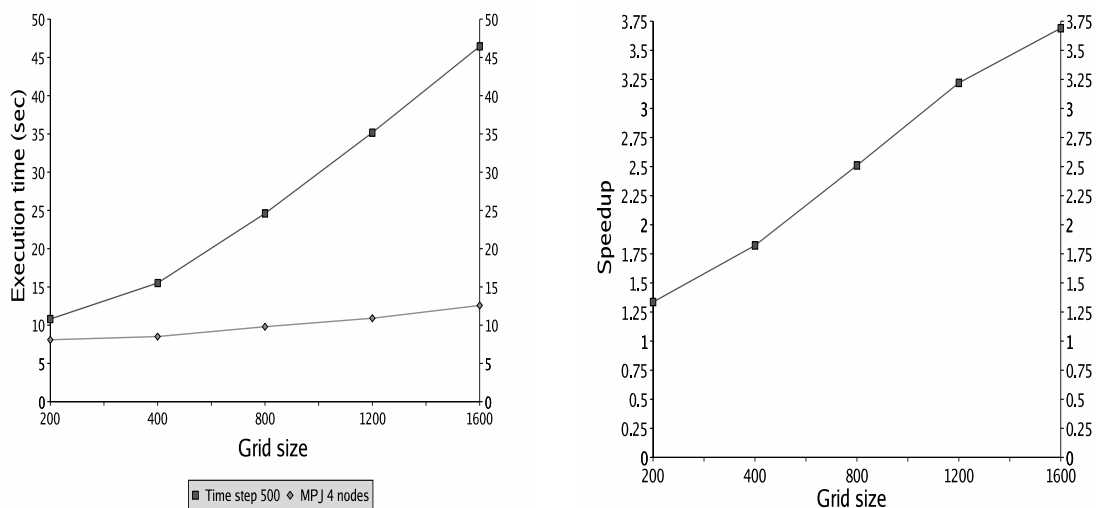


Figure 4.7. Comparison of the performance between sequential programming model and MPJ model on 4 machines without synchronization and molecule exchange.

4.10 Conclusion

In this chapter, several approaches for exploring the performance and scalability improvement of a typical scientific application, the NOM simulation model, has been presented. These approaches are summarized in Table 4.1. The speedup varied by the problem size and different situations. The speedup shown in the table came from the benchmarks described in the previous sections. All the numbers of speedup list in the table show the approximate highest performance gain in each benchmark.

Besides the approaches that listed in the Table, using a native code compiler

Table 4.1. Summary of the performance improvements

Approaches	Speedup	Comments
Data structure and algorithms	2.8	using <code>ArrayList</code> provides higher overall performance than using <code>LinkedList</code> in the benchmark.
Object reuse	-	reduce the memory footprint, the garbage collection cycle, and the overhead of object allocation and deallocation. The performance gain is relatively small compare to the overall computation time in the NOM simulation model.
JDBC	3	JDBC tuning can effect the performance of data I/O.
Parallel data output	1.3	overlap the computation and I/O using Java threads can improve the performance.
Java runtime	1.4	Sun Server VM has higher performance than Sun Client VM for large-scale scientific applications.
Java threads model	1.1	performance of Java threads model largely depends on the JVM implementation and how efficient the operating system handle the threads.
MPJ model	1.5	distributing the job to 4 nodes in a cluster has relative larger performance gains when problem size is big and the communication between nodes is minimized. However, it is still much lower than the ideal performance gain (4).

that can compile the Java source code to native code can increase the performance for some applications.

Program profiling is a crucial step for high performance computation in Java-based applications. Two major aspects, CPU time and memory usage, need to be monitored.

Selecting the appropriate runtime environment for a particular application is important. Besides the JVMs that evaluated here, IBM JVM is also valuable to be evaluated.

Using Java built-in threads to parallelize the Java applications is a convenient approach. How much performance can be gained from this parallelism depends on the JVM implementation and the efficiency of the operating system handling the Java threads. The experiments that we did are on a dual CPU PC with Linux operating system. It is valuable to extend these experiments to Windows operating system or Solaris operating systems.

Distributing the job on multiple machines in a cluster environment is an efficient approach for large size problems. However, the communication among nodes and the grid and list maintenance offsets the performance gain. In order to avoid this overhead, instead of synchronizing all the processes at each time step, we can consider to synchronize them at every 5 time steps or more. For this particular application, it is also possible to distribute the job on multiple machines using MPJ and combine the results at the end of simulation in the database. There is no communication at all after the job is distributed. However, validation is necessary for these two approaches.

CHAPTER 5

A COLLABORATORY BUILT UPON J2EE PLATFORM FOR SUPPORTING SCIENTIFIC COLLABORATIONS

5.1 Introduction

A collaboratory is defined as “an open meta-laboratory that spans multiple geographical areas with collaborators interacting via electronic means—working together apart” [56]. It is a merger of the words “collaboration” and “laboratory” first coined by Wulf (1996) [57]¹. Collaboratories intend to promote collaborations among scientists in various research areas across geographic boundaries. They allow scientists to share the expensive research instruments and the data and information across distributed sites, to exchange the personal experience, and to accelerate the development and dissemination of knowledge. Several collaboratories have been developed for various scientific purposes, including Disel Collaboratory [58] in Combustion science, BioCoRE² in Biology science, and the EMSL Collaboratory³ in Environment science.

Creating a collaboratory involves integrating existing software and hardware tools to build a seamless environment where scientists can work together virtually. A number of commercial and research tools have been developed to allow Web-based collaboration. These tools can be separated into two categories, synchronous tools

¹William Wulf is a University Professor of Computer Science at the University of Virginia. He was elected as President of the National Academy of Engineering in 1997.

²<http://www.ks.uiuc.edu/Research/biocore/>

³<http://www.emsl.pnl.gov:2080/docs/collab>

and asynchronous tools. Electronic mail, discussion boards, and electronic notebooks are typical asynchronous tools, while audio/video conferencing, chat boxes, and the white board are synchronous tools. However, some collaboration tools, such as audio/video conferencing, rely on high performance hardware support. Audio/video conferencing allows scientists to see and hear each other and to communicate with each other as if face-to-face. Although audio/video conferencing greatly supports scientific interaction, it is adopted in very few laboratories due to the cost, hardware restrictions, low network bandwidth, and poor quality. Therefore, which tools are suitable to build into a laboratory depends on whether these tools can add value to the scientific interaction process or not. It also heavily depends on the requirements of end-users, the collaborators.

The NOM project involves researchers from different research areas, including chemists, ecologists, biologists and computer scientists. As in the common case, these researchers are also geographically separated. In order to enable scientists from different geographical areas and research areas to work together virtually, a Web-based laboratory for supporting scientific collaborations in the NOM research community is built based on J2EE platform. J2EE architecture simplifies the application development process by automatically handling many details of application behavior, without complex programming. The underlying J2EE architecture enables easy development of robust, secure, and platform independent Web-based applications, integration of collaboration tools, and maintenance of user interfaces. The NOM laboratory provides capabilities for scientists to share the computational resources (including large-scale databases and a high performance simulation cluster), the analysis tools, the simulation results, the data and information, and to communicate with each other through a wide range of tools. The NOM laboratory also provides a XML-based Markup Language, NOML, which is used to manage

molecule information and simulation configurations. Additionally, NOML is used to build the XML-based Web components to support the collaboration. Using the universal XML-based data representations enhances component reuse, enables data sharing, and facilitates Web services development.

In this chapter, an overview of the NOM collaboratory is introduced in section 2. NOML, a XML-based format for describing the molecule structure and the simulation configuration, is presented in section 3. The implementation of several collaboration components is described in section 4, and a conclusion is drawn in section 5.

5.2 The NOM Collaboratory overview

The NOM collaboratory provides the following functions:

- Distributed computational resource utilization: After users invoke their simulations through a Web interface, computational resources on the remote sites are allocated transparently by a job manager.
- Data analysis: Users can view their simulation data that are generated by the NOM simulator from the Web. These data and information are represented in various types of graphs (bar charts, pie charts, line charts) and statistical reports by employing data query and data mining technologies.
- Information sharing: Users can share the results of their simulation, the molecule definitions, and the simulation configurations through web interfaces and a search engine.
- Data repository: Oracle databases are used to store the internal data that are generated from the NOM simulator. Additionally, external data, including

publications, technical reports, and other forms of dissemination, which are uploaded by scientists, are also stored in the database.

- Secure access: Users do not have the same level privilege to access all the tools in the NOM collaboratory. Some particular tools, such as the “Molecule validator”, the “NOM simulator”, can only be accessed by authorized persons.
- Communication tools integration: A discussion board and a chat room are integrated into the collaboratory in order to facilitate communications between users.

The NOM collaboratory is developed upon the J2EE architecture and JSPs, Servlets, JavaBeans and JDBC technology are applied. In order to easily maintain the programs, add new components, and reuse existing components, the design of the collaboratory follows the MVC design diagram. Figure 5.1 illustrates the Web-based interfaces and components in the NOM collaboratory. Users can access these tools and services in the collaboratory environment through multiple Web-based interfaces from a standard Web browser.

The NOM collaboratory includes the following components:

- Web-based NOM simulator. Users can access the on-line NOM simulator through an intelligent Web interface using a standard Web browser. They can configure their simulations with multiple HTML and JSP pages and invoke their simulations on the remote computer cluster. The intelligent web interface provides a set of intelligent agents to find similar simulations, stop and resume simulations, predict running time of simulations, and send email to users after simulations are finished. These agents guide users to use the computational resource more efficiently. A user can submit one or more simulations and several users can submit their simulations simultaneously. A

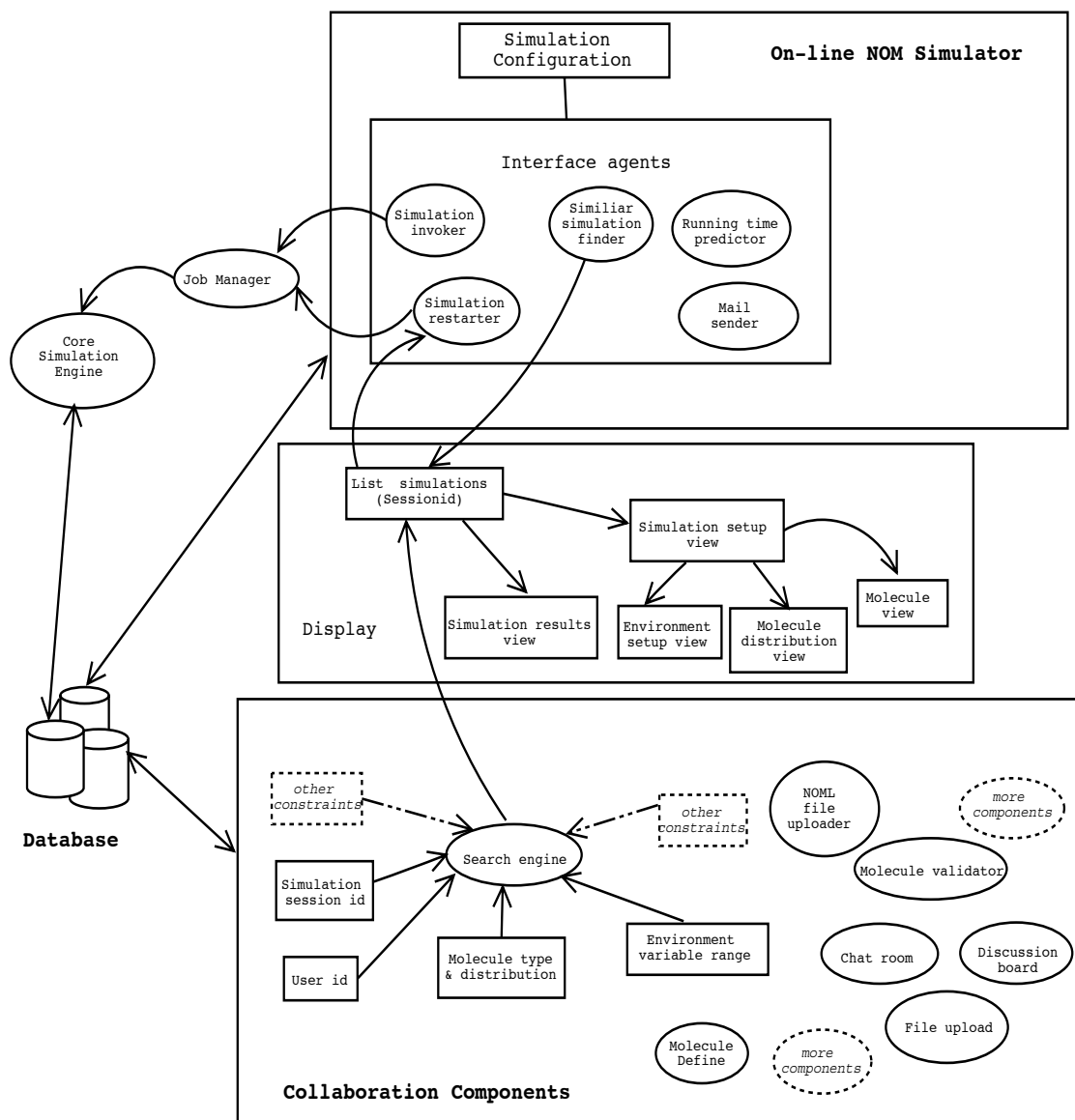


Figure 5.1. Components of the NOM collaboratory environment for supporting cooperation between remotely located scientists

simple job manager assigns tasks on several simulation servers to achieve load balancing.

- Search engine. An ad-hoc query-based search engine provides users maximum flexibilities to search any information in the data repositories.
- NOML file uploader. This Web service is built to support uploading of NOML format files. These files are parsed on the fly and the parsed data pieces are stored in the database.
- Molecule editor. Users can access the Molecule editor to define the new molecule structure through a Web-based interface or upload a NOML format file.
- Molecule validator. Authorized users can access this tool from the Web interface to validate the newly created molecule types or add new molecule types for public usage.
- Chat room and discussion board. These two collaboration tools are integrated into the collaboratory in order to facilitate the communication between users.
- File upload. This tool is integrated to exchange and disseminate the knowledge.

5.3 XML-based NOM Markup Language (NOML)

Extensible Markup Language (XML) is a standard, simple text document that conforms to a specification defined by the World Wide Web Consortium (W3C) at <http://www.w3.org>. Compared with HTML, XML allows user-defined tags to be used to represent any kind of information. Users can develop their own XML docu-

ment formatting, content, and specifications by defining a set of tags and attributes to represent their own data.

Document modeling involves creating a specification that lays out the rules for how a document can look. The most common way to model documents is with a document type definition (DTD). A DTD is a set of rules or declarations that specify tags and the content of these tags. In order to validate a XML based document, a reference to a particular DTD needs to be added at the top of the document. Another document-modeling standard, XML Schema, introduces more powerful data type checking which make it possible to find errors in content as well as the usage of tags.

An XML document can be created using either a text editor (*vi*, *emacs*, or *Notepad*) or a graphical editor (*Arbortext Adept*⁴, *Adobe FrameMaker+SGML*⁵.) In order to use the data in a XML document, a parser is needed to validate the XML document with a DTD or an XML Schema and parse the document into data pieces. SAX (Simple API for XML) and DOM (Document Object Model) are two common APIs for parsing the XML document. A Java application needs to wrap this parsing behavior and provides methods to manipulate XML data. McLaughlin (2001)[59] and Ray (2001) [60] provide more details about XML.

Several XML-based Markup Languages are developed for different XML applications, such as Mathematics Markup Language⁶ (MathML) for encoding equations, Scalable Vector Graphics language⁷ (SVG) for storing graphics, and Chemical Markup Language⁸ (CML) for managing molecular information.

The XML-based model ensures uniformity across components and helps to ab-

⁴<http://www.arbortext.com>

⁵<http://www.adobe.com.au/products/frameMaker>

⁶<http://www.dessci.com/en/reference/webmath/tech/mathml.htm>

⁷<http://www.w3.org/Graphics/SVG/Overview.htm8>

⁸<http://www.xml-cml.org/>

stract the component structure and implementation from the component interface. The NOM collaboratory provides a set of standardized XML DTD definitions that forms an ontology for the simulation configuration and the molecular structure definition. This standard format can reduce efforts of information interchange between users. Users can create XML documents, conforming to the standardized NOML DTD definitions, to describe simulation configurations and molecular structures. Once users have stored the information in NOML format files, they can attach these files into email or documents and share the data with others. Users can also upload the file using the NOML uploader service that is embedded in the NOM collaboratory. Without using the Web-based configuration interface, a simulation can also be invoked after a NOML-based configuration file is uploaded. Three DTDs are defined in NOML to describe various data information.

(1) *environment.dtd* describes the format of the environment information. Users can define a set of environment parameters using tags that are defined in the DTD and indicate the owner as well as the accessing privilege.

(2) *molecules.dtd* describes the format of the molecular information. Users can define more than one new molecule structure in one XML file as well as the owner of these molecule definitions.

(3) *setup.dtd* describes the format of the simulation configuration. Users can define one set of parameters, including environment parameters, molecule types, and molecule distributions, to configure a simulation.

These DTDs and sample NOML-based files are given in Appendix B. Users can view the DTDs and download the NOML examples from the Web. The simple tree structures of the sample documents in NOML format are shown in Figure 5.2.

NOML can be extended to support various Web services.

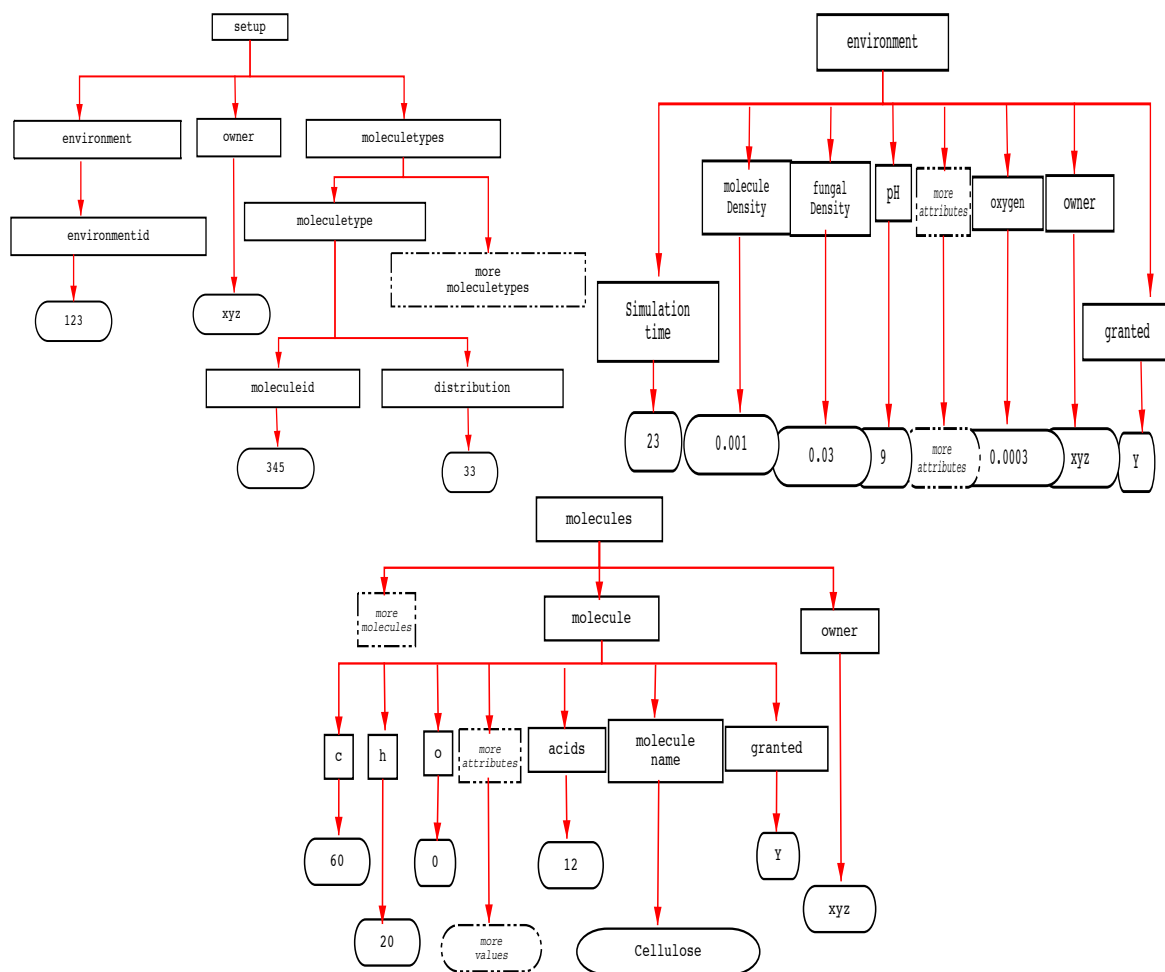


Figure 5.2. Tree structured view of three types of documents in NOML format.
 Note: not all the elements are shown in the graph.

5.4 Components design and capabilities

All components of the NOM collaboratory prototype are implemented within a three-tier architecture by following the MVC design diagram. Servlets process the incoming requests and manage the session and then call the corresponding JavaBeans components to handle the application logic. Finally JSPs generate the dynamic presentations to users. Components can be easily added, removed, and modified without changing any other features. The page layout can be changed without modifying the JavaBeans.

When researchers are geographically far apart, network latencies and limited bandwidth can limit the usability of software tools. In order to accomplish fast data access, the database accessing information is cached using Java Naming and Directory Interface (JNDI) and the database connection pool is used in the components design.

5.4.1 Search engine

Collaborative research demands sharing the knowledge and experiments to reduce the time for producing new results. Executing a large-scale scientific simulation is a time-consuming task. The NOM collaboratory offers a capability to allow users to share the configurations and results of their simulations by providing a search engine. Users can take advantage of the simulation's results that were completed earlier instead of doing the same simulation over and over again. By accomplishing this capability, the instruments and computational resources can be used more effectively.

Users can access the search engine through a Web interface by providing several search conditions. These conditions include the range for each environment variable, the molecule types and distributions, simulation sessionid, userid, and so on. Users

can either leave all the fields as blank or make some reasonable combinations and submit the request form to the server. The remote server processes the request and returns a set of simulation *sessionid* associated with simulation configurations that meet with users' requests.

Users can view data information corresponding to each *sessionid* in more depth by clicking links. Clicking the link with one *sessionid*, users can view the environment setup, molecule types and distributions of the simulation. Users can view the molecule structure (including atoms and functional groups), molecule creation date, owner, and molecule name by simply clicking a link associated with the molecule's *typeid*. Users can also view the analysis data of a particular simulation. If the simulation results are valuable for users, they can extend the simulation by entering extra simulation time and resuming the simulation from the Web interface.

The design of the search engine includes a Java Servlet, several JSPs, and JavaBeans. Requests from users are processed by the Servlet and a set of SQL query statements are generated. These statements are sent to the JavaBeans and are processed by communicating with a remote database through JDBC. The query results are generated dynamically with JSPs and displayed on the client side Web browser.

By following MVC design architecture, the tasks for presenting, controlling, and modeling are separated. The same presentation layout can be shared in both the NOM simulator and the search engine for displaying simulation information.

5.4.2 NOML file uploader

NOML file uploader provides an interface for users to upload a set of NOML format files. Users issue a Multipart Request by submitting a HTML form from the client side. On the server side, the O'Reilly Multipart request and parser package,

which can be downloaded from the [servlets.com](http://www.servlets.com) website⁹, is used in a Servlet. Instead of saving files in the disk on the server, the Servlet reads incoming files and parameters and parses the incoming file's information to a XML FileInputStream. The FileInputStream is fed into a corresponding JavaBean according to the content of the incoming files. JavaBeans are implemented to validate and parse the NOML documents, establish a connection to the Oracle database, and write the data in the database. The Servlet returns the information about the *moleculeid*, *enviornmentid*, or *sessionid* to users for further reference. The design model is shown in Figure 5.3.

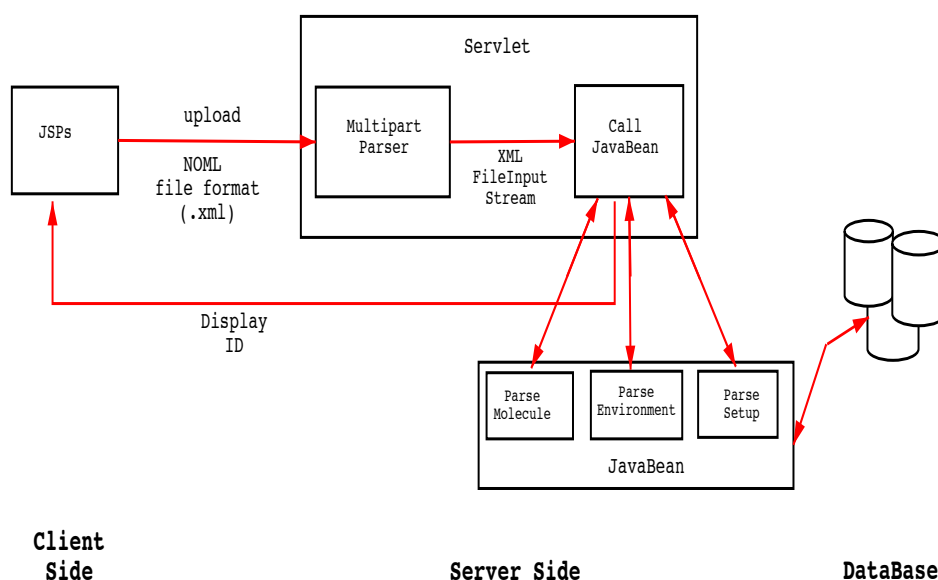


Figure 5.3. The design model for NOML file uploader

Users can access the NOML file uploader by providing a file name. If users upload a file that contains the simulation configuration parameters, they can either upload the file or upload the file and invoke the simulation. If users only choose to upload the file, the Servlet returns a *sessionid* for further reference. Otherwise, the file is uploaded and the simulation is sent to the job manager to be invoked. If users

⁹<http://www.servlets.com/cos/index.html>

upload a file that defines one or more molecule types, *moleculeids* are returned to users for further reference. If users upload a file that defines a set of environment parameters, *environmentid* is returned to users for further reference.

5.4.3 Molecule editor

The “molecule editor” tool, embedded in the on-line NOM simulator, can only be used during the simulation configuration. In order to provide a more flexible way for users to define new molecule, a separate “molecule editor” is provided.

Several simple rules are encoded to automatically validate the newly defined molecule in order to prevent the specification of chemically impossible molecule structures. For example, a simple rule is defined as follows:

If the “benzene” ring structure, shown in Figure 5.4, is part of the molecule structure, then the total number of carbons (atom C) in this molecule structure definition must equal to or larger than 6 times the number of “benzene” rings. Also, the number of carbon-carbon double bonds must be 3 times the number of “benzene” rings.

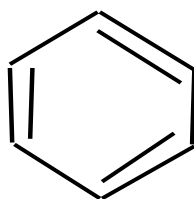


Figure 5.4. The benzene ring structure

Besides providing a Web interface for the molecule definition, users can also define new molecule types by uploading a NOML format file to the server. The uploaded file is parsed on the fly and the data is stored in the database through the NOML file uploader.

5.4.4 Molecule validator

Before one molecule type definition is ready to be shared by the NOM community, this molecule definition must be validated. However, the molecular validation is not only an objective work that can be accomplished by a piece of software, but also a subjective work that needs human involvement.

General rules that can be encoded in software are not sufficient for validating a newly created molecule. Therefore, an administrator role is provided to an authorized person who can validate newly defined molecules manually.

The authorized user, a chemist in the general case, can access the validator by clicking a button. A set of molecules that are indicated for public usage by their owner are displayed. The user can click the *moleculeid* to view the molecule structure and click a button to validate a particular molecule type. Only validated molecules can be shared by every user who participates in the collaboratory. Non-validated molecules can only be used by their owners.

An authorized user can also define the molecule type from the separate molecule editor or upload a NOML format file to the server. This user can remove a particular molecule from the NOM system.

5.4.5 Chat room and Discussion board

In order to facilitate the communications between researchers, a chat room (a synchronous collaboration tool) and a threaded discussion board (an asynchronous tool) are integrated into the collaboratory. Users can enter into the chat room through a Web interface by providing userid and password. The threaded discussion board help users to easily trace the discussion contents about one topic.

5.4.6 File sharing

Collaborative research often demands exchange of large data files and documentation for faster dissemination of the knowledge. Users can upload research papers and data sets through a Web interface into a shared place residing on the remote servers. These files are accessible to other researchers after these files are uploaded.

5.5 Conclusion

In this chapter, a prototype of a Web-based collaboratory environment that supports new ways for conducting scientific research on NOM is presented. The NOM collaboratory deploys collaborative technologies to NOM researchers geographically separated or in different disciplines. The environment integrates a number of collaboration tools, including NOM simulator, molecular validator, molecule editor, search engine, file sharing, discussion board, and chat room. The architectural design of the NOM collaboratory makes it easy to add, modify, and remove components in the collaboratory according to the requirements from scientists.

At this stage, all the tools integrated into the NOM collaboratory are “home-made” tools using J2EE technology. In the future, some powerful third-party tools, such as video/audio conferencing, which relies on a fast and efficient high bandwidth network technology and advanced hardware (e.g., Internet2), may be integrated. Oracle Collaboration Suite¹⁰ offers integrated email, voice mail, phone, fax, scheduling, calendaring, meeting management, and file management. It could be considered for integration into the NOM collaboratory.

An XML-based NOM Markup Language, NOML, which provides a standard definition for the molecule and simulation configuration, is defined. The NOML not only supports the data communication between users but also give us flexibility of

¹⁰<http://otn.oracle.com/products/cs/content.html>

extending the on-line NOM simulation. It also supports building new XML-based Web services in the near future. A set of new simulation models for the NOM study will be built and integrated into the collaboratory. The communication between these models can be achieved by the extension of the NOML. The NOM collaboratory allows users from different places to collaborate by sharing their simulation data and configuration as well as conduct discussions in this NOM community.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

The NOM simulation model, a Web-based scientific application model, is presented in this thesis. The NOM simulation model was built using J2EE technology. Compared with most other simulation models that are currently available for scientific applications, our Web-based scientific simulation model eliminates the need for installation of software on the user's computer and difficulties that may accompany such installation. Users can remotely access the model using any standard Web browser, such as Netscape or Internet Explorer. It takes advantage of the high-speed network, advanced computer technology (J2EE), and a scalable Web-based database management system (ORACLE) to facilitate analysis of large datasets produced during simulation runs. The NOM simulation model is the first Web-based simulation model used in the study of NOM. It also serves as an example of E-science.

Web-based scientific applications are intended to focus on the end users and collaboration between them. The intelligent Web interface guides users in the configuration of their simulations. Several intelligent components are built into the Web interface to facilitate the entry of reasonable parameters. They also provide the functionalities to reduce the simulation time and efficiently use the computational resources. The sending email agent can automatically send a user email after

the simulation is done. The running time prediction agent can help the user adjust the configuration of the simulation. The similar simulation finder allows the user to take advantage of simulation results produced earlier, either by the same user or by other users. The automatic restarting agent can restart a particular simulation from the point where it stopped.

A Web-based “NOM collaboratory” using J2EE technology is built to promote collaborative research among scientists studying NOM, even if they are geographically separated. It allows scientists to share the expensive computational resources, data, and information generated across distributed sites. It also allows scientists to exchange personal experiences, thus accelerating the dissemination of information. A set of collaboration tools are integrated in this collaboratory. These include a chat room, discussion board, and file uploader. The XML-based Markup Language, NOML, was developed to provide a fundamental standardized data format for the development of Web services. A set of Web services based on the NOML were developed to support data sharing between users and applications.

The stochastic NOM core simulation engine is built using an agent-based modeling approach. It is the first simulation model in NOM research that uses such an approach to model NOM complex systems. The model explicitly treats NOM as a heterogeneous mixture. Doing so allows the global distribution of physical, chemical, and biological properties to be discovered and predicted on the basis of the actions of individual molecules in a bottom-up approach. The NOM simulation model is intended to provide scientists from various disciplines a powerful tool for supporting their NOM studies.

In order to adequately represent the complexity of NOM, the number of molecules in the simulation must be sufficiently large. As the number of molecules in the simulation increases, however, there is a decrease in the simulation’s performance.

Several potential aspects that can lead to decrease the performance are investigated. Several approaches are applied to improve the performance and the scalability for the NOM simulation model. The analysis for the NOM simulation model is conducted from the CPU usage and memory footprint.

6.2 Future work

The NOM simulation model currently serves as a testbed for scientists. They can compare their experimental data with results produced from their simulations. In the current stage of development, the environmental conditions remain steady during the entire process of the NOM evolution. In the future, the model's environmental conditions may vary due to different seasons and rainfall.

The NOM simulation model divides the total time period into a large number of equal steps. At each step, each molecule is evaluated whether or not a reaction occurs. The time step size in the current model can be tuned for efficiency. While the small size of the time step increases the accuracy of the simulation, the large size of the time step reduces the number of computations thereby contributing to a decrease in the overall run time of a simulation. Since the probability of an individual reaction is low in some cases, a more efficient approach (discrete event) can be developed to treat the reaction time as a random variable.

There are a total of ten popular types of chemical reactions that can be modeled in the NOM simulator so far. Many more reactions exist in the real world and we may add more reactions to the simulator in the future. In addition, a general XML-based chemical reaction formula definition can be added into NOML. A Web service can be provided to allow users to input the reaction formula from a Web interface or upload a XML-based file into the server. A set of new simulation models and tools can be built and integrated into the collaboratory. The communication

between applications can be done with the extension of the NOML. Furthermore, additional tools that support the collaboration between users can be integrated into the NOM collaboratory in order to meet the needs of scientists.

For large-scale NOM applications, the distributed memory model is suitable for performance improvement. Communication and grid maintenance, however, offset the improvement in performance when the problem size is not sufficiently large. A more efficient approach for handling communication needs to be explored from both the software and hardware perspectives. Instead of synchronizing all the processes at each time step, the processes can be synchronized every five time steps, or more. Alternatively, the job can be distributed to the cluster machines when the simulation starts and all the results can be merged into a database. The performance can be largely improved using this approach. Validation for these approaches can be accomplished by comparing the results with the sequential model. The performance for a particular scientific application largely depends on the operating system and the hardware architecture. Profiling and performance tuning of scientific applications written in Java are necessary for achieving high performance.

APPENDIX A

GLOSSARY

DOM	Document Object Model
DTD	Document Type Definition
EIS	Enterprise Information System
EJB	Enterprise JavaBeans
ERP	Enterprise resource planning
J2EE	Java 2 Platform, Enterprise Edition
JCA	Java Connectivity Architecture
JDBC	Java Database Connectivity
JMS	Java Message Service
JMX	Java Management Extensions
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JSPs	JavaServer Pages
JVM	Java Virtual Machine
MVC	Model-View-Controller
NOML	XML-based NOM Markup Language
RDBMS	Relational Database Management Systems
SAX	Simple API for XML

APPENDIX B

NOML

```
<!-- Environment DTD environment.dtd -->

<!-- Root element -->
<!ELEMENT environment (simulationtime, microbialDensity,
fungaldensity, pH, temperature, pkw, oxygen, lightIntensity,
moleculeDensity, owner, granted)>

<!-- Basic element -->
<!ELEMENT simulationtime (#PCDATA)>
<!ELEMENT microbialDensity (#PCDATA)>
<!ELEMENT fungaldensity (#PCDATA)>
<!ELEMENT pH (#PCDATA)>
<!ELEMENT temperature (#PCDATA)>
<!ELEMENT pkw (#PCDATA)>
<!ELEMENT oxygen (#PCDATA)>
<!ELEMENT lightIntensity (#PCDATA)>
<!ELEMENT moleculeDensity (#PCDATA)>
<!ELEMENT owner (#PCDATA)>
<!ELEMENT granted (#PCDATA)>
```

Example:

```
<?xml version='1.0'?>
<!DOCTYPE environment SYSTEM "http://localhost:8888/environment.dtd">
<environment>
  <simulationtime>130</simulationtime>
  <microbialDensity>0.03</microbialDensity>
  <fungaldensity>0.01</fungaldensity>
  <pH>7</pH>
  <temperature>237</temperature>
  <pkw>0.000014</pkw>
  <oxygen>0.0004</oxygen>
```

```

        <lightIntensity>0.0001</lightIntensity>
        <moleculeDensity>0.01</moleculeDensity>
        <owner>xyz</owner>
        <granted>y</granted>
</environment>

<!-- Molecules DTD molecules.dtd -->

<!-- Root element -->
<!ELEMENT molecules (molecule*, owner)>

<!-- Main element -->
<!ELEMENT molecule (c, h, n, o, s, p, doublebond, rings, phenyl, alcohols,
phenols, ethers, esters, ketones, aldehydes, acids, arylacids, amines,
ringn, amides, thioethers, thiols, phosphoesters, hphosphoesters,
phosphates, molculename, granted)>

<!-- Basic elements -->
<!ELEMENT c (#PCDATA)>
<!ELEMENT h (#PCDATA)>
<!ELEMENT n (#PCDATA)>
<!ELEMENT o (#PCDATA)>
<!ELEMENT s (#PCDATA)>
<!ELEMENT p (#PCDATA)>
<!ELEMENT doublebond (#PCDATA)>
<!ELEMENT rings (#PCDATA)>
<!ELEMENT phenyl (#PCDATA)>
<!ELEMENT alcohols (#PCDATA)>
<!ELEMENT phenols (#PCDATA)>
<!ELEMENT ethers (#PCDATA)>
<!ELEMENT esters (#PCDATA)>
<!ELEMENT ketones (#PCDATA)>
<!ELEMENT aldehydes (#PCDATA)>
<!ELEMENT acids (#PCDATA)>
<!ELEMENT arylacids (#PCDATA)>
<!ELEMENT amines (#PCDATA)>
<!ELEMENT ringn (#PCDATA)>
<!ELEMENT amides (#PCDATA)>
<!ELEMENT thioethers (#PCDATA)>
<!ELEMENT thiols (#PCDATA)>
<!ELEMENT phosphoesters (#PCDATA)>
<!ELEMENT hphosphoesters (#PCDATA)>
<!ELEMENT phosphates (#PCDATA)>
<!ELEMENT molculename (#PCDATA)>

```

```
<!ELEMENT granted          (#PCDATA)>
<!ELEMENT owner            (#PCDATA)>
```

Example:

```
<?xml version='1.0'?>
<!DOCTYPE molecules SYSTEM "http://localhost:8888/molecules.dtd">
<molecules>
  <molecule>
    <c>400 </c>
    <h>322 </h>
    <n> </n>
    <o>81 </o>
    <s> </s>
    <p> </p>
    <doublebond>59 </doublebond>
    <rings>40 </rings>
    <phenyl>40 </phenyl>
    <alcohols>1 </alcohols>
    <phenols>1 </phenols>
    <ethers>118 </ethers>
    <esters> </esters>
    <ketones> </ketones>
    <aldehydes> </aldehydes>
    <acids> </acids>
    <arylacids> </arylacids>
    <amines> </amines>
    <ringn> </ringn>
    <amides> </amides>
    <thioethers> </thioethers>
    <thiols> </thiols>
    <phosphoesters> </phosphoesters>
    <hphosphoesters> </hphosphoesters>
    <phosphates> </phosphates>
    <moleculename>lignine </moleculename>
    <owner>xyz </owner>
    <granted>y </granted>
  </molecule>
  <owner>xyz </owner>
</molecules>

<!-- simulationsetup DTD setup.dtd -->

<!-- Root Element -->
```

```

<!ELEMENT setup (environment, moleculetypes, owner)>

<!-- Main Element -->
<!ELEMENT environment (environmentid)>
<!ELEMENT moleculetypes (moleculetype)*>
<!ELEMENT moleculetype (moleculeid, distribution)>

<!-- Basic Element -->
<!ELEMENT environmentid (#PCDATA)>
<!ELEMENT moleculeid (#PCDATA)>
<!ELEMENT distribution (#PCDATA)>
<!ELEMENT owner (#PCDATA)>

```

Example:

```

<?xml version='1.0'?>
<!DOCTYPE molecules SYSTEM "http://localhost:8888/setup.dtd">

<setup>
  <environment>
    <environmentid>102 </environmentid>
  </environment>
  <moleculetypes>
    <moleculetype>
      <moleculeid>1 </moleculeid>
      <distribution>33 </distribution>
    </moleculetype>
    <moleculetype>
      <moleculeid>2 </moleculeid>
      <distribution>67 </distribution>
    </moleculetype>
  </moleculetypes>
  <owner>xyz </owner>
</setup>

```

APPENDIX C

Screen capture and benchmark results

C.1 OptimizeIt

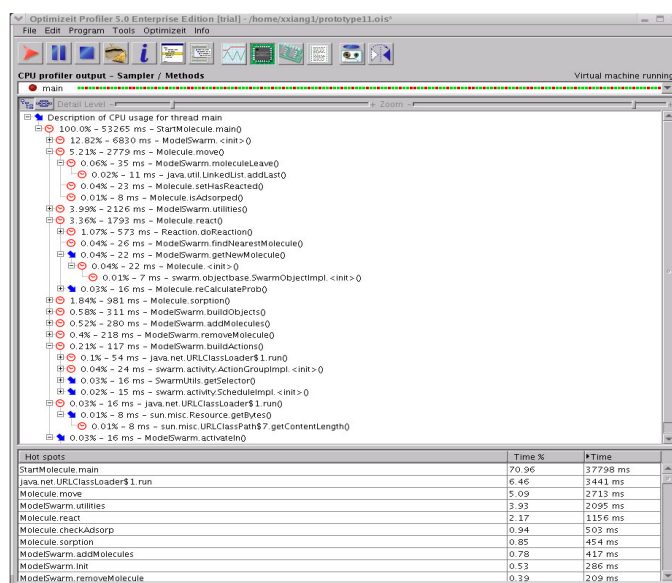


Figure C.1. OptimizeIt provides a tree structure view of the CPU usage while the application is running, and identify the HotSpots.

C.2 Java Grande benchmark

Java Grande benchmark suite¹ is a group of Grande applications written by Java. A grande application is defined as a large-scale application which requires large amount of processing power, network bandwidth, I/O and memory to solve

¹www.epcc.ed.ac.uk/javagrande

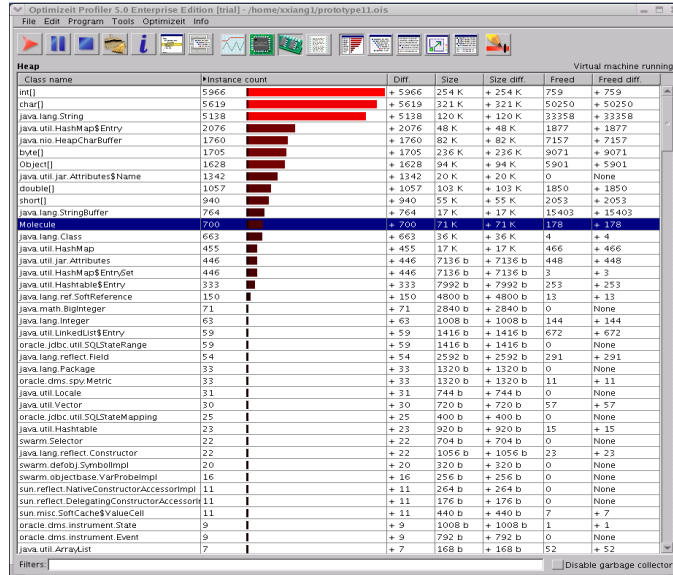


Figure C.2. OptimizIt provides the number of each type of objects that are allocated in the heap and the deallocation information; it also shows the object size.



Figure C.3. OptimizIt provides the graph that shows the heap usage, the activation of garbage collector, the thread activity, and the class load information.

large-scale problem. Java Grande benchmark provides metrics for comparing Java execution environment.

We measure the performance improvement by compiling the benchmark application into native code using GCJ compiler.

We choose the programs, Loop, HeapSort, SOR, Crypt, Search, Euler, MolDyn, from the benchmark suite. These programs are compiled into native code using GCJ compiler from GCC 3.3 version. Elapsed time (wall clock time) is used to compare the execution time of these native codes against these Java programs running on JVM.

Table C.1. Benchmark results for GCJ compiler and Java Hotspot virtual machine.

Benchmarks	GCJ	Java 2 SDK1.4.1_01 JVM (Hotspot)
HeapSort	1.58	2.80
Loop	9.22	32.33
Euler	30.94	26.78
MolDyn	8.75	10.5
Search	12.49	12.72
SOR	5.74	8.63
Crypt	3.85	4.96

C.3 Design for Java threads version and MPJ version

A `Barrier` class has been introduced which provides fast synchronization of threads. Barrier is a particular state that all the threads or processes must wait in until all of them reach this point and only after this state the execution can be continued. Figure C.4 shows the design.

In Figure C.5, a grid has been separated into 3 equal parts. Each node in the cluster holds one subset of the grid. When molecules move across the boundary, molecules have been removed from the current portion of grid and removed from the molecule List. These molecules have been stored in a temporary buffer

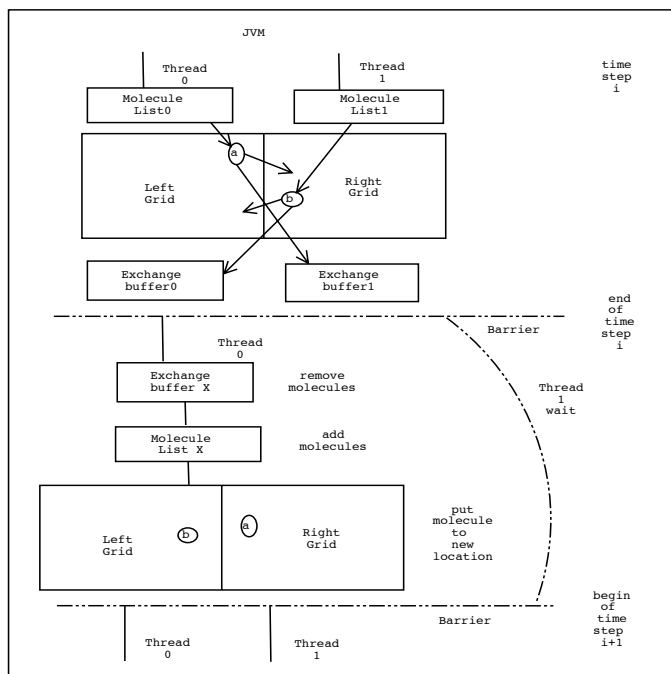


Figure C.4. The design for parallelism of NOM simulation model using Java threads.

according to the boundary they crossed. At the end of each time step, all the processes are synchronized using `MPI.COMM_WORLD.Barrier`. Molecules in the send buffers are sent to their neighbor grids using blocking send and receive mode, `MPI.COMM_WORLD.Sendrecv`, `MPI.COMM_WORLD.Send`, and `MPI.COMM_WORLD.Recv`. The molecule list and grid on each nodes are updated after all the send and receive finished.

C.4 Sequential model results

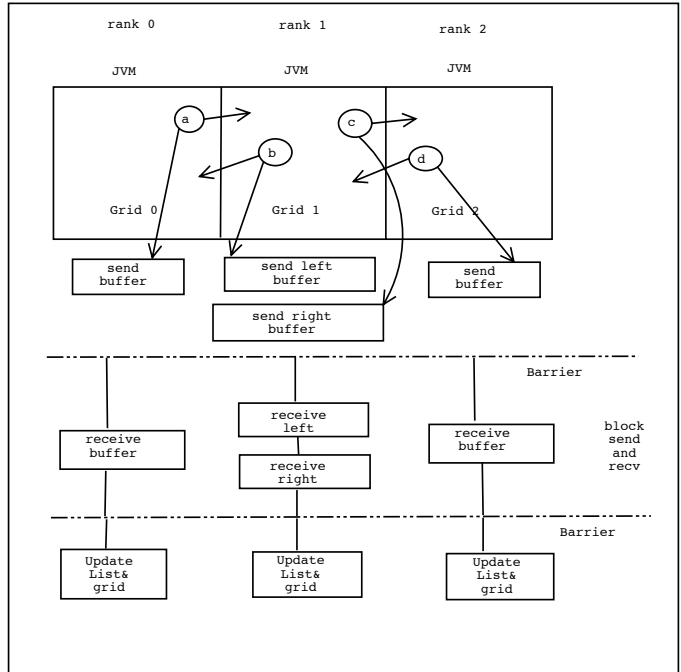


Figure C.5. The design for parallelism of NOM simulation model using MPI.

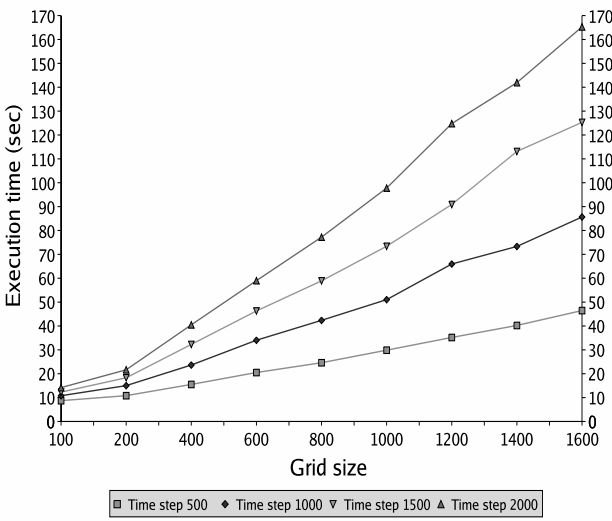


Figure C.6. The behavior of sequential model when time step and grid size increase.

APPENDIX D

Molecular structures and chemical reactions

D.1 Molecular structures

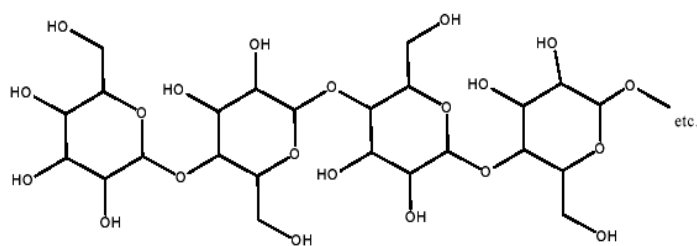


Figure D.1. The structure of cellulose

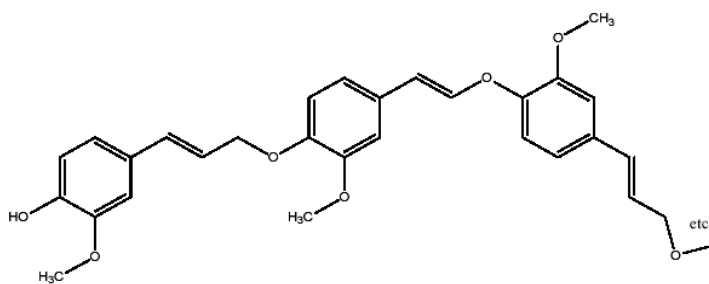


Figure D.2. The structure of lignin

D.2 Chemical reactions

Amino acid	R group
10% Glutamic acid	CH ₂ -CH ₂ -COOH
10% Lysine	CH ₂ -CH ₂ -CH ₂ -CH ₂ -NH ₂
10% Glutamine	CH ₂ -CH ₂ -CONH ₂
10% Serine	CH ₂ -OH
10% Threonine	CHOH-CH ₃
10% Glycine	H
10% Alanine	CH ₃
10% Valine	CH-(CH ₃) ₂
10% Leucine	CH ₂ -CH-(CH ₃) ₂
10% Phenylalanine	CH ₂ -C ₆ H ₅

Figure D.3. The description of protein

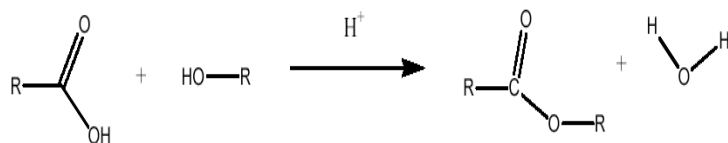


Figure D.4. Ester Condensation

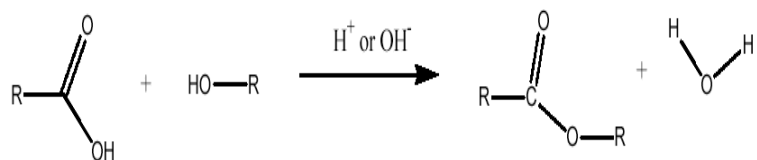


Figure D.5. Ester Hydrolysis

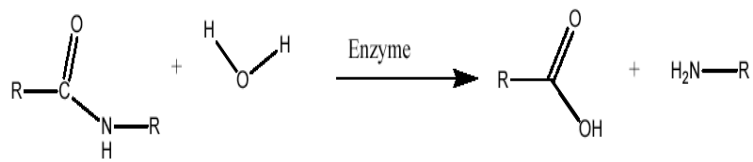


Figure D.6. Amide Hydrolysis

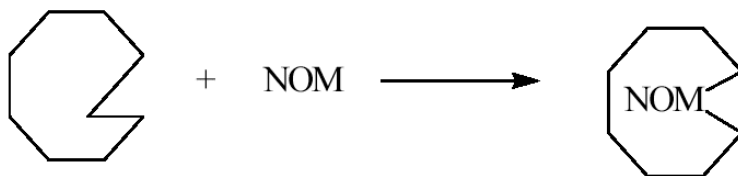


Figure D.7. Microbial uptake

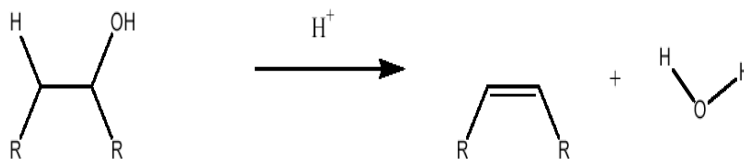


Figure D.8. Dehydration

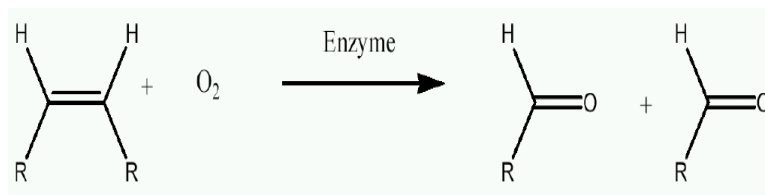


Figure D.9. Strong C=C oxidation

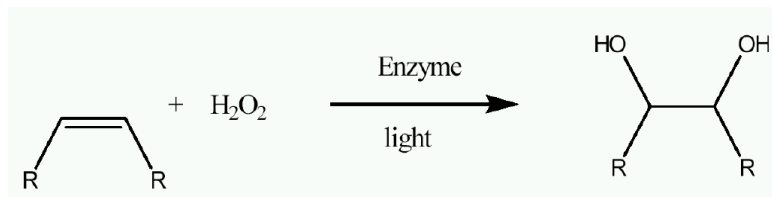


Figure D.10. Mild C=C oxidation

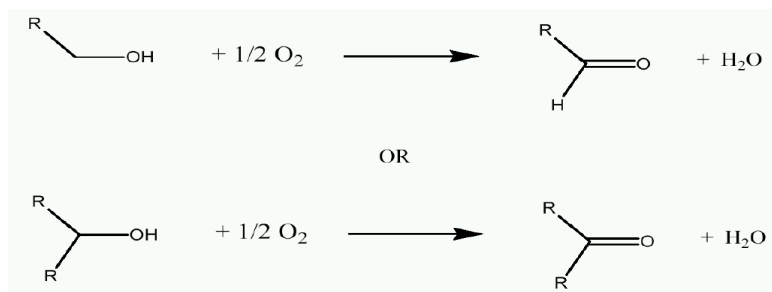


Figure D.11. Alcohol (C-O-H) oxidation

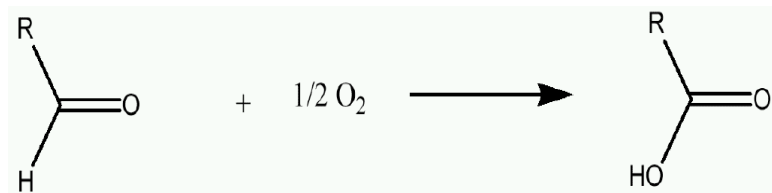


Figure D.12. Aldehyde C=O oxidation

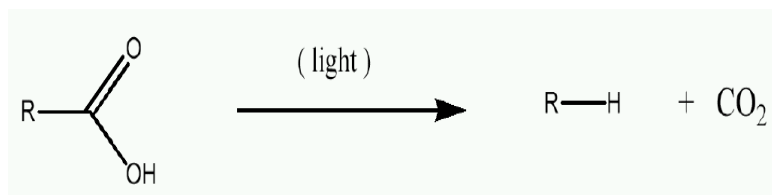


Figure D.13. Decarboxylation

BIBLIOGRAPHY

- [1] Nicholas Kassem and the Enterprise team. *Designing enterprise applications with the Java 2 Platform, Enterprise Edition*. Addison-Wesley, 2000.
- [2] Steven Strogatz. The real scientific hero of 1953. *The New York Times*, March 2003. <http://www.nytimes.com/2003/03/04/opinion/04STRO.html>.
- [3] Geoffrey Fox. E-science meets computational science and information technology. *Computing science & Engineering*, 2002.
- [4] Java 2 platform, enterprise edition. <http://java.sun.com/j2ee/>.
- [5] Microsoft .net. <http://www.microsoft.com/net/>.
- [6] US national virtual observatory. <http://www.us-vo.org/software.html>.
- [7] A scientific web-based application for global tropical cyclone monitoring. <http://www.cio.noaa.gov/hpcc/projects/200128.html>.
- [8] R. M. Jakobovits, J. F. Brinkley, C. Rosse, and E. Weinberger. Enabling clinicians, researchers, and educators to build custom web-based biomedical information systems. In *AMIA Annual Fall Symposium*.
- [9] The study of complex systems. <http://www.pscs.umich.edu/complexity.html>.
- [10] Santa Fe Institute research. <http://www.santafe.edu/sfi/indexResearch.html>.
- [11] J. Gross Louis. Agent-based modeling in ethnobiology: A brief introduction from outside. <http://www.tiem.utk.edu/gross>, April 2002. Talk to NSF Workshop on Priorities in Ethnobiology.
- [12] C. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 1987.
- [13] H. Van Dyke Parunak, Robert Savit, and Rick L. Riolo. Agent-based modeling vs. equation-based modeling: A case study and user's guide. In *Proceedings of Multi-agent systems and Agent-based Simulation (MABS'98)*, 1998.
- [14] Individual-based models. <http://eco.wiz.uni-kassel.de/mod-info/index.html>.
- [15] Steve Cabaniss. Modeling and stochastic simulation of nom reactions, working paper, July 2002.
- [16] Natural organic matter in the nordic countries. <http://www.kjemi.uio.no/envir/nominic/documents/background.html>.

- [17] J.R. Williams. The EPIC model - An overview. In *Natural Resources Modeling Symp.*, 1985.
- [18] S. Hansen, H.E. Jensen, and N.E. Nielsen. Daisy - a soil plant atmosphere system model. *NPO research from the National Agency of Environmental Protection*, (A10), 1990.
- [19] G.K. Brown, S.E. Cabaniss, P. MacCarthy, and J.A. Leenheer. Cu(II) binding by a ph fractionated fulvic acid. *Anal. Chim. Acta* 402, 1999.
- [20] B. Gu, J. Schmitt, Z. Chen, L. Liang, and J.F. McCarthy. Adsorption and desorption of different organic matter fractions on iron oxide: Mechanisms and models. *Environ. Sci. Technol.* 28, pages 38–46, 1995.
- [21] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*. Sun Microsystems, 2000.
- [22] Repast. <http://repast.sourceforge.net/>.
- [23] Swarm development group. <http://www.swarm.org>.
- [24] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations. Technical report, Santa Fe Institute Working Paper 96-06-042, 1996.
- [25] Carl Jason Morton-Firth. *Stochastic simulation of cell signalling pathways*. PhD thesis, University of Cambridge, 1998.
- [26] P. Erdi and J. Toth. *Mathematical models of chemical reactions*. Manchester University Press, Manchester, 1989.
- [27] H. M. Sauro. Scamp: a general-purpose simulator and metabolic control analysis program. *Comput. Appl. BioSci* 9, pages 441–450, 1993.
- [28] G. Zacchi M.Ehlde. Mist: a user-friendly metabolic simulator. *Computer Applications in the Biosciences* 11(2), pages 201–207, 1995.
- [29] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
- [30] Carl Jason Morton-Firth and Deniss Bray. Stochastic simulation of cell signalling pathway. In *Computational Modeling of Genetic and Biochemical Networks*. The MIT Press, Cambridge, MA, 2001.
- [31] Introduction to Monte Carlo methods. <http://csep1.phy.ornl.gov/mc/mc.html>.
- [32] Marty Hall. *Core Servlets and JavaServer Pages*. Sun Microsystems, 2000.
- [33] Jim Gray and Alex Szalay. The world-wide telescope. *Communication of the ACM*, 45(11):51–55, November 2002.
- [34] Yingping Huang. Infrastructure, query optimization, data warehousing and datamining for scientific simulation. Master’s thesis, University of Notre Dame, 2002.

- [35] Mike Ashworth. The potential of Java for high performance applications. In *The First International Conference on the Practical Application of Java*, pages 19–33, 1999.
- [36] J. M. Bull, L. A. Smith, L. Pottage, and R. Freeman. Benchmarking Java against C and FORTRAN for scientific applications. In *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*, pages 97–105, June 2001.
- [37] Steve Wilson and Jeff Kesselman. *Java platform performance strategies and tactics*, chapter Appendix B. Addison-Wesley, 2000.
- [38] Dennis M. Sosnoski. Java performance programming, part 1: Smart object-management saves the day. <http://www.javaworld.com/javaworld/jw-11-1999/jw-11-performance.html>, Nov 1999. Java World.
- [39] Steve Wilson and Jeff Kesselman. *Java platform performance strategies and tactics*, chapter Appendix A. Addison-Wesley, 2000.
- [40] Greg Barish. *Building Scalable and High-Performance Java Web Applications using J2EE Technology*. Addison-Wesley, 2002.
- [41] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification, Second Edition*. Addison-Wesley, 1999.
- [42] Scott Robert Ladd. Benchmarking compilers and languages for ia32. <http://www.coyotegulch.com/reviews/almabench.html>, 01 2003.
- [43] Per Bothner. Compiling Java with GCJ. Linux Journal. <http://www.linuxjournal.com/article.php?sid=4860>, Jan 2003.
- [44] OpenMP Architecture Review Board. OPENMP C and C++ application programming interface. Technical report, OpenMP Architecture Review Board, 1998. Available from <http://www.openmp.org>.
- [45] Mark Bull and Mark Kambites. JOMP—An OpenMP-like interface for Java. In *ACM 2000 Java Grande Conference*. ACM, 2000. Available from <http://www.epcc.ed.ac.uk/research/jomp>.
- [46] Sun Microsystems. Java remote method invocation specification. Technical report, Sun Microsystems, 1998. Available at: <http://java.sun.com/products/jdk/rmi/>.
- [47] V. Getov, G. von Laszewski, M. Philippsen, and I. Foster. Multiparadigm communications in Java for grid computing. *Commication of the ACM*, 44(10):118–125, 2001.
- [48] MPI. <http://www-unix.mcs.anl.gov/mpi/>.
- [49] MPICH: A portable implementation of MPI. <http://www-unix.mcs.anl.gov/mpi/mpich/>.
- [50] LAM/MPI parallel computing. <http://www.lam-mpi.org/>.

- [51] Bryan Carpenter, Vladimir Getov, Glenn Judd, Anthony Skjellum, and Geoffrey Fox. MPJ: MPI-like message passing for Java. *Concurrency: Practice and Experience*, 12(11), 2000.
- [52] Mark Baker, Bryan Carpenter, Geoffrey Fox, Sung Hoon Ko, and Sang Lim. mpiJava: An objected-oriented Java interface to MPI. In *International Workshop on Java for parallel and Distributed Computing, IPPS/SPDP 1999*, April 1999.
- [53] Steven Morin, Israel Korean, and C. Mani Krishna. JMPI: Implementing the message passing standard in Java. In *Internatinal Parallel and Distributed Processing Symposium: IPDPS 2002 Workshops*. IEEE, 2002.
- [54] Kivanc Dincer. Ubiquitous message passing interface implementation in Java: jmp. In *13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*. IEEE, 1999.
- [55] Glenn Judd, Mark Clement, and Quinn Snell. Dogma: Distributed object group management architecture. In *Concurrency: Practice and Experience*, volume 10. ACM 1998 Workshop on Java for High-Performance Network Computing., 1998.
- [56] Stephen H. Koslow and Michael F. Huerta, editors. *Electronic collaboration in science*. Lawrence Erlbaum Associates, 2000.
- [57] Richard T. Kouzes, James D. Myers, and William A. Wulf. Collaboratories: Doing science on the Internet. *IEEE Computer*, 1996.
- [58] Carmen M. Pancarella, Larry A. Rahn, and Christine L. Yang. The diesel combustion collaboratory: combustion researchers collaborating over the Internet. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*.
- [59] Brett McLaughlin. *Java & XML, 2nd Edition*. O'Reilly & Associates, 2001.
- [60] Erik T. Ray. *Learning XML*. O'Reilly & Associates, 2001.