

INFRASTRUCTURE, QUERY OPTIMIZATION, DATA WAREHOUSING AND
DATA MINING FOR SCIENTIFIC SIMULATION

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

Yingping Huang, M.S.

Department of Computer Science and Engineering

Notre Dame, Indiana

September 2002

ACKNOWLEDGEMENTS

I would like to thank Dr. Gregory Madey, for his ideas and direction. His guidance played an important role on the development of this thesis. The NOM research group also includes Xiaorong Xiang and Eric Chanowich. The thesis can not come into being without their contribution. I also would like to thank Dr. Steve Cabaniss and Dr. Patricia Maurice for providing the NOM problem and assistance with its formulation.

I also would like to thank Dr. Brockman and Dr. Scheutz for agreeing to serve as my defense committee members.

Last but not least, I want to thank my parents, my wife and my daughter for their support and encouragement throughout my time at Notre Dame.

INFRASTRUCTURE, QUERY OPTIMIZATION, DATA WAREHOUSING AND
DATA MINING FOR SCIENTIFIC SIMULATION

Abstract

by

Yingping Huang

This thesis examines the application of infrastructure, query optimization, data warehousing and data mining technologies to the area of scientific simulation. One application of scientific simulation is on the behavior of natural organic matter (NOM). NOM is a heterogeneous mixture of organic molecules found in terrestrial and aquatic environment - from forest soils and streams to coastal rivers and marshes to the open sea. NOM plays a vital role in ecological and biogeochemical processes. In this thesis, we present an agent-based stochastic simulation of NOM transformations, including biological and non-biological reactions, as well as adsorption and physical transport. It employs recent advances in web-based development environments such as J2EE, and scalable web-based database management systems such as Oracle to improve the reliability and scalability of the stochastic simulations and to facilitate analysis of the resulting large datasets. A data warehouse is built by extracting, transforming and loading data from the simulation databases. Furthermore, data mining tools and techniques are applied to the data warehouse to discover interesting knowledge. The NOM simulation system is useful in chemistry, geology and environmental science.

To my wife Qiuqiu and my daughter Duoduo.

CONTENTS

ACKNOWLEDGEMENTS	i
TABLES	vi
FIGURES	vii
CHAPTER 1: INTRODUCTION	1
1.1 Introduction to NOM	1
1.2 Introduction to Stochastic Simulation	3
1.2.1 Deterministic Models	3
1.2.2 Stochastic Models	4
1.3 Introduction to Agent-based Technology	4
1.3.1 Intelligent Agents	5
1.3.2 Swarm	6
1.4 The NOM Simulation System	7
1.5 Main contributions of the thesis	7
1.6 Organization of the thesis	8
CHAPTER 2: Background And Technologies	9
2.1 Previous Work	9
2.1.1 Deterministic Models	9
2.1.2 Stochastic Models	10
2.2 The Stochastic Model of NOM	11
2.2.1 The Simulation Algorithm	11
2.3 Technologies	14
2.3.1 Oracle RDBMS and Data Warehouse	14
2.3.2 Oracle Reports	16
2.3.3 Oracle Data Mining	16
2.3.4 JDBC	16
2.3.5 J2EE	17
2.3.6 Java Servlet	17
2.3.7 JSP	18
2.3.8 OC4J/Orion Server	19

CHAPTER 3: The Infrastructure	20
3.1 Multi-tier Architecture	20
3.1.1 How Do Users Access The System	20
3.1.2 How Does The Infrastructure Work	22
3.2 Load Balance and Fail Over	24
3.2.1 Load Balance The Application Servers	24
3.2.2 Load Balance The Database Servers	25
3.2.3 Fail Over	28
CHAPTER 4: CORE SIMULATION ENGINE AND WEB INTERFACE	30
4.1 Objects, Reactions and Processes	30
4.2 Program Structure	34
4.3 Web Interface	38
4.4 Conclusion	51
CHAPTER 5: QUERY OPTIMIZATION	52
5.1 Database Design to Optimize Insertion	52
5.1.1 Information Stored in Databases	53
5.1.2 Parameters While Creating Table NOM	56
5.1.3 Overhead of Constraints and Indexes On Insertion Performance	59
5.2 Query Optimization	59
5.2.1 The Query Statements	61
5.2.2 Temporary Aggregation Tables	65
5.3 Performance Comparison and Conclusion	68
CHAPTER 6: DATA WAREHOUSING FOR THE NOM SIMULATION	71
6.1 Logical Design of The Data Warehouse	71
6.1.1 Detail And Summary Schema	71
6.1.2 Star Schemas	74
6.2 Physical Design of The Data Warehouse	75
6.3 Build The Data Warehouse	78
6.4 Populate The Data Warehouse	81
6.5 Query Optimization	82
6.6 Conclusion	83
CHAPTER 7: DATA MINING FOR THE NOM SIMULATION	85
7.1 Introduction to Oracle Data Mining	85
7.2 Clustering	90
7.2.1 Enhanced k-Means Algorithm	94
7.2.2 O-Cluster Algorithm	95
7.2.3 Description of The Clustering Programs	96
7.2.4 Comparison of Enhanced k-means and O-Cluster	103
7.3 Conclusion	105

CHAPTER 8: CONCLUSIONS	107
8.1 Summary	107
8.2 Conclusions	108
8.3 Future Work	109
BIBLIOGRAPHY	110

TABLES

3.1	The NOM_SESSIONS Table	27
5.1	The ReactionType Table	53
5.2	The structure of the table NOM	55
5.3	The structure of the table NOM (continued)	56
5.4	The REACTIONS_BY_TYPE Table	65
5.5	The REACTIONS_BY_TIME Table	66
5.6	Insertion and query performance comparison	70
7.1	Comparison of Enhanced k-means and O-Cluster)	105

FIGURES

2.1	Flow-chart representation of one molecule in a time step	13
3.1	The infrastructure	21
4.1	Simulation running in batch mode	31
4.2	Simulation running in GUI mode	32
4.3	The class diagram	34
4.4	The use case diagram	37
4.5	UML class diagram for the users information database	38
4.6	The home page of the simulation system	40
4.7	The sign up page for new users	41
4.8	The introduction step of the simulation configuration wizard	42
4.9	The environment step of the simulation configuration wizard	43
4.10	Step 3a of the Molecule Editor	45
4.11	Step 3b of the Molecule Editor	46
4.12	Step 4 of the simulation configuration wizard	47
4.13	Simulation invoked successfully	48
4.14	Reports summary of the user	49
4.15	Chemical reactions statistics	50
5.1	Constraints and indexes are overhead for insertion	60
5.2	Reports for the two examples	62
5.3	Query statement for report 1 and result	63
5.4	Query statement for report 2	64

5.5	Insertion comparison of the three scenarios	69
6.1	Examples of “Detail and Summary” Schema	73
6.2	Star Schema	76
6.3	Star Join	84
7.1	Data mining as step of knowledge discovery	86
7.2	Model-Build Process	89
7.3	Model-Apply Process	91
7.4	Model-Apply Results Table	104

CHAPTER 1

INTRODUCTION

1.1 Introduction to NOM

Natural Organic Matter (NOM) is a complex mixture of compounds formed as a result of the breakdown of animal and plant material in the environment. The composition of the mixture is strongly dependent on the environmental source. Generalizations regarding chemical character can be prone to misinterpretation as the character of the compounds present in the mixture is extremely diverse. However, if the complexity of the NOM is kept in mind, a broad understanding of its character is possible. NOM consists mainly of carbon, oxygen and hydrogen. Nitrogen, phosphorus and sulphur can also be present; their prevalence will depend on the source of the NOM. A range of compounds, from small hydrophilic acids, proteins and amino acids to large humic and fulvic acids, are constituents of most NOM. The organic compounds can range from largely aliphatic to highly aromatic, from molecular weights around 10,000 down to 2000, and from highly charged to uncharged compounds.

NOM is ubiquitous in terrestrial, aquatic, and marine ecosystems, playing a crucial role in such important processes as the evolution and fertility of soils; the mobility and transport of pollutants such as trace metals, radionuclides and hydrophobic organic compounds; the availability of nutrients to microorganisms and plant communities; the growth and dissolution of minerals; and the global biogeo-

chemical cycling of the elements [6].

NOM is very important to drinking water systems. It has a significant impact on all aspects of drinking water treatment. NOM is responsible for the majority of the coagulant demand. Therefore waters with high dissolved organic carbon levels usually have a high coagulant requirement and consequent high treatment costs. Where activated carbon is used for the removal of micro-contaminants, e.g., tastes and odors, pesticides, NOM competes strongly for adsorption sites, and the amount of activated carbon required is increased or the lifetime of the carbon is drastically reduced. NOM exerts a high demand for chemicals, such as chlorine, chloramine and ozone, thus increasing the costs associated with their application. In addition, the reaction between oxidants and NOM can produce by-products, some of which are known to be harmful to health at high concentrations. NOM is responsible for fouling of membranes, reducing the flux, resulting in higher frequency of back-washing and cleaning membrane systems. NOM serves as a substrate for bacterial growth. This may lead to re-growth in the distribution system where a sufficient disinfectant residual cannot be maintained.

The importance of NOM attracts numerous researchers, including chemists, geologists or even computer scientists. Despite decades of research, we still know relatively little about the structure, chemical composition, and chemical properties such as molecular weight, functional group concentrations, structure, composition, and reactivity [32] [9] [23].

During the last 50 years, some researches have been done on the functional behavior of NOMs, including pH buffering, metal complexation, pollutant solubilization, adsorption onto mineral surfaces, alteration of mineral precipitation and dissolution, bioavailability to heterotrophs and promotion of photochemical reactions [35] [21] [27]. Different models are proposed to handle NOM research, including ODE

(Ordinal Differential Equation), PDE (Partial Differential Equation), etc. But the diversity of the compounds present in NOM leads to the difficulty of describing the mixture adequately and computationally expensive.

1.2 Introduction to Stochastic Simulation

Models of chemical reactions can be classified in three ways: the simulation time can be continuous or discrete, the state-space can be continuous or discrete, the evolution of the system can be deterministic or stochastic [13].

A deterministic model is one in which the state of the system at one moment in time completely specifies the system for all times. In a stochastic model, the state of the system is represented by a set of values with a certain probability distribution, such that the evolution over time is dependent on a series of probabilistic events. Deterministic models are easier to mathematical analysis than stochastic ones, due to the difficulty in solving stochastic differential equations.

As the time is continuous, it is natural to represent temporal relationships of a simulation model. However, because discrete time models are simpler than continuous models and computers can perform calculations only at discrete points, it is often preferable to model a system using discrete time steps, and the system state is updated at the end of each time step. State-space refers to the n -dimensional space containing all the possible states of a given system with n variables.

1.2.1 Deterministic Models

The mathematical basis for chemical reactions stems from work done by Michaelis and Menten in 1913. They proposed a situation in which a chemical reaction is represented by a system of ordinary differential equations (ODEs). For example, a chemical reaction: $A + B \longrightarrow C$ would be described using $\frac{dC}{dt} = k[A][B]$ where k is the reaction rate constant ($M^{-1}s^{-1}$). Here $[A]$ denotes the concentration of A in

moles per liter, so does $[B]$.

The problem of this model is that the system of ODEs assume that the system is a continuous predictable process. However, it turns out that in reality, some situations are not continuous and predictable and the random fluctuations or external cases drive many of the reactions.

1.2.2 Stochastic Models

Stochastic methods for modeling chemical reactions dates back to 1940 by H.A. Kramers, but a real breakthrough came in 1976 when D.T. Gillespie published a paper describing a new method for solving a system of ODEs [18]. In his paper, Gillespie considers the time evolution of the system as a kind of random walk that is governed by the “master equation”. The “master equation” is a differential difference equation which keeps track of all the molecules in the system. Even though it may be impossible to solve a complicated chemical system exactly, Gillespie’s method can be used to numerically simulate the time evolution of the system. With his method, reactions are thought of as occurring with certain probabilities, and the events which occur change the probabilities of subsequent events. The algorithm is based on the quantity $P(t, u)$ which is the probability that the reaction u occurs in the time interval t . In reality, a chemical reaction is a stochastic event involving a discrete number of molecules; therefore, it is more realistically described using a stochastic model than a deterministic one.

1.3 Introduction to Agent-based Technology

Agent-based systems are of increasing importance. They are regarded as a new paradigm enabling an important step forward in empirical sciences, technology and theory. Agent-based technology has an enormous influence on model building and simulation. Novel design patterns and models are opening up new possibilities.

1.3.1 Intelligent Agents

Intelligent agents can be equipped with different attributes, although many may not be necessary or useful in every application. The following are some important features of intelligent agents:

- Autonomous Behavior

Every agent has autonomous behavior. This means that an intelligent agent behaves autonomously without external control. In addition there is the induced dynamism which describes how the intelligent agent reacts in response to inputs from the environment. Strategies that guide the behavior of an agent when it is pursuing its goals are of particular importance.

- Individual World View

Every agent has its own model of the external world that surrounds it. This model describes how the intelligent agent sees the world. The manner in which the intelligent agent builds up its model of the world on the basis of the information it receives from its environment is of particular interest.

- Communicative and Cooperative Capacity

Intelligent agents can exchange information with their environment and with other intelligent agents. By means of the possibility of communication an intelligent agent must obtain information about its environment which enables it to build up its own world model. Moreover, the possibility of communication with other intelligent agents is the precondition of common action in pursuit of a goal.

- Intelligent Behavior

As the term Intelligent Agent indicates, behavioral possibilities such as the ca-

pability to learn, logical deduction or construction of an environmental model are required.

- Spatial Mobility

Intelligent agents are sometimes required to display spatial mobility. Spatial mobility is of particular interest to us in our simulation of molecular behavior.

Not all agents are intelligent. In the NOM simulation system, the molecules are not intelligent. The modeling and simulation of real systems consisting of agents that cooperate with each other has recently emerged as an important field of research. Two areas are of particular interest:

- Strategies and Decentral Control

The object is to develop individual strategies that individual agents pursue and that ensure that a common goal can be achieved even without central regulation. Examples of this area are social insects and birds, in which there is no lead or centralized control in a group.

- Emergent Behavior

Cooperation between agents can produce a stable system that displays new global behavior on a higher level of abstraction. The task is to explain this global behavior on the basis of the individual attributes of the intelligent agents and of their interactions.

1.3.2 Swarm

Swarm is a software package for multi-agent simulation of complex systems, originally developed at the Santa Fe Institute [4]. Swarm is intended to be a useful tool for researchers in a variety of disciplines. The basic architecture of Swarm is the simulation of collections of concurrently interacting agents. With this architecture,

we can implement a large variety of agent based models. The swarm software is open source and is available to the general public under GNU licensing terms. Swarm is experimental software, which means that it is complete enough to be useful but will always be under development.

1.4 The NOM Simulation System

We develop an agent-based stochastic model for time-dependent evolution of NOM in the environment. The objective of this model is to produce both a new methodology and a specific program for predicting the properties of NOM over time as it evolves from precursor molecules to eventual mineralization. The methodology developed is an agent-based stochastic simulation of NOM transforms, including biological and non-biological reactions, as well as adsorption and physical transport. It employs recent advances in web-based interfaces, and scalable web-based database management systems to improve the reliability of the stochastic simulation and to facilitate analysis of the resulting large datasets. Furthermore, data-mining techniques and models are developed to discover knowledge from the simulation data warehouse.

1.5 Main contributions of the thesis

In this thesis, we present an agent-based stochastic model which has enormous potential benefits in a wide range of disciplines, since literally all aspects of biogeochemistry are related to NOM. It acts as a starting point of a powerful tool which can be applied to aquatic ecosystem studies, soil and crop science, environmental protection, remediation in the surface and sub-surface, and global climate change prediction. Unlike current models, this model explicitly treat NOM as a heterogeneous mixture, so that distributions of physical, chemical and biological properties

can be predicted. To summarize, this thesis:

- presented an agent-based stochastic simulation model;
- employed advanced IT technologies to scientific simulation;
- applied data mining techniques to simulation data.

1.6 Organization of the thesis

The rest of this thesis consists of seven chapters. Chapter 2 discuss the background of the problem we are studying, and the technologies which are used in the simulation system. Chapter 3 presents the infrastructure which supports our NOM simulation system. Chapter 4 gives an overview of our web-based interface and the simulation program. Chapter 5 discusses the status reports pages of the simulation. Chapter 6 describes our approaches to designing and building the data warehouse. Chapter 7 presents the work of applying data mining techniques to the simulation data warehouse. Chapter 8 gives some ideas for further work.

CHAPTER 2

Background And Technologies

In this chapter, we list some previous work done by other researchers which is closely related to our research on the NOM simulation system. Then we briefly present the stochastic model with underlying algorithms. Finally, we discuss some web-based interface and RDBMS technologies used in the next few chapters.

2.1 Previous Work

Over the recent years, many simulation models have been developed to simulate bio-chemical reactions. These models can be categorized as deterministic or stochastic.

2.1.1 Deterministic Models

Several simulators are available which can be used to predict the behavior of a network of chemical reactions. Among them, SCAMP [39] and METASIM [34], employ non-standard biochemical languages to specify a model in a series of command files. The simulator then uses these files to construct differential equations which then are solved using numerical integration. Other deterministic models are also proposed with more specific programs to solve a single set of differential equations using similar methods [8] [26].

Under some situations, deterministic models no longer represent the physical system adequately and cannot be used to predict the concentrations of chemical

species.

2.1.2 Stochastic Models

In 1976, Gillespie [18] developed a discrete, stochastic algorithm to simulate chemical reactions. In his model, time is quantized into small time steps with variable lengths. At each time step, based on the rate constants and population size of each chemical species, a random number is generated to choose which reaction will occur, and another random number to determine how long the step will last. The chemical populations are altered according to the stoichiometry of the reaction and the process is repeated. To determine which chemical reaction will occur in a given time step, the Gillespie algorithm calculates the probability of each reaction relative to another by multiplying the rate constant of each reaction with the concentration of its substrates. A random number is then used to determine which reaction will occur according to this set of weighted probabilities.

In 1995, IBM Almaden Research Center developed a Chemical Kinetics Simulator (CKS) [10]. This simulator has a stochastic simulation package which allows you to calculate concentrations of reactants and products in a chemical system as a function of time.

In 1997, Punch et al reported a software system called BESS which simulated the action of biodegradation pathways on compounds. It did so by encoding biodegradation pathways in a knowledge base and applying those pathways in sequence to the compound, breaking it down into metabolites [43]. BESS used the expert system approach. As its heart, BESS was a rule-based system. It was implemented using VisualWorks Smalltalk. It was then re-implemented using C++ and Java.

Firth and Bray [15] [14] later developed a simulation algorithm called STOCHSIM using stochastic model to predict cell signalling pathways. In their model, time is

quantitized into discrete slices. At each time step, one or two molecules are selected from the set of molecules depending on some pre-calculated probabilities. Then first order or second order reactions could occur based on some generated random numbers. Note that at each time step, only one chemical reaction is allowed.

Swarm has been used in chemical simulations. In [3], McMullin implemented an artificial chemistry using Swarm.

2.2 The Stochastic Model of NOM

In this thesis, we develop a new stochastic simulation model using agent-based technology. This stochastic model of NOM represents individual molecules as discrete objects of specified elemental and functional group composition, size and reactivity. Temporal evolution of NOM from biological precursor compounds such as lignin, polysaccharides and proteins is simulated in which specific probabilities are assigned to particular transformations. The reactivity of the resulting NOM assemblage over time can be predicted based on the distributions of molecular properties.

This stochastic approach has several advantages: it is much less computationally intensive than molecular modeling or explicit kinetic simulation of hundreds of compounds, it can be adapted to a variety of time scales and processes, and it intrinsically handles NOM structural and functional heterogeneity. This approach also employ state-of-the-art information technology including web interface and scalable relational database management systems (RDBMS).

2.2.1 The Simulation Algorithm

The NOM simulation is implemented in a discrete 2D space with discrete time. The simulated space is a rectangular lattice. Each molecule can occupy at most one cell, and each cell can host at most one molecule. During execution of the simulation, each molecule may move to another lattice or stay in a fix position

according to predefined simple rules in physical processes. In chemical reactions, one molecule could split and occupy two cells in the world; two or more molecules could combine and occupy just one cell. In the situation of adsorption, two molecules can be in the same cell if one of them is adsorbed.

There are 10 types of chemical reactions which could occur in the simulation system, namely, ester condensation, ester hydrolysis, amine hydrolysis, microbe uptake, dehydration, strong C=C oxidation, mild C=C oxidation, Alcohol (C-O-H) oxidation, aldehyde C=O oxidation, decarboxylation. Each molecule has a probability for each type of chemical reaction. The calculation of the probability is based on the structure of the molecule, and the environment in which the molecule resides. These environmental variables include the length of the time step, microbe density, fungal density, pH value, temperature, pKw (the equilibrium constant for the autolysis of water, which is very close to 14.0), oxygen density, light density, etc. After each chemical reaction, the probabilities of these reaction types of a molecule is re-calculated. The reaction probabilities are stored in an array associated with each molecule.

As illustrated in Figure 2.1, in each time step, for each molecule, a random number is generated which is used to determine whether a chemical reaction will occur, and if one occurs, which reaction type. In the simulation, the sum of all the reaction probabilities is controlled to be less than 1 percent. The interval $[0, 1]$, is partitioned into 11 subintervals. 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, etc. The length of the last interval is the probability in which no reaction will occur. The generated random number from the interval $[0,1)$ will reside in one of these intervals, and it will decide which chemical reaction will occur if any at all.

If the chosen chemical reaction type is a second order reaction, i.e., there will be two molecules involving in the reaction, the second molecule will be chosen from one of its nearest neighbors who are not involved in a chemical reaction yet.

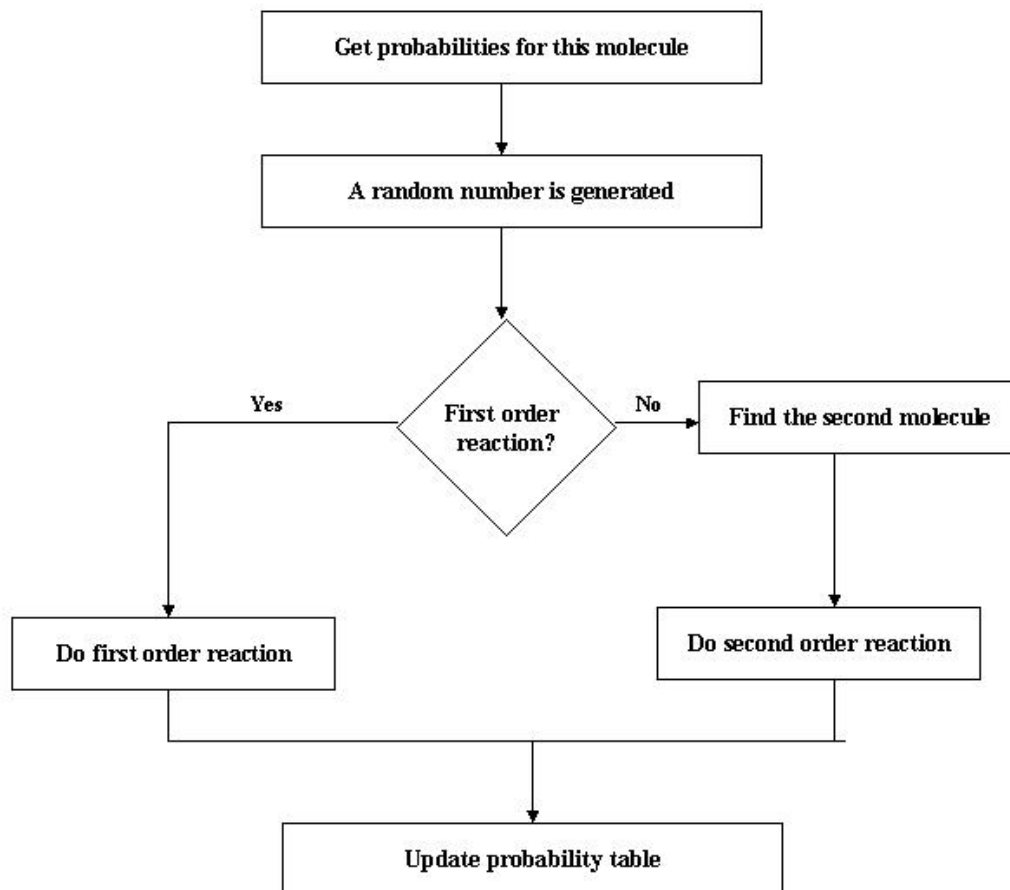


Figure 2.1. Flow-chart representation of one molecule in a time step

After the reaction takes place, the probability tables for involved molecules are updated at the end of the time step and the new probability tables are assigned to newly produced molecules. Note that, at each time stamp, one or more chemical reactions could occur.

2.3 Technologies

The NOM simulation system includes a web interface which allows users to provide input for the core simulation and view immediate status reports of their simulation through their browsers. The web interface is written using JavaServer pages (JSP), the core simulation is invoked by a Java servlet. To run JSP and Java servlets, we use the Java 2 Enterprise Edition (J2EE) platform. Among many J2EE platforms such as Apache Tomcat, Sun's JavaServer Web Development Kit (JSWDK), Java Web Server, Orion Server, and Oracle Application Server, we choose the Oracle9iAS container for J2EE (OC4J, also known as Orion Server) which is part of the Oracle Application Server Suite. The core simulation is written in Java with the Swarm library. The simulation data is stored in Oracle RDBMS using Java Database Connection (JDBC). Reports of the simulation results are generated upon user request, using SQL and PL/SQL language with the help of Oracle Reports. A data warehouse is built by extracting, transforming and loading data from the simulation databases. Then data mining programs are developed to make classifications, predictions, clustering, etc. to the data warehouse using the Oracle Data Mining APIs. In the following section, we briefly introduce these technologies.

2.3.1 Oracle RDBMS and Data Warehouse

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance.

We use both Oracle8i and Oracle9i in the simulation system. For detailed introduction of Oracle database, SQL and Oracle PL/SQL languages, see Loney and Koch [28].

A common way of introducing data warehousing is to refer the characteristics of a data warehouse as set forth by Inmon [22]: subject oriented, integrated, non-volatile, time variant.

- Subject Oriented: Data warehouses are designed to help users analyze data. For example, to learn more about a company's sales data, we can build a warehouse that concentrates on sales. Using this warehouse, we can answer questions like "Who was our best customer for this item last year?" This ability to define a data warehouse by subject matter, sales in this case, makes the data warehouse subject oriented.
- Integrated: Integration is closely related to subject orientation. Data warehouses must put data from disparate sources into a consistent format. They must resolve such problems as naming conflicts and inconsistencies among units of measure. When they achieve this, they are said to be integrated.
- Non-volatile: Non-volatile means that, once entered into the warehouse, data should not change. This is logical because the purpose of a warehouse is to enable you to analyze what has occurred.
- Time Variant: In order to discover trends in business, analysts need large amounts of data, collected over time (i.e., with temporal attributes). A data warehouse's focus on change over time is what is meant by the term time variant.

Dodge and Gorman elaborate in more detail on Oracle data warehousing [12].

2.3.2 Oracle Reports

Oracle9i Reports is Oracle's reporting tool. It consists of Oracle9i Reports Developer and Oracle9iAS Reports Services. Oracle9i Reports Developer uses a declarative, document-centric development model to help the J2EE developer rapidly create web and paper reports against the Oracle databases. While allowing developers to leverage recent J2EE technologies such as JSP and XML, Oracle9iAS Reports Services publishes these reports in any format including HTML, XML, JSP to any destination including web browser and Oracle9iAS Portal. In this thesis, we use Oracle Reports to publish reports in JSP format on the web so that users can view the reports through their browsers. Additional information on Oracle Reports is available in Muller [33].

2.3.3 Oracle Data Mining

Oracle9i Data Mining, an option to Oracle9i Enterprise Edition, allows companies to build applications that mine corporate databases to discover new insights, and integrate those insights into business applications. Oracle9i Data Mining embeds data mining functionality into the Oracle9i database, for making classifications, predictions, and associations. All model-building, scoring and metadata management operations are initiated via a Java-based API and occur entirely within the relational database. In this thesis, we apply Oracle Data Mining to the simulation data warehouse. The Oracle Data Mining manual [5] contains an introduction of the data mining algorithms, APIs and simple examples.

2.3.4 JDBC

JDBC (Java Database Connectivity) is a standard Java interface for connecting from Java to relational databases such as Oracle. The JDBC standard was defined by Sun Microsystems, allowing individual providers to implement and extend the

standard with their own JDBC driver. JDBC is based on the X/Open SQL Call Level Interface and complies with the SQL92 Entry Level standard. In addition to supporting the standard JDBC API, Oracle drivers have extensions to support Oracle-specific datatypes and to enhance performance. In this thesis, JDBC is used by the core simulation to write simulation data to the Oracle databases. To learn more about Oracle JDBC, refer to Price and Wald [36].

2.3.5 J2EE

The J2EE architecture is based on the Java programming language. An advantage of Java is that it enables organizations to write their code once and deploy that code onto any platform. The process is as follows: developers write source code in Java, the Java code is compiled into bytecode, which is a cross-platform intermediary, halfway between source code and machine language. When the code is ready to run, the Java Runtime Environment (JRE) interprets this bytecode and executes it at run-time. J2EE is an application of Java. J2EE components are transformed into bytecode and executed by a JRE at runtime.

J2EE has historically been an architecture for building server-side deployments in the Java programming language. It can be used to build traditional web sites, software components, or packaged applications. J2EE has been extended to include support for building XML-based web services as well. These web services can interoperate with other web services that may or may not have been written to the J2EE standard. For an introduction of J2EE, see Deitel et al [1].

2.3.6 Java Servlet

A Java servlet is a program that extends the functionality of a Web server. A servlet receives a request from a client, dynamically generates the response (possibly querying databases to fulfill the request), and sends the response containing

an HTML or XML document to the client. Servlets are similar to CGI and its alternatives such as PHP and ASP, but much easier to write, because servlets use Java classes and streams. Servlets execute faster, because servlets are compiled to Java Byte code. At run time, the servlet instance is kept in memory, and each client request spawns a new thread.

Servlets make it easy to generate data to an HTTP response stream in a dynamic fashion. The issue facing servlets is that HTTP is a stateless protocol. That is, each request is performed as a new connection, so flow control does not come naturally between requests. Session tracking or session management maintains the state of specific clients between requests.

2.3.7 JSP

JavaServer Pages (JSP) are a text-based, presentation-centric way to develop servlets. JSPs allow Web developers and designers to rapidly develop and easily maintain information-rich, dynamic Web pages that leverage existing business systems. JSPs enable a clean separation and assembly of presentation and content generation, enabling Web designers to change the overall page layout without altering the underlying dynamic content. JSPs use XML-like tags and scriptlets, written in the Java programming language, to encapsulate the logic that generates the content for the page. Additionally, the application logic can reside in server-based resources, such as JavaBeans, that the page accesses with these tags and scriptlets. All formatting (HTML or XML) tags are passed directly back to the response page. By separating the page logic from its design and display, and supporting a reusable component-based design, JSP technology is faster and easier when building Web-based applications. A JSP page looks like a standard HTML or XML page with additional elements that the JSP engine processes and strips out. Typically, the

JSP generates dynamic content, such as XML, HTML, and WML. Refer to Hall [19], which elaborates the details of Java Servlet and JSP technologies.

2.3.8 OC4J/Orion Server

Oracle provides a complete set of the J2EE container written entirely in Java that executes on the Java virtual machine (JVM) of the standard Java Development Kit (JDK). You can run the Oracle9iAS containers for J2EE (OC4J) on the standard JDK that exists on most operating systems. OC4J is J2EE compliant and provides all the containers that J2EE specifies. OC4J is based on technology licensed from Ironflare Corporation, which develops the Orion Server - one of the leading J2EE containers. OC4J supports the standard J2EE APIs including Java Servlets, JavaServer Pages (JSP), Enterprise JavaBeans (EJB), Java Database Connectivity Services (JDBC), Java Naming and Directory Interface (JNDI), Java Transaction API (JTA), Java Message Service (JMS), etc.

CHAPTER 3

The Infrastructure

The simulation model employs recent advances in web-based interfaces, and scalable web-based database management systems to improve the reliability of the stochastic simulations and to facilitate analysis of the resulting large datasets. In the previous chapter, we briefly introduced the technologies we used in the simulation. In this chapter, we will build the infrastructure of the simulation system using those technologies. Here infrastructure means the group of database servers, web servers, J2EE platform, reports servers and data mining servers, as well as the underlying software which supports our simulation. In the following sections, sometimes we may use architecture instead of infrastructure, but they are interchangeable.

3.1 Multi-tier Architecture

The multi-tier infrastructure of the simulation system is shown in Figure 3.1. Before explaining the infrastructure in detail, let us show how users access the simulation system.

3.1.1 How Do Users Access The System

The simulation system is web-based, so users will access the system through their web browser, including Netscape and Internet Explorer, using the well-known HyperText Transfer Protocol (HTTP). Let us list the main steps a user will encounter

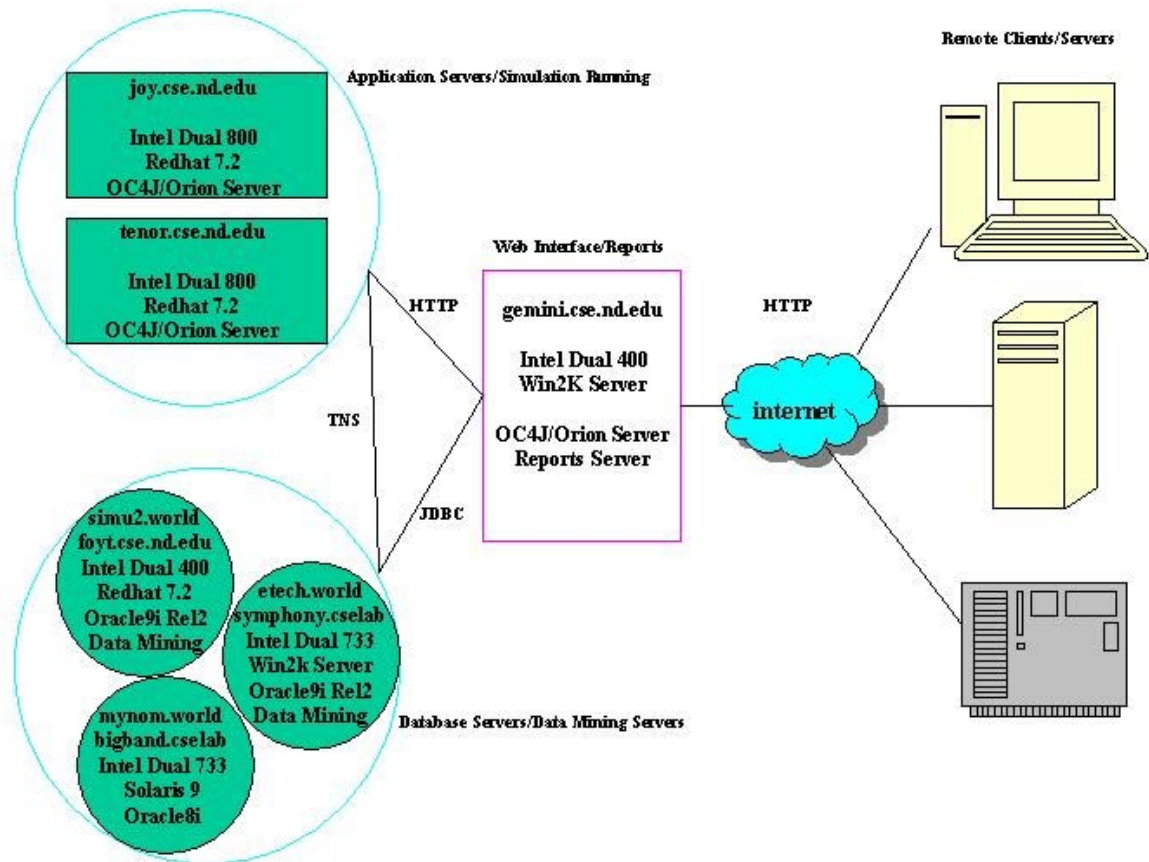


Figure 3.1. The infrastructure

when using the system.

1. The user points his/her browser to the home page:
`http://joy.cse.nd.edu:8000/nom/homepage.jsp`.
2. If the user is a new user, he/she registers with the simulation system. Existing users will skip this step.
3. He/she logs in the system, provides environment variables and molecule information through the Molecule Editor Wizard.
4. He/she invokes the simulation after finishing the wizard.
5. His/her browser will be pointed to the reports page.
6. Data mining requests will be sent to the data mining server if the user requests.

3.1.2 How Does The Infrastructure Work

The core simulation program is written in Java, using the Swarm library. We discuss the program in more detail in chapter 4. Data generated in the simulation is stored in databases. The reports pages retrieve simulation results from these databases. The large datasets are summarized, extracted, transformed and loaded to a data warehouse which is another database enabling us to apply data mining techniques.

Application Server

The Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multi-tier enterprise applications [1]. J2EE supports Enterprise Java Beans (EJB), Java Servlets, Java Server Pages (JSP) and XML technology. In this section, application server means a J2EE platform which enables us to publish JSP pages to

the web and to invoke simulation using Java Servlet. Among all application servers, we choose the OC4J (Oracle9iAS container for J2EE) or Orion Server. Orion Server is embedded in OC4J Server. Either Orion Server or OC4J Server can help us build our web interface using JSP and Java Servlet. You can download OC4J Server from <http://www.oracle.com> and Orion Server from <http://www.orionserver.com>.

Currently, we have two Linux machines, `joy.cse.nd.edu` and `tenor.cse.nd.edu`, serving as the application servers. The simulations are running either on `joy` or on `tenor`, invoked by Java Servlets on `joy` and `tenor` respectively. A large dataset is generated by the simulation and stored in Oracle databases.

Our simulation uses the Swarm library and this is installed on both `joy` and `tenor`. Swarm can be freely downloaded from <http://www.swarm.org>.

Oracle Database Server and Data Mining Server

We use Oracle to store the dataset generated by the simulation. When the simulation is running, data is inserted into the Oracle database using Java Database Connection (JDBC).

To store the simulation data while the simulations are running, we need to create Online Transactional Processing (OLTP) databases. Since our simulation will generate a huge set of data, the database was designed for fast insertion and efficient storage. We also created a data warehouse which stores summarized, transformed and aggregated data from those OLTP databases. The main goal of the data warehouse is for data mining, so fast query is our main concern. We will discuss in detail how we created the OLTP databases and data warehouse in chapter 5 and 6.

Currently, we have three database servers: `foyt.cse.nd.edu`, `symphony.cselab.nd.edu` and `bigband.cselab.nd.edu`, two of which, `foyt` and `symphony`, will also act as data mining servers. The data mining server includes a library of Java APIs which help

users design and develop data mining programs. We discuss more on data mining servers in chapter 7.

Oracle Reports Server

To enable users to view simulation results such as statistics after they invoke their simulations, we created reports pages to let them visualize results on their browser. These reports pages are generated by the Oracle Reports tool in JSP format. The reports pages query data from the Oracle databases using the Transparent Network Substrate (TNS) protocol. Since these pages are JSP pages, a J2EE platform is necessary. The reports server is embedded in an OC4J server as a Java Servlet. In chapter 5, we present in detail how we created the reports pages.

3.2 Load Balance and Fail Over

As shown in Figure 3.1, we have 2 application servers on which simulation is invoked and running. We also have 3 database servers which can store simulation data. To effectively use these resources, we add code in our web interface to distribute the simulations evenly over the two application servers. We also add code in the core simulation program such that simulation data can be stored evenly in the three database servers.

3.2.1 Load Balance The Application Servers

The idea of load balancing application servers is straight forward: we don't want our users to run the simulation only on joy or tenor. So we use the round robin method to load the simulation among the two application servers as in the following JSP code:

```
<%! private static int access=0; %>
<%
if ((++access)%2==0){
```

```

%>
<a href='http://joy.cse.nd.edu:8000/nom/runnom?user_id=10''>
  Invoke</a>
<%
}
else{
%>
<a href='http://tenor.cse.nd.edu:8000/nom/runnom?user_id=10''>
  Invoke</a>
<%
}
%>

```

In the above JSP code, we declare a **static** variable “access”. The reason we declare it as static is that every load of the page will increment the variable access by 1. If access is even, the simulation will run on joy, otherwise it will run on tenor. In this way, if multiple simulations are invoked, half of them will run on joy and the other half on tenor.

3.2.2 Load Balance The Database Servers

The idea of load balancing database servers are the same as balance the application servers. The following piece of Java code in our simulation program shows us how database servers are load balanced, where sessionid is the identifier for the current simulation invoked by a user.

```

...
int sid=(sessionid)%3+1;
String url=null;
if(sid==1)
  url='jdbc:oracle:thin:@foyt.cse.nd.edu:1521:simu2'';
else if (sid==2)
  url='jdbc:oracle:thin:@symphony.cselab.nd.edu:1521:etech'';
else
  url='jdbc:oracle:thin:@bigband.cselab.nd.edu:1521:mynom'';
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
Connection conn=DriverManager.
    getConnection(url, ‘username’, ‘password’);
...

```

But a problem arises here because the reports pages will be pointed after invoking the simulation and the reports pages need to know which database should be accessed to get the simulation results. To handle this, we create a table `nom_sessions` with attributes `session_id`, `user_id`, `sid` and `status`, where `session_id` is the identifier for the session of simulation invoked by a user, `user_id` is the identifier of the user, `sid` is the identifier for the database which stores the simulation data for this session, `status` denotes the status of the session. There are three possible values for the status attribute: `executing`, `terminating`, `terminated`. When the simulation is started, it's status is `executing`. When the user click the “terminate session” link in the reports summary page, status changes to “terminating”. The core simulation will check the status at the beginning of each time step by querying status from database. If status is “terminating”, the program will change the status to “terminated” and terminates itself. This enables external processes to communicate a process invoked by a Java servlet. It's hard to kill a process explicitly using the “kill” command in Java, because it's not easy to get the process id (pid) of a process invoked by Java, since Java is platform independent and there is no such concept of pid in some platforms. This technique is similar to capability proficed by Java Message Services (JMS).

After invoking the simulation, the reports page will accept two parameters: `user_id` and `session_sid`. The users can view reports for each session they invoked. The table `nom_sessions` is in a centralized database which is different from the databases storing the simulation data. The reports page needs to first retrieve information from the `nom_sessions` table to identify which database stores the simulation data, then query the identified database. Table 3.1 shows some records in the `nom_sessions` table.

Table 3.1. The NOM_SESSIONS Table

SESSION_ID	USER_ID	SID	STATUS
53	1	3	terminated
54	1	1	terminated
55	1	2	terminated
56	1	3	terminating
57	1	1	executing

The following Java code shows the procedure to load balance the database servers.

```

Connection conn=DriverManager.getConnection(url, user, passwd);

Statement stmt=conn.createStatement();
ResultSet rset=stmt.executeQuery("
    select session_id.nextval from dual");
while(rset.next())
    sessionid=rset.getInt(1);

int sid=(sessionid%3)+1;

stmt.executeUpdate("insert into nom_sessions values (" +
    sessionid+", "+StartMolecule.userId+", "+sid+"");

String nom_url=null;
switch(sid){
case 1:
    nom_url="jdbc:oracle:thin:@foyt.cse.nd.edu:1521:simu2";
    break;
case 2:
    nom_url="jdbc:oracle:thin:@bigband.cselab.nd.edu:1521:mynom";
    break;
case 3:
    nom_url="jdbc:oracle:thin:@symphony.cselab.nd.edu:1521:etech";
    break;
default:
    break;
}

```

The three database servers are used in a round robin fashion. If sid is 1, data will be stored on foyt; if sid is 2, data will be stored on bigband; if sid is 3, data will be stored on symphony. It is easy to see that, if multiple simulations are running, database servers will be load balanced.

3.2.3 Fail Over

Multiple application servers and database servers enable us reduce the down time of the simulation system. The idea of fail over is that if the simulation detects that some node involved in the simulation is down, it should be failed over to another peer node. For example, if the simulation is intended to run on joy and store data on foyt, but there is error connecting to the assigned database, then our program will choose another database to connect to, until a connection is successful. This greatly increases the availability of the whole simulation system. To have a quantitative measure of the improvement, let's simply assume that each machine has a probability p of down time. If fail over is not utilized, the possibility that a simulation can not run successfully is $1 - (1 - p)^2 = 2p - p^2$. If fail over is utilized, the probability that a simulation will not run is reduced to $1 - (1 - p^2)(1 - p^3) = p^2 + p^3 - p^5$. For example, let's assume that $p=0.01$, then the down probability is reduced from 0.0199 to 0.0001. This is a 20 times improvement.

Fail over to another database can be implemented in the following way. When a database connection can not be established, a runtime exception will be thrown by the simulation program. We simply catch the simulation and try to connect to another database, until we succeed.

Fail over to another application server can be implemented as follows. The Web JSP page will provide links to a Java Servlet which will invoke a simulation. If a simulation is invoked by the Servlet successfully, a "success" message will be sent by

the Servlet to the JSP page. If the simulation is not successfully invoked, a “failure” message will be sent to the JSP and JSP will redirect to the Java Servlet on another application server. Since we have only two application servers, if the simulation fails again, then we stop trying. The user will be notified that system is not available.

CHAPTER 4

CORE SIMULATION ENGINE AND WEB INTERFACE

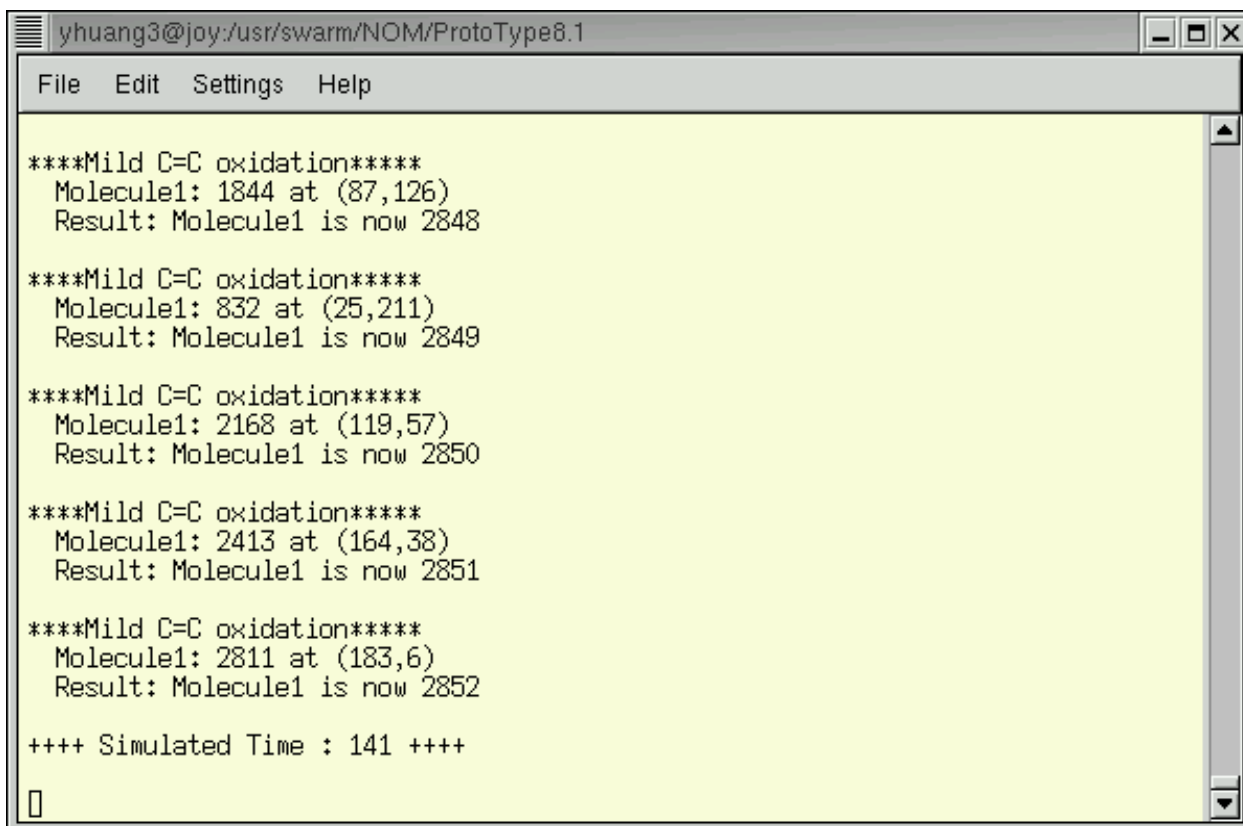
The NOM simulation is implemented using Java and the Swarm library. It can be “compiled once, run everywhere”. The simulation can run either in batch mode or in graphical user interface (GUI) mode. Since our main purpose is to let the user invoke the simulation through web and obtain reports of the simulation, the GUI mode will not be available for them, i.e., the user will not see how molecules move and react, but the state of the simulation for each time step is completely stored in the Oracle databases. The users can get the same information such as the trajectory of a molecule from the simulation databases. Figure 4.1 and Figure 4.2 show the two modes of the simulation respectively.

In the following sections, we discuss the simulation objects, reactions and processes, simulation inputs and outputs, and finally, the web interface.

4.1 Objects, Reactions and Processes

The NOM simulation system involves various NOM molecules. These molecules could be macromolecules or micro-molecules. The macromolecules could be polysaccharides, protein, polynucleotide, tannin, lignin structure, polyterpene and cutin. The micro-molecules may consist of phospholipids, sugars, amino acids, flavonoids and quinones.

To present each type of molecules, a Java class called Molecule is created. Each molecule in the simulation is represented as numbers of different atoms and func-



The image shows a terminal window with a title bar containing the path 'yhuang3@joy:/usr/swarm/NOM/ProtoType8.1'. The window has a menu bar with 'File', 'Edit', 'Settings', and 'Help'. The main area contains the following text:

```
****Mild C=C oxidation****  
Molecule1: 1844 at (87,126)  
Result: Molecule1 is now 2848  
  
****Mild C=C oxidation****  
Molecule1: 832 at (25,211)  
Result: Molecule1 is now 2849  
  
****Mild C=C oxidation****  
Molecule1: 2168 at (119,57)  
Result: Molecule1 is now 2850  
  
****Mild C=C oxidation****  
Molecule1: 2413 at (164,38)  
Result: Molecule1 is now 2851  
  
****Mild C=C oxidation****  
Molecule1: 2811 at (183,6)  
Result: Molecule1 is now 2852  
  
++++ Simulated Time : 141 ++++
```

Figure 4.1. Simulation running in batch mode

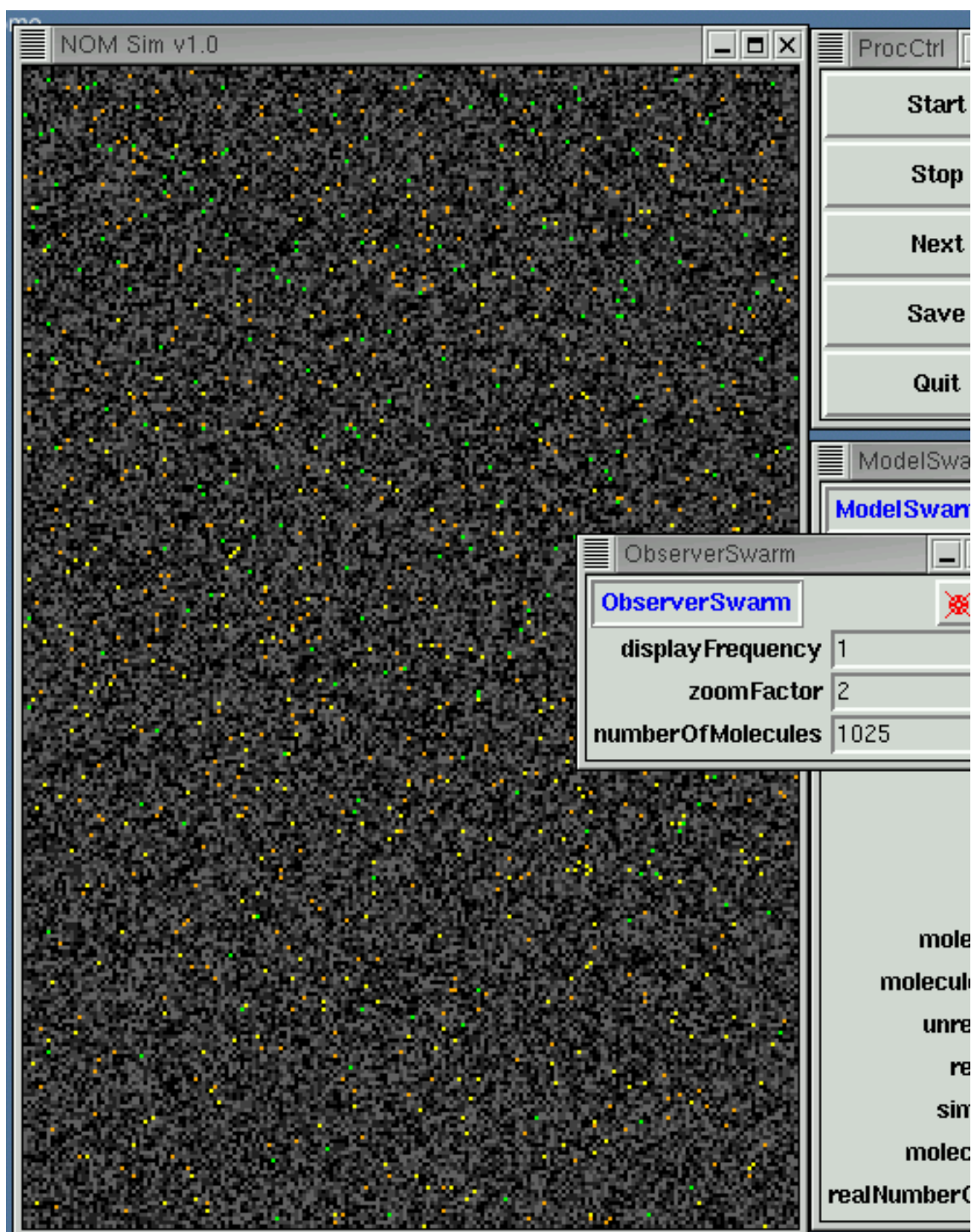


Figure 4.2. Simulation running in GUI mode

tional groups, and its position in the simulated world. More precisely, each instance of Molecule has the following attributes: number of C, N, O, H, S and P atoms; number of such functional groups as phenyl groups, alcohols, phenols, ethers, esters, ketones, aldehydes, acids, aryl acid, amines, ring N, amides, thioethers, thiols, phosphoesters, H-phosphoesters, phosphates; x, y coordinates of the molecule in the simulated world which is a 2-dimensional lattice; and some other attributes such as probabilities of chemical reactions, etc.

The possible reactions and processes are very complicated since there are too many complex movements of molecules and the interactions between them. Overall, the reactions and processes could be:

- Physical reactions: adsorption to mineral phases, aggregation, formation of a micelle, transport downstream, transport through porous media and volatilization. Note that there could be chemical reactions within physical reactions.
- Chemical reactions: chemical reactions could consists of abiotic bulk reactions and abiotic surface reactions, direct photochemical reactions, indirect photochemical reactions, extracellular enzyme reactions and microbial uptake of small molecules. Note that these chemical reactions will definitely change the attributes of the molecules.

The environment of the simulated world which the molecules reside will affect the reactions and processes of the molecules. The molecules can also affect the environment. In our simulation, the environmental variables include simulation time step, microbe density, fungal density, pH value, temperature, pKw, oxygen density, and light density, etc.

The simulation is implemented in a discrete 2D space with discrete time. The space is a rectangular lattice. Each molecule can occupy at most one cell, and

each cell can host at most one molecule. During execution of the simulation, each molecule may move to another location according to simple rules predefined in physical processes. In chemical reactions, one molecule could split and occupy two cells in the simulated world; two or more molecules could combine and occupy only one cell.

4.2 Program Structure

Figure 4.3 shows the Unified Model Language (UML) class diagram that defines the program structure [2]. Only some of the core classes are listed in the class diagram.

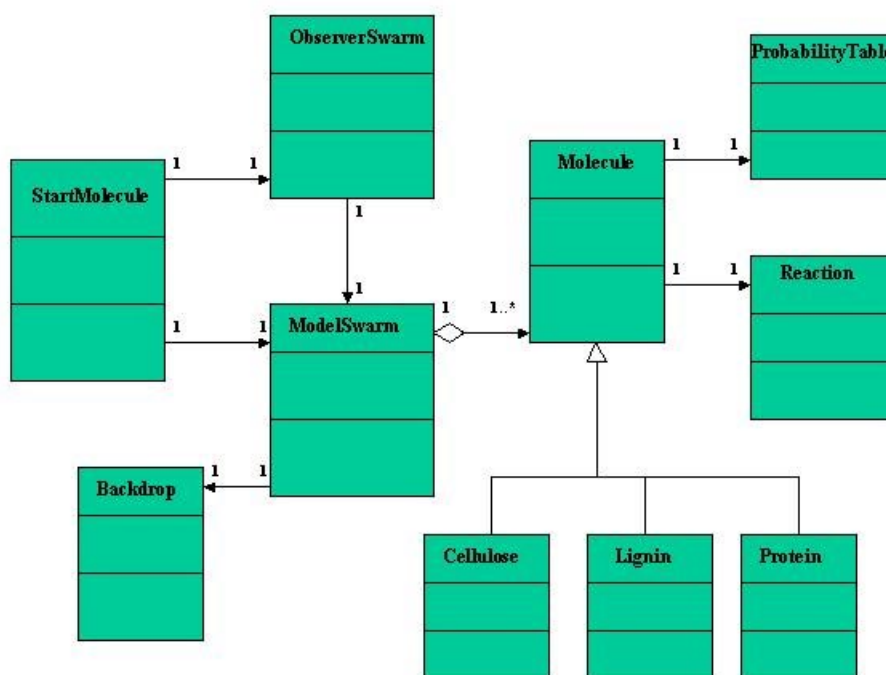


Figure 4.3. The class diagram

A *class diagram* gives an overview of a system by showing its classes and the relationship among them. Class diagrams are static - they display what interacts but not what happens when they do interact.

The class diagram in Figure 4.3 models our simulation system. The central class is **StartMolecule**. Associated with it are **ObserverSwarm** running the program in GUI mode and **ModelSwarm** running the program in batch mode. If the program is running in GUI mode, **ModelSwarm** is contained in **ObserverSwarm**. The **ModelSwarm** contains a list of **Molecules**, each with its associated **ProbabilityTable** and **Reaction**. A **Molecule** could be one of the predefined **Cellulose**, **Lignin** or **Protein**. (In the simulation, new molecule types can be constructed using the Molecule Editor.) A **Backdrop** is associated with the **ModelSwarm** when a molecule is in adsorption state (one of the possible physical processes of a molecule in its simulated world).

UML class notation is a rectangle divided into three parts: class name, attributes, and operations. Relationships between classes are the connecting links. Our class diagram has three kinds of relationships:

- **association** – a relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other in order to perform its work. In a diagram, an association is a link connecting two classes.
- **aggregation** – an association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole. In our diagram, **ModelSwarm** has a collection of **Molecules**.
- **generalization** – an inheritance link indicating one class is a superclass of the other. A generalization has a triangle pointing to the superclass. **Molecule**

is a superclass of **Cellulose**, **Lignin**, and **Protein**.

An association has two ends. A **navigability** arrow on an association show which direction the association can be traversed or queried. Associations with no navigability arrows are bi-directional. The **multiplicity** of an association end is the number of possible instances of the class associated with a single instance of the other end. In our class diagram, there can be only one **ModelSwarm** for each **Molecule**, but a **ModelSwarm** can have any number of **Molecules**.

Figure 4.4 shows the UML use case diagram of the program. **Use case diagrams** describe what a system does from the standpoint of an external observer. The emphasis is on *what* a system does rather than *how*. Use case diagrams are closely connected to scenarios. A **scenario** is an example of what happens when someone interacts with the system. Here is a scenario that a user invoke the simulation either in GUI mode or batch mode. The observerSwarm updates the GUI and probe (which contains instance variables that describe the referent's class and type), the modelSwarm executes the scheduled simulation, updates each molecule, updates the simulated world, writes simulation data to the database.

A **use case** is a summary of scenarios for a single task or goal. An **actor** initiates the events involved in that task. Actors are simply roles that people or objects play. For example, in Figure 4.4, an instance of modelSwarm is an actor, **execute simulation schedule** is a use case. The connection between actor and use case is a **communication association** (or **communication** for short). A use case diagram is a collection of actors, use cases, and their communications.

Figure 4.4 shows how the simulation system works in one time step.

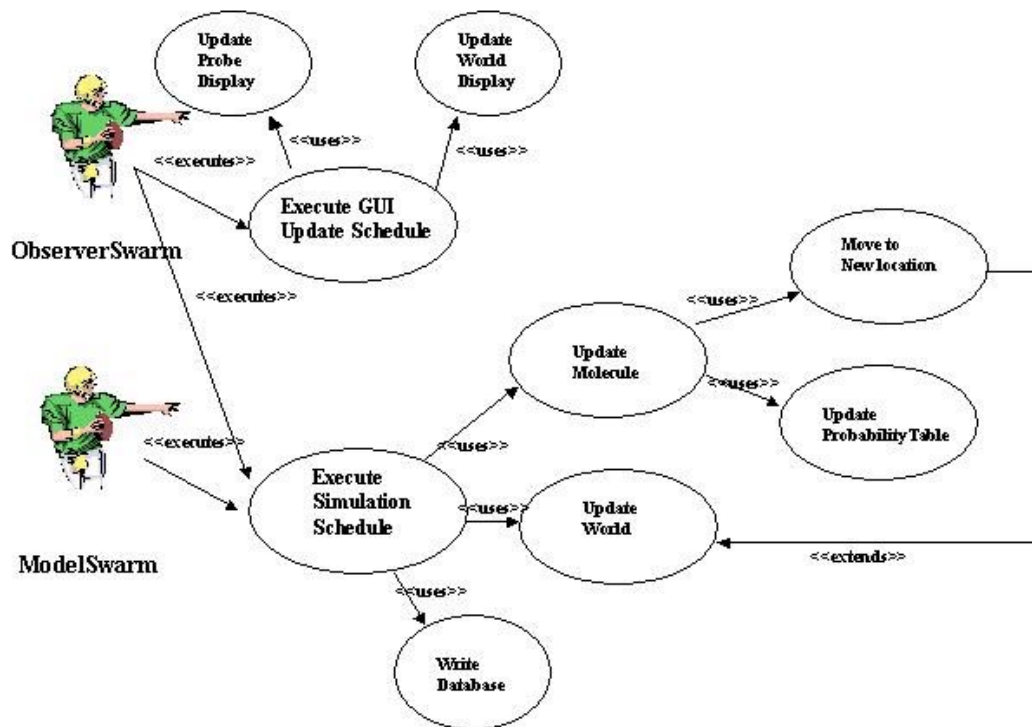


Figure 4.4. The use case diagram

4.3 Web Interface

The web interface allows users to access the simulation system through web browsers such as Netscape and IE. It contains a Molecule Editor which receives user input and stores the input in database. When the user registers with the simulation system, he has a unique user_id, a record of environment variables and a set of molecule information. Figure 4.5 shows the UML diagram that defines the users information database.

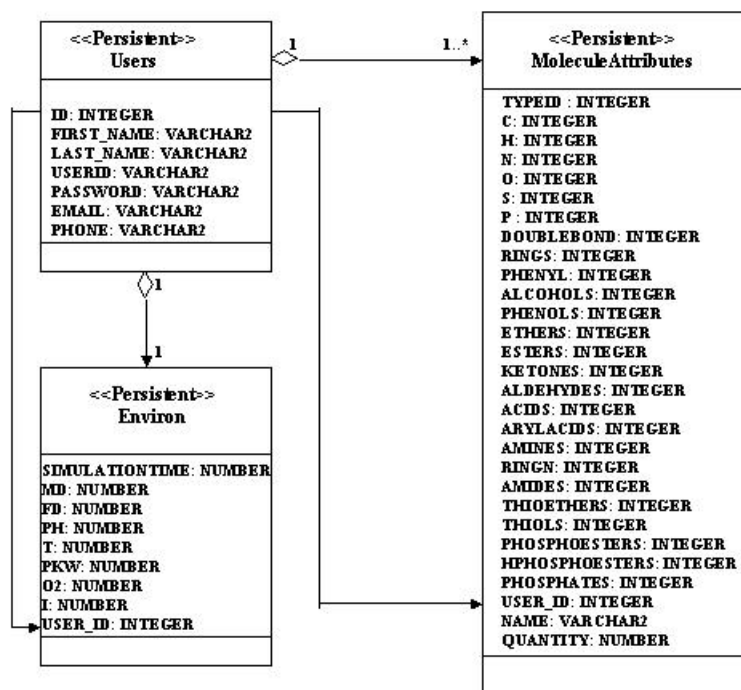


Figure 4.5. UML class diagram for the users information database

From the diagram, we see that there are three tables, namely, USERS, ENVIRON and MOLECULEATTRIBUTES. The table USERS is the owner of both

ENVIRON and MOLECULEATTRIBUTES. The attribute ID is the primary key of USERS. The foreign key USER_ID in ENVIRON and MOLECULEATTRIBUTES references the primary key of USERS. The table ENVIRON has USER_ID as its primary key, while the table MOLECULEATTRIBUTES has TYPEID as its primary key. Oracle provides the ability to declare a *sequence*, which is an object that generates unique integers in sequence. The user information database uses two sequences USER_ID and TYPE_ID to generate the ID in USERS and the TYPEID in MOLECULEATTRIBUTES when new records are inserted into the tables.

The web interface uses the three tables as the backend to store user information. Next we will show the web interface pages which are written in JSP and use JDBC to insert user input to the database.

Figure 4.6 shows the home page of the simulation system.

New users are asked to sign up. Existing users will fill in their userid and password to login to the system. Figure 4.7 shows the sign up page for new users.

A new user needs to choose a unique USERID which is a string with length at most 10. The USERID is different from the ID generated by the sequence, which is a number. If the USERID exists, then the user will be asked to choose another userid. The user needs to provide a password and some other information such as first name, last name, email and phone. After signing up, the user can go to the home page and login to use the Molecule Editor. The simulation configuration wizard contains four steps. Step 1 is a short introduction of the wizard. Figure 4.8 shows the introduction step. Step 2 will ask user to provide environment variables. If the user has already provided this information before, the information is retrieved from the database, and the user can either edit the information or choose to skip this step. Figure 4.9 shows the environment step of the wizard.

Step 3 is the step to edit molecule information, and is separated into step 3a and

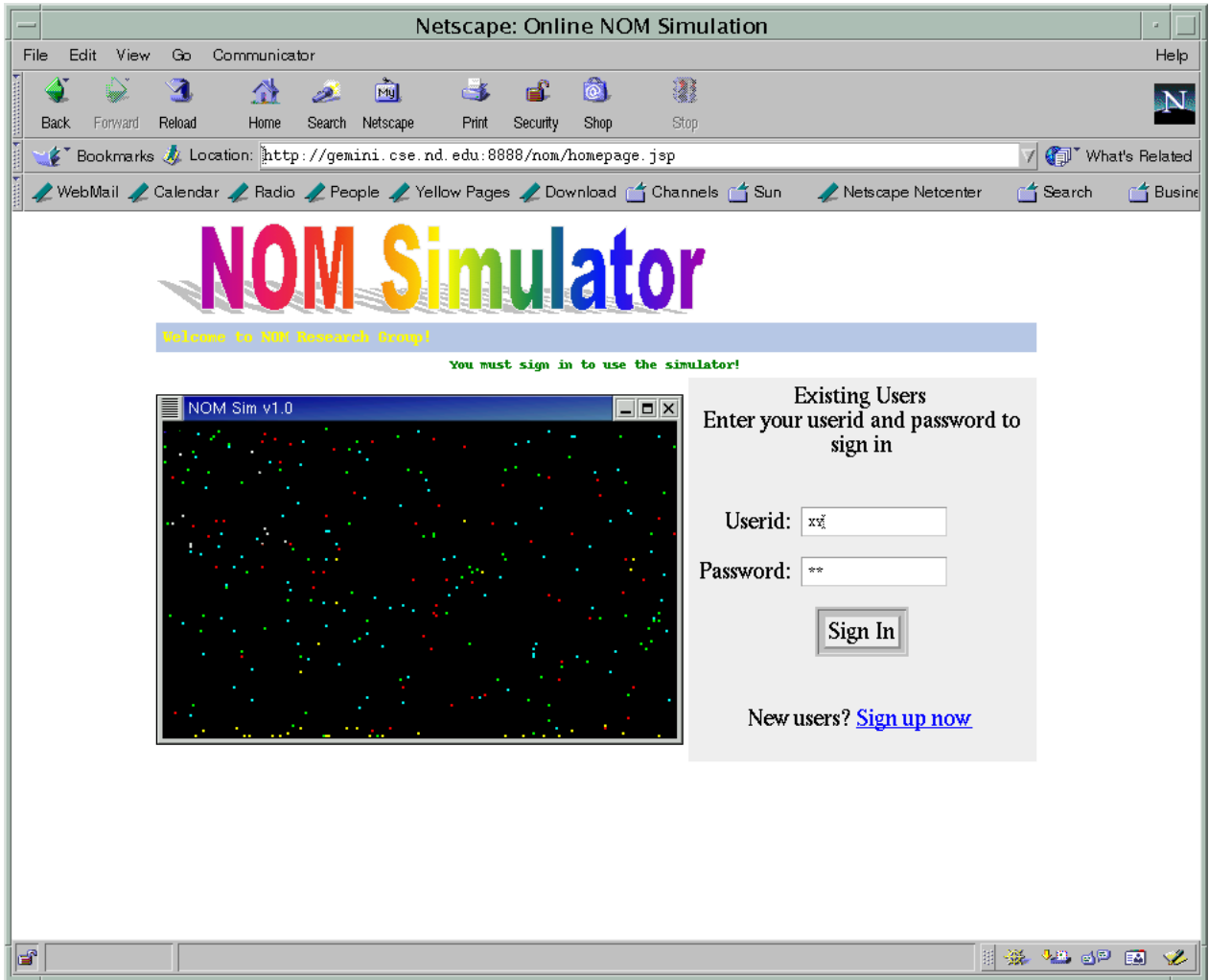


Figure 4.6. The home page of the simulation system

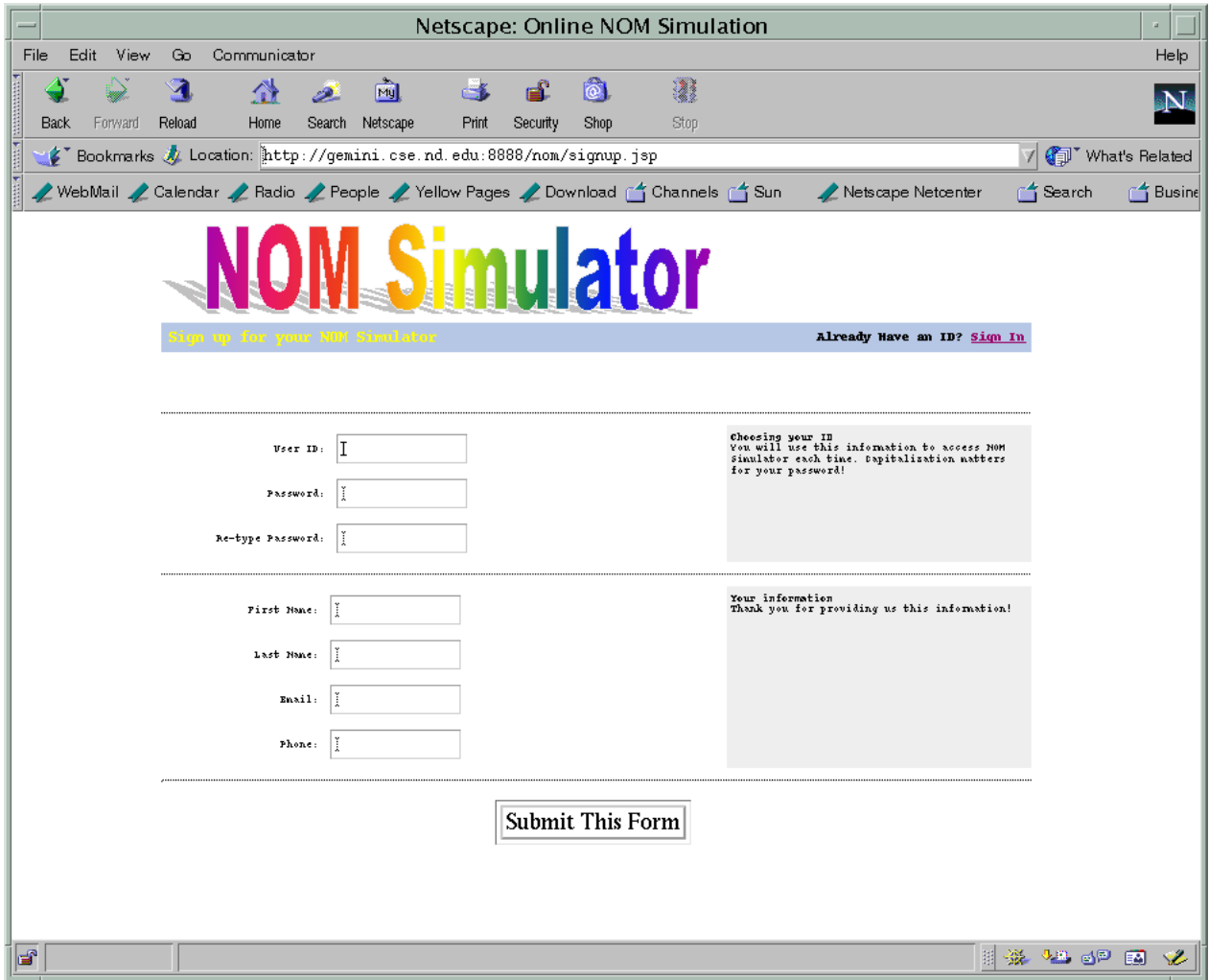


Figure 4.7. The sign up page for new users

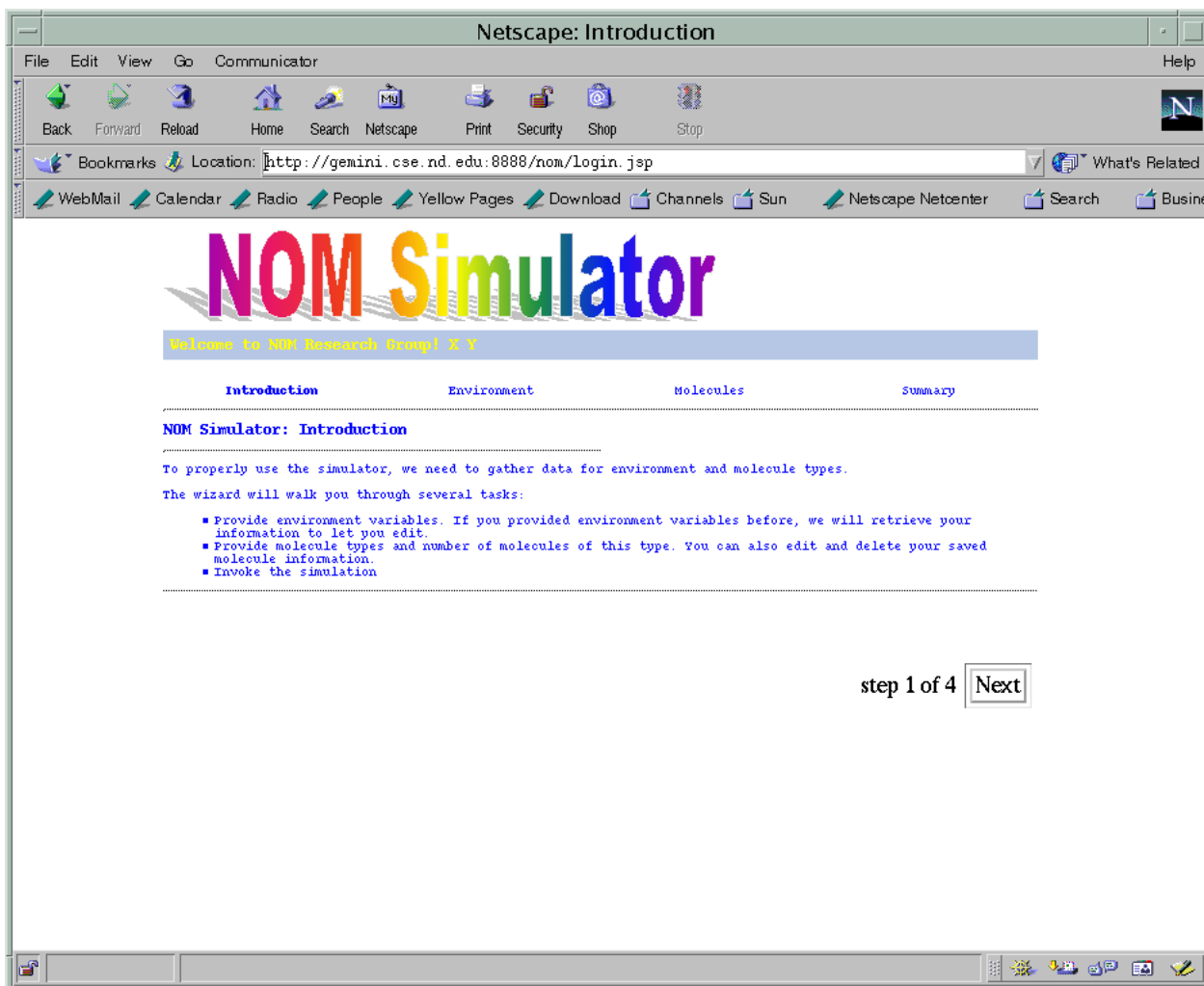


Figure 4.8. The introduction step of the simulation configuration wizard

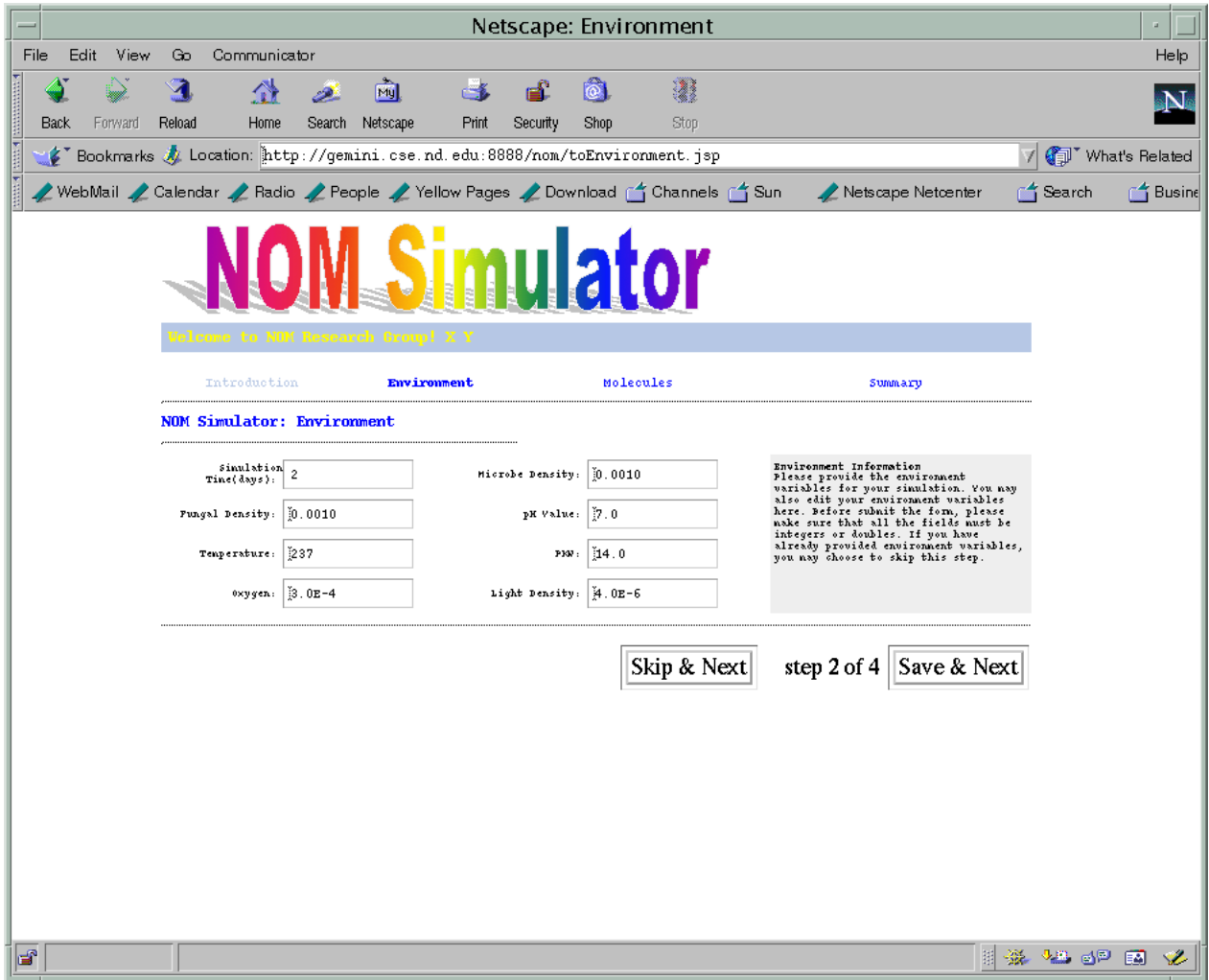


Figure 4.9. The environment step of the simulation configuration wizard

step 3b. In step 3a, the user will be asked to specify the percentage of the three built-in molecules: Cellulose, Lignin and Protein, in the whole set of the molecules. The users can skip this step if they decide not to include the three kinds of the molecules in their total molecule set. They can also specify percentage 0 to indicate that they do not choose any of the three molecules. Figure 4.10 shows this step. In step 3b, the users can edit new molecule types; they can also skip this step if they provided this information before; they can also delete molecules they entered before, by click the links at the bottom of the page. Figure 4.11 shows this step.

After the users provide all the information about environment and molecules, step 4 will let them invoke the simulation. The simulation is invoked by a Java servlet. The Java Servlet is on either joy or tenor and it will invoke simulation on joy or tenor respectively. The user input is stored in an Oracle database, when the user invokes the simulation, the core program will load all the user input and start the simulation process. Figure 4.12 shows this step.

After the users invoke the simulation, they will be notified that their simulation is invoked successfully and they can click a link to go to the reports pages. Figure 4.13 shows this step.

The users can invoke multiple sessions, and they can view reports for each session. The reports summary page lists all the sessions invoked by the user, the most recent session is listed on the top. The status of the sessions is also listed in the page. The users can terminate their simulation by clicking the corresponding “terminate session” link. After the user terminates the session, the status will become “terminated”. Figure 4.14 shows this step.

If the users click one of the reports link, they will be able to see the simulation results on line. Figure 4.15 shows the chemical reactions statistics of the simulation.

Netscape: Environment

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop Stop

Location: <http://gemini.cse.nd.edu:8888/nom/toKnown.jsp> What's Related

WebMail Calendar Radio People Yellow Pages Download Channels Sun Netscape Netcenter Search Business

(Atom) S:	0	0	0
(Atom) P:	0	0	0
Double Bond:	60	199	59
Total Ring Structures:	60	40	5
Phenyl Groups:	0	40	5
Alcohols:	182	1	10
Phenols:	0	1	0
Ethers:	119	118	0
Esters:	0	0	0
Ketones:	0	0	0
Aldehydes:	0	0	0
Acids:	0	0	6
Aryl Acid:	0	0	0
Amines:	0	0	6
Ring N:	0	0	0
Amides:	0	0	54
Thioethers:	0	0	0
Thiols:	0	0	0
Phosphoesters:	0	0	0
H-phosphoesters:	0	0	0
Phosphates:	0	0	0
Percentage:	<input type="text" value="0.33"/>	<input type="text" value="0.33"/>	<input type="text" value="0.34"/>

Skip & Next step 3a of 4 Save & Next

Figure 4.10. Step 3a of the Molecule Editor

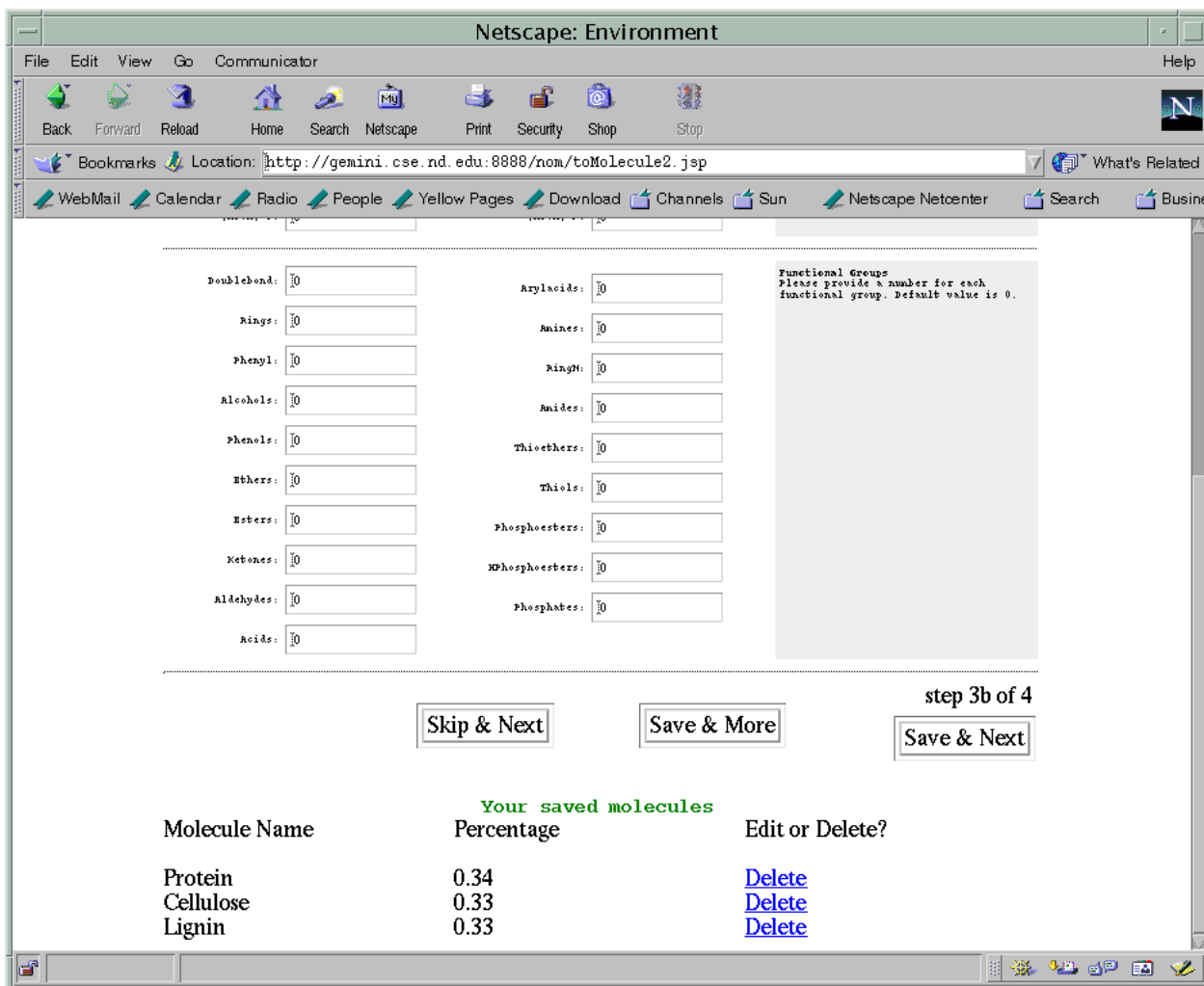


Figure 4.11. Step 3b of the Molecule Editor

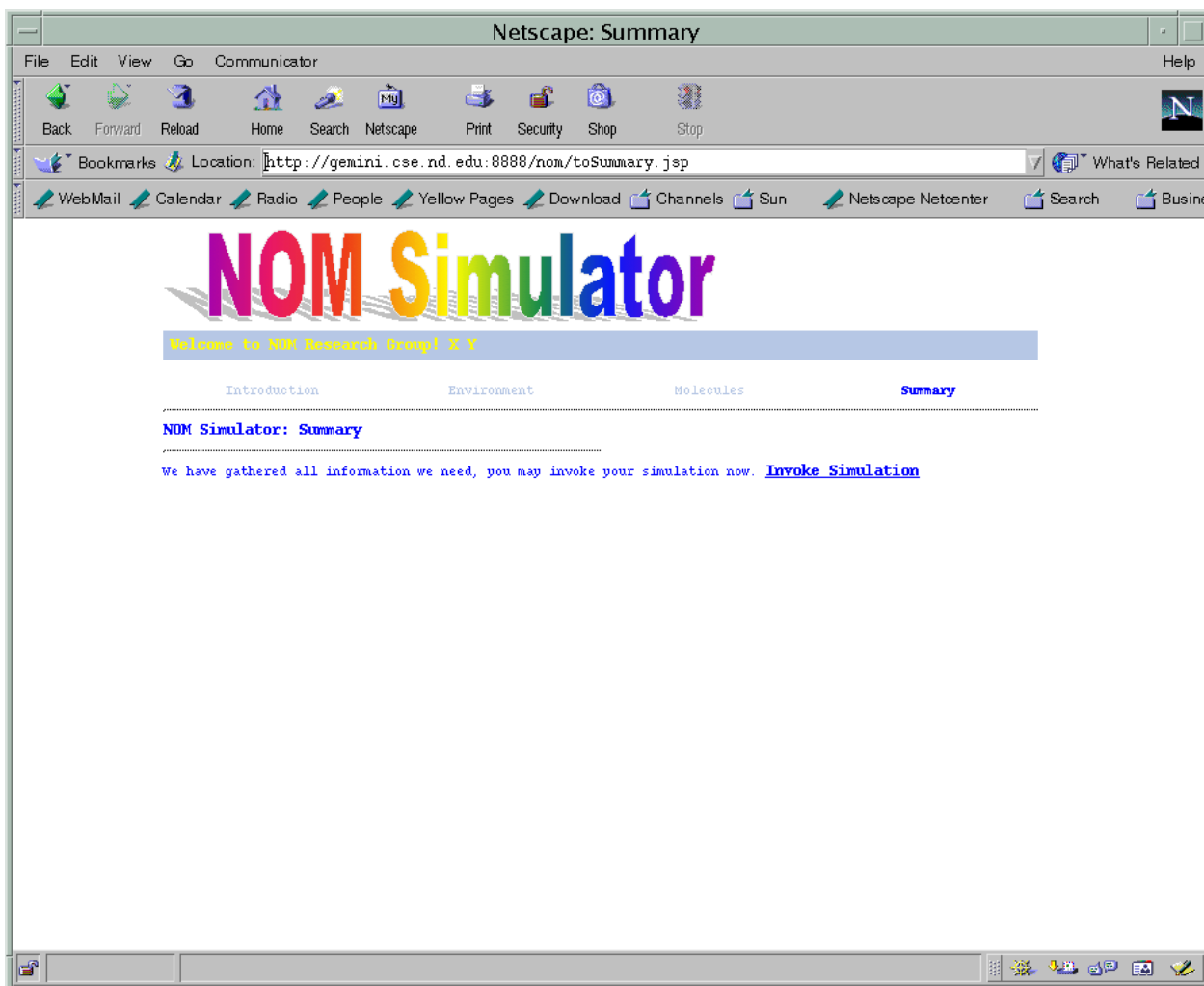


Figure 4.12. Step 4 of the simulation configuration wizard

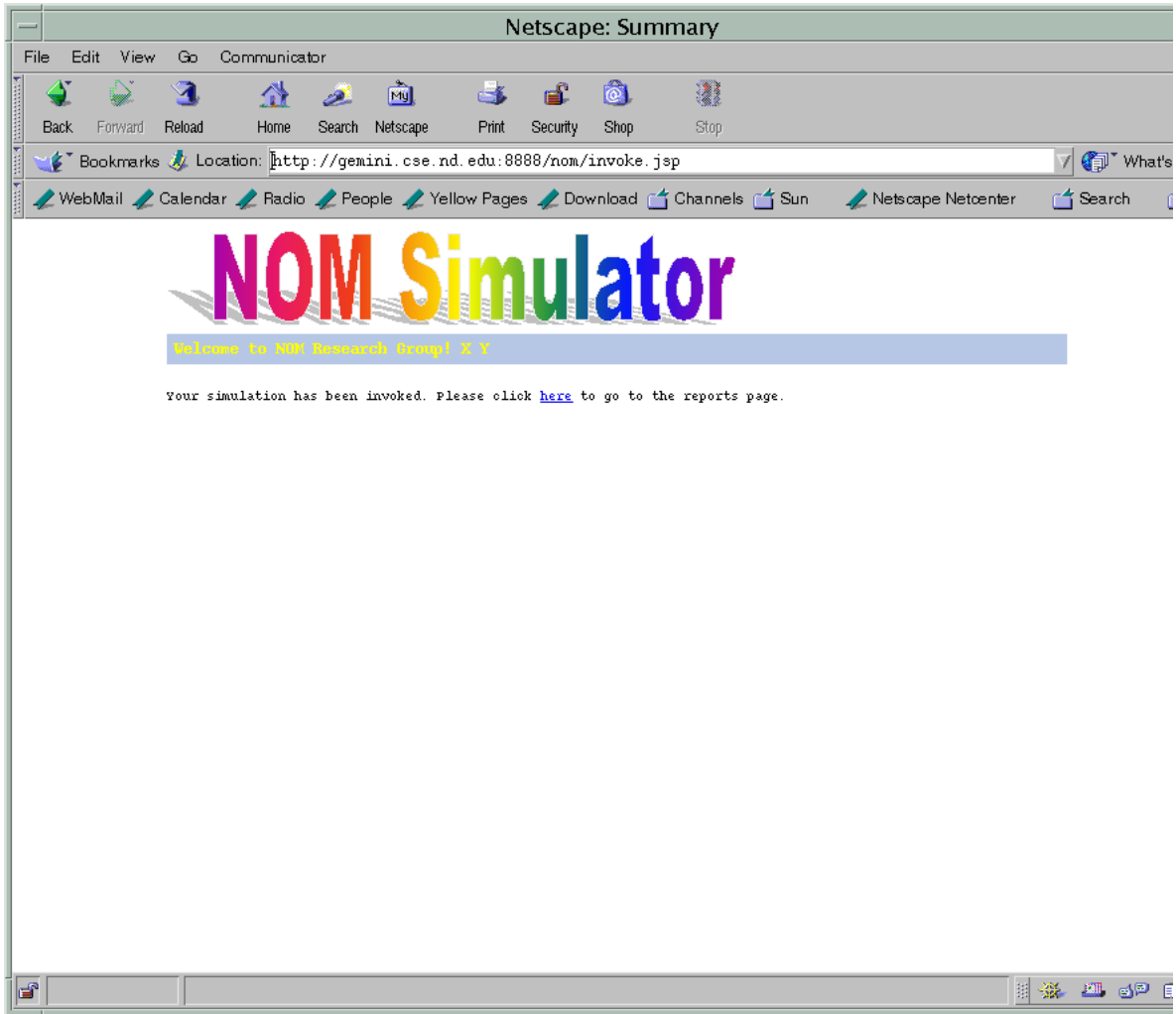


Figure 4.13. Simulation invoked successfully

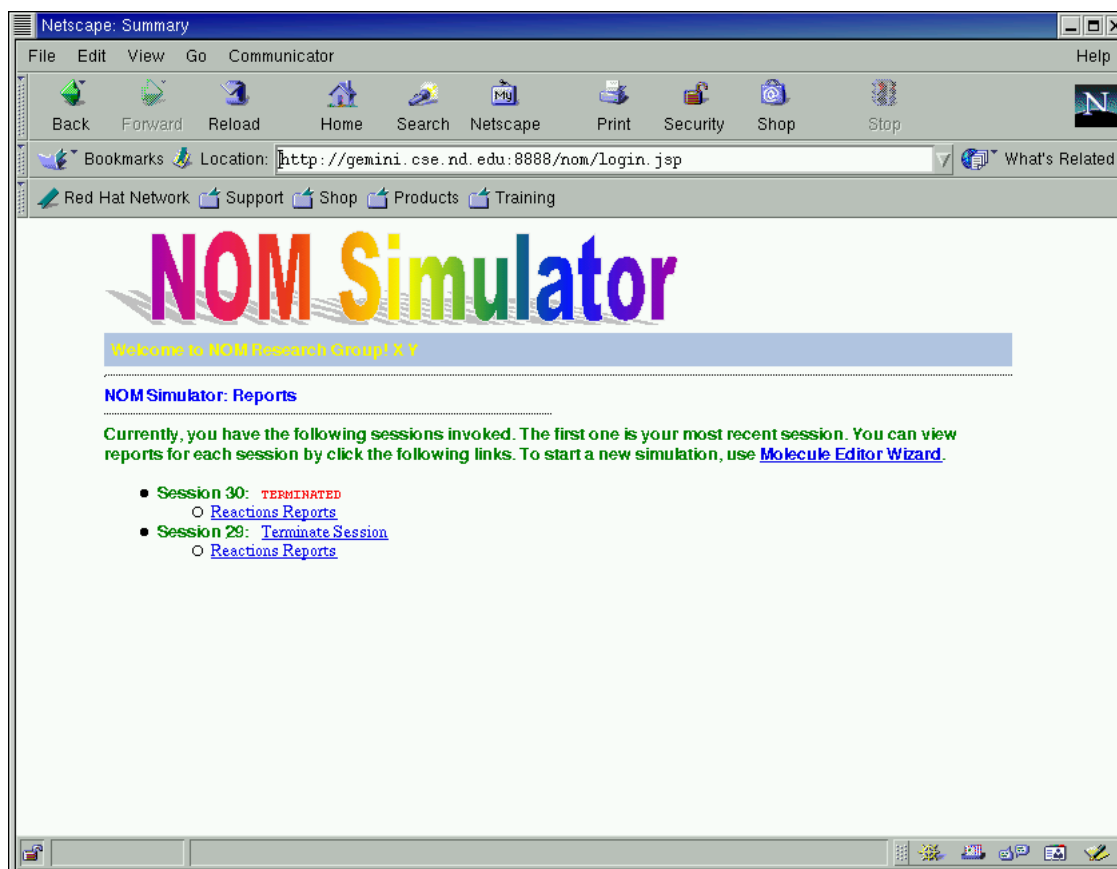


Figure 4.14. Reports summary of the user

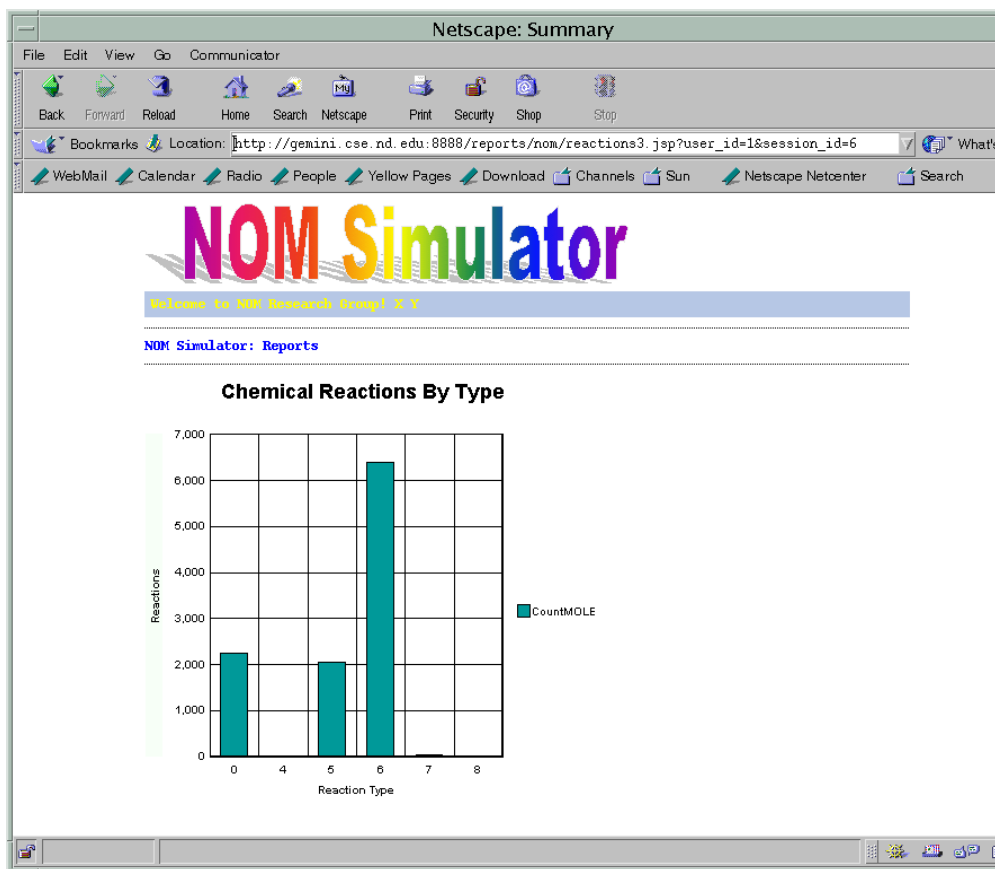


Figure 4.15. Chemical reactions statistics

4.4 Conclusion

In this chapter, we discussed the core simulation program using the Unified Model Language (UML). We listed some of the core classes and their relationships using class diagrams. We also presented a use case diagram which show what the simulation system does when a user invoke the simulation. Finally, we showed the web interface which allows the user to access the simulation system. In next chapter, we will discuss in detail how we create reports page and how we “tune” the query performance.

CHAPTER 5

QUERY OPTIMIZATION

As mentioned in chapter 4, users will not be able to use the simulation system in graphical user interface (GUI) mode, unless they connect to our system using an X-terminal. All the activities of the system are recorded in the Oracle databases. It is a critical task to both insert and retrieve information from the huge database efficiently. According to our measurements, 100 records are inserted into the database by each simulation every second. It is not a trivial task to query the database efficiently while preserving the insertion performance. In this chapter, we define query optimization to include both insertion optimization and query optimization.

5.1 Database Design to Optimize Insertion

We need to create multiple OLTP (Online Transaction Processing) databases on the database servers. We want the simulation data to be inserted into the databases fast, and information to be retrieved from the databases fast. We can not meet both goals, since there two goals somewhat contradict to each other when the databases are inserted and queried at the same time. For example, for better query performance, we may create necessary indexes for some columns, but to maintain and update the indexes will be an huge overhead for the insert operations, since each insertion will update the index structures and storages.

Next we show how we create the database objects in the three database servers to store the simulation data. Note that database schema objects structures are the

same on all the databases to enable load balance and fail over as mentioned in the previous chapters.

5.1.1 Information Stored in Databases

We try to store all information about the simulation system at any time step. (Recall, we use discrete time to simulation the continuous time.) At any time step, there are many molecules in the system. We need to record every movement and reaction of these molecules. For example, the location of the molecule in the system, whether the molecule is involved in chemical reactions, and what new molecules are produced during a chemical reaction.

As we mentioned in previous chapters, the “real world” system is quite complex. There could be many chemical reactions in the real world. But it is not possible to allow all types of chemical reactions to occur in the simulation. To simplify the simulation system, only 10 types of most likely chemical reaction types are considered. The following Table 5.1 lists all the chemical reaction types used in the simulation.

Table 5.1. The ReactionType Table

Reaction Type	Reaction Name
0	Ester condensation
1	Ester hydrolysis
2	Amine hydrolysis
3	Microbial uptake
4	Dehydration
5	Strong C=C oxidation
6	Mild C=C oxidation
7	(C-O-H) oxidation
8	Aldehyde C=O oxidation
9	Decarboxylation
10	No reaction

We create a table NOM to store the status of the simulation at any time step. The following Table 5.2 defines the structure of the table NOM.

The attribute SESSIONID is the identifier for a session invoked by a user. USER_ID is the identifier for the user who invokes the session. TIMESTAMP records the time stamp of the system. MOLECULEID is the identifier of the molecule in the system. Each molecule has a different MOLECULEID. XPOS and YPOS denote the position of the molecule in the system. REACTED denotes whether the molecule is involved in a chemical reaction. If it is, REACTIONTYPE denotes the reaction type, otherwise REACTIONTYPE is NULL. REACTIONTYPE is a foreign key which references the table REACTTIONTYPE. If the reaction is a first order reaction (only one molecule involved in the reaction), PARENT1 will denote the moleculeid of a molecule which produced the molecule, and PARENT2 will be NULL. If the reaction is a second order reaction (two molecules involved in the reaction), PARENT1 and PARENT2 will denote the parents of the molecule, i.e., the molecules which produce this molecule. We will not consider third order (or above) reactions, since the probability of this kind of reactions is so small that they can be ignored.

The attributes from C to PHOSPHATES define the physical structure of the molecule. The attributes from PROB_0 to PROB_9 denote the probabilities for the 10 types of chemical reactions. In each time step, all the molecules currently in the system will contribute one row to the table NOM. For best insertion performance, we intentionally disabled all constraints which are used to ensure data integrity, since these constraints will be enforced by the database for each insertion, thus a huge impact for the insertion performance. This is one of the methods to tune insertion performance. Other methods include disabling indexes, table creation parameters, memory allocations, etc. In the next section, we discuss other table

Table 5.2. The structure of the table NOM

Attribute Name	Data Type
SESSIONID	NUMBER(38)
MOLECULEID	NUMBER(38)
PARENTID1	NUMBER(38)
PARENTID2	NUMBER(38)
XPOS	NUMBER(38)
YPOS	NUMBER(38)
REACTED	VARCHAR2(3)
REACTIONTYPE	NUMBER(38)
TIMESTAMP	NUMBER(38)
USER_ID	NUMBER(4)
C	NUMBER(38)
H	NUMBER(38)
N	NUMBER(38)
O	NUMBER(38)
S	NUMBER(38)
P	NUMBER(38)
DOUBLEBOND	NUMBER(38)
RINGS	NUMBER(38)
PHENYL	NUMBER(38)
ALCOHOLS	NUMBER(38)
PHENOLS	NUMBER(38)
ETHERS	NUMBER(38)
ESTERS	NUMBER(38)
KETONES	NUMBER(38)
ALDEHYDES	NUMBER(38)
ACIDS	NUMBER(38)
ARYLACIDS	NUMBER(38)
AMINES	NUMBER(38)
RINGN	NUMBER(38)
AMIDES	NUMBER(38)
THIOETHERS	NUMBER(38)
THIOLS	NUMBER(38)
PHOSPHOESTERS	NUMBER(38)
HPHOSPHOESTERS	NUMBER(38)
PHOSPHATES	NUMBER(38)

Table 5.3. The structure of the table NOM (continued)

PROB_0	NUMBER(7,5)
PROB_1	NUMBER(7,5)
PROB_2	NUMBER(7,5)
PROB_3	NUMBER(7,5)
PROB_4	NUMBER(7,5)
PROB_5	NUMBER(7,5)
PROB_6	NUMBER(7,5)
PROB_7	NUMBER(7,5)
PROB_8	NUMBER(7,5)
PROB_9	NUMBER(7,5)

creation parameters which affect the performance of insertion.

5.1.2 Parameters While Creating Table NOM

In this section, we discuss some parameters in the SQL statement “create table NOM”. These parameters describe aspects of managing space in data blocks. Our goal is to adjust these parameters to efficiently utilize the disk space and meanwhile to reduce the probability of “chaining” (one row of a table spans two data blocks) which is a huge impact on query performance since it increases the number of I/Os. The size of a data block is specified at database creation, it is a multiple of OS block size which is normally 512 bytes. In this thesis, the data block size is chosen to be 8192 bytes.

The PCTFREE and PCTUSED parameters are physical attributes that can be specified when a table is created or altered. These parameters allow you to control the use of the free space within a data block. This free space is available for inserts and updates of rows of data.

The PCTFREE and PCTUSED parameters allow you to:

- Improve performance when writing and retrieving data

- Decrease the amount of unused space in data blocks
- Decrease the amount of row chaining between data blocks

The INTRANS and MAXTRANS parameters are also physical attributes that can be specified when tables are created or altered. These parameters control the number of concurrent update transactions allocated for data blocks of a table, which in turn affects space usage in data block headers and can have an impact upon data block free space.

- PCTFREE: This parameter is used to set the percentage of a block to be reserved for possible updates to rows that already are contained in that block. For example a value 20 indicates that 20% of each data block used for this table's data segment will be kept free and available for possible updates to the existing rows already within each block. Default value for this parameter is 10. Since our table is primary used for insertion, we will choose a low value for this parameter which allows insert to fill the block more completely and ensure that that the disk space is tightly used for the table. In particular we choose a value 0.
- PCTUSED: After a data block becomes full as determined by PCTFREE, Oracle does not consider the block for the insertion of new rows until the percentage of the block being used falls below the parameter PCTUSED. Before this value is achieved, Oracle uses the free space of the data block only for updates to rows already contained in the data block. For example, assume that you specify the following parameter within a CREATE TABLE statement: PCTUSED 40. In this case, a data block used for this table's data segment is not considered for the insertion of any new rows until the amount of used space in the block falls to 39% or less (assuming that the block's used space

has previously reached PCTFREE). The default value for this parameter is 40, here we accept the default, since this parameter is of no importance because the NOM table is rarely deleted or updated.

- **INITRANS and MAXTRANS:** INITRANS specifies the number of DML (refers to data manipulation language, including update, insert and delete) transaction entries for which space is initially reserved in the data block header. Space is reserved in the headers of all data blocks in the associated segment. As multiple transactions concurrently access the rows of the same data block, space is allocated for each DML transaction's entry in the block. Once the space reserved by INITRANS is depleted, space for additional transaction entries is allocated out of the free space in a block, if available. Once allocated, this space effectively becomes a permanent part of the block header. The MAXTRANS parameter limits the number of transaction entries that can concurrently use data in a data block. Therefore, you can limit the amount of free space that can be allocated for transaction entries in a data block using MAXTRANS. The INITRANS and MAXTRANS parameters for the data blocks allocated to a specific table should be set individually for each table based on the following criteria: The space you would like to reserve for transaction entries compared to the space you would reserve for database data; The number of concurrent transactions that are likely to touch the same data blocks at any given time. For example, if a table is very large and only a small number of users simultaneously access the table, the chances of multiple concurrent transactions requiring access to the same data block is low. Therefore, INITRANS can be set low, especially if space is at a premium in the database. Alternatively, assume that a table is usually accessed by many users at the same time. In this case, you might consider preallocating transaction entry space by using a high

INTRANS. This eliminates the overhead of having to allocate transaction entry space, as required when the object is in use. Also, allow a higher MAX-TRANS so that no user has to wait to access necessary data blocks. When multiple simulations are running, the table NOM is inserted simultaneously by these simulations; thus, we will specify these parameters to be 5 and 10 respectively.

5.1.3 Overhead of Constraints and Indexes On Insertion Performance

Referential integrity constraints are added to table to enforce application logic. The constraints that affect insertion performance include check constraints (for example, to check the functional group values are integers), primary key constraints (a unique key to identify a row in the table), and foreign key constraints (for example, USER_ID must exist in the USERS table). When performing insertions, we intentionally disable all these constraints to speed up insertion performance.

Whenever a row is inserted into a table, all indexes in which that the index participates have to be updated in real time. This can dramatically slow down the performance of the system. Figure 5.1 shows that when a record is inserted into the target table, all constraints and indexes are enforced which are overhead for the insertion performance. Therefore, for best insertion, these constraints and indexes should be disabled.

5.2 Query Optimization

The NOM table records every action of all molecules at each time step. It's critical to retrieve information from the table and build reports in an efficient way. One of the major interests of our users is the statistics of the simulation, such as statistics for chemical reactions. We call these type of queries the **aggregation queries**. For illustration purposes, we provide two simple examples here and show

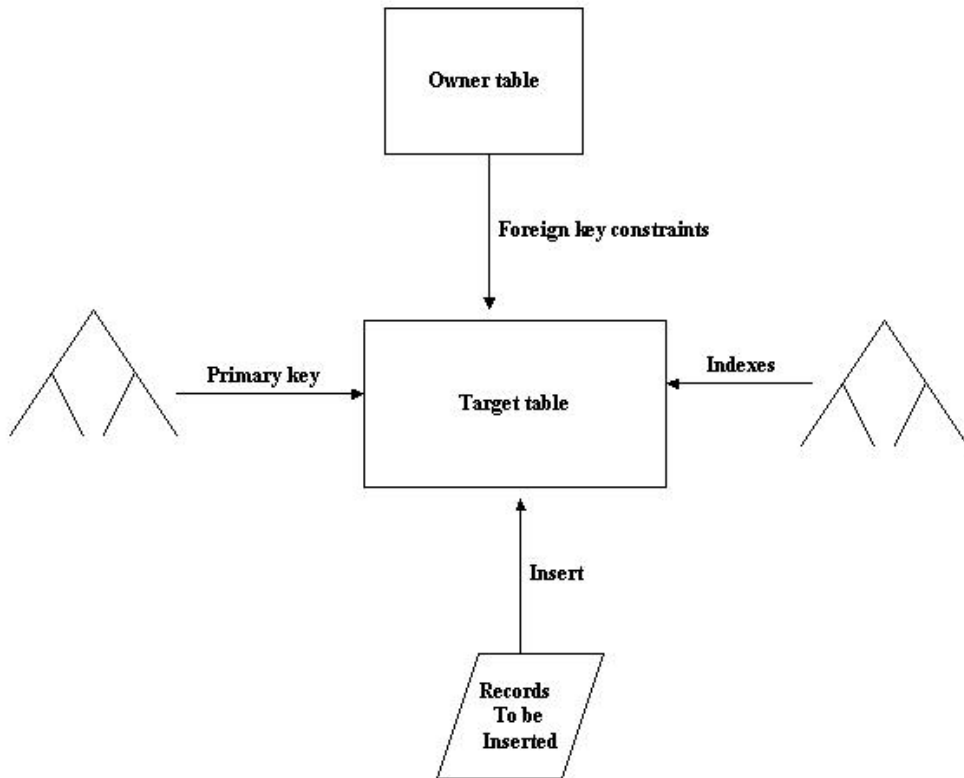


Figure 5.1. Constraints and indexes are overhead for insertion

how to generate fast reports for the two examples.

1. Show the number of chemical reactions for each of the ten reaction types occurred so far in the simulation in bar chart.
2. Create a line graph which shows the trend of the total number of chemical reactions vs time steps.

Figure 5.2 shows the graphs for the two reports, the graphs are generated using Oracle Reports. Since it is a web based application, it is important to generate this reports efficiently (in a few seconds), otherwise, our users will not wait for the reports to show up.

The reports are generated when the simulation is running, i.e., when the tables are growing. Each time you click the refresh or reload button, you will get a different reports since the retrieved information is changed. The reports will receive two parameters, `user_id` and `session_id` which denote the identifier of the user who invokes the simulation and the session invoked by the user. The table `NOM` is shared by all simulations either running or terminated. This table will get huge when lots of simulations are executed. It is not a easy task to get the answers for the two examples. In the following sections, we will discuss our solutions.

5.2.1 The Query Statements

In this subsection, we present the query statement for the two reports against the table `NOM` and other joined tables. Figure 5.3 shows the query statement for report 1 and the corresponding query results, as well as the time needed to run the query statement. In this example, the query completed in just a little more than 10 seconds. Actually, the performance is not bad. But since the reports will be shown in bar chart, it will take more time to generate the graph. Therefore, if we could take some action to speed it up, it would be beneficial.

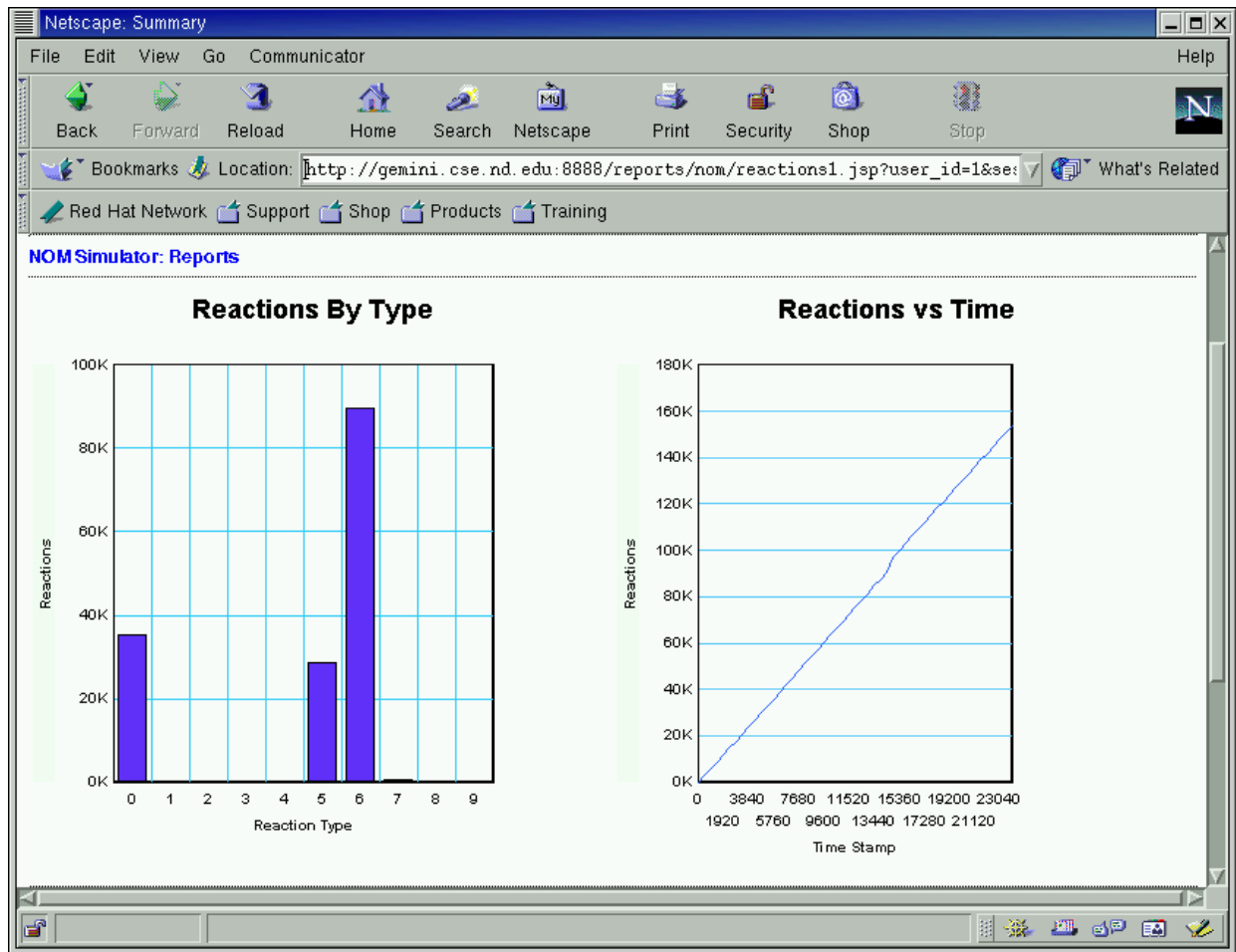


Figure 5.2. Reports for the two examples


```
SQL> select nom.reactiontype "Reaction Type",  
2      reactiontype.rname "Reaction Name",  
3      count(nom.moleculeid) "Reactions"  
4  from nom, reactiontype  
5  where nom.reactiontype=reactiontype.rtype  
6      and sessionid=:session_id and user_id=:user_id  
7  group by nom.reactiontype, reactiontype.rname  
8  order by nom.reactiontype;
```

8 rows selected.

Elapsed: 00:00:10.03

Figure 5.3. Query statement for report 1 and result

Before we discuss more about the query statement, let's show how to build the query statement for report 2. Figure 5.4 shows this query statement.

```
SQL> select t1.timestamp "Time Step",
       sum(t2.total) "Reactions"
  from (select timestamp,
              count(moleculeid) total
        from nom
        where sessionid=:session_id
              and user_id=:user_id
        group by timestamp ) t1,
       (select timestamp,
              count(moleculeid) total
        from nom
        where sessionid=:session_id
              and user_id=:user_id
        group by timestamp ) t2
 where t2.timestamp <= t1.timestamp
  group by t1.timestamp;
```

Figure 5.4. Query statement for report 2

This query statement is quite complicated, and it will take hours to execute. For web reports, this is not acceptable, because it is slow and the reports does not reflect the current statistics of the system. Therefore, it is very important that we can speed up the query or rewrite the query statement. We can create regular B*-tree index, bitmap join index for join operation, partitioned tables, aggregation strategies, etc. to tune the query.

Typically, you insert or load data into a table (using SQL*Loader, Import or JDBC) before creating indexes. Otherwise, the overhead of updating the index slows down the insert or load operations dramatically. Since we don't want to slow down insertion during the simulation, we did not create indexes. Partitioning is a very useful feature in data warehouse applications; we will create partitioned tables in the data warehouse. But partition tables also need indexes to enable partitions. The choice left for us is to use temporary aggregation tables.

The idea of aggregation is pretty simple. We create temporary tables which store the statistics of the system at the end of each time step. Therefore, we will have the information available whenever a query needs it, and bypass computations in the query. This approach will slow down insertion a little, but the gain of query performance is huge.

5.2.2 Temporary Aggregation Tables

For the two queries, we create two tables, namely, REACTIONS_BY_TYPE and REACTIONS_BY_TIME. Table 5.4 and Table 5.5 show the structures of the two tables. In the REACTIONS_BY_TYPE table, REACTIONS denotes the number of chemical reactions occurred so far in the system for the particular RTYPE (reaction type). In the REACTIONS_BY_TIME table, TOTAL denotes the total number of chemical reactions occurred so far at this TIMESTAMP (time step). As can be seen, our reports will just query the two tables, instead of the huge NOM table.

Table 5.4. The REACTIONS_BY_TYPE Table

Name	Null?	Type
SESSION_ID	NOT NULL	NUMBER(38)
RTYPE		NUMBER(38)
REACTIONS		NUMBER(38)

Table 5.5. The REACTIONS_BY_TIME Table

Name	Null?	Type
SESSION_ID	NOT NULL	NUMBER(38)
TIMESTAMP		NUMBER(38)
TOTAL		NUMBER(38)

Next we show how to populate the two tables. In the core simulation program, we need to update or insert records into the tables at the end of each time step. In the REACTIONS_BY_TYPE table, for a particular SESSION_ID, which identifies this session, at the end of the first time step, a new record is inserted into the table. At the end of other time steps, this record is updated. This record stores the number of chemical reactions occurred so far for each reaction type. In the REACTIONS_BY_TIME table, for a particular SESSION_ID, at the end of each time step, a new record is inserted into the table, which stores the total number of chemical reactions occurred so far in the system. As can be seen, the query statements for the two reports is simplified to the following two statements:

- select rtype “Reaction Type”, reactions “Reactions”
from reactions_by_type where session_id=:session_id;
- select timestamp “Time Step”, total “Total Reactions”
from reactions_by_time where session_id=:session_id;

The time to execute the two queries is reduced to 0.01 second and 5.26 seconds respectively. The two queries form the “Data Model” in Oracle Reports. To transform the data model to bar chart and line graph as shown in Figure 5.2, a tool called “Oracle Graphics” is used to generate the graphs. Since there are two many records (over 24 thousands) in the second query, it is hard to plot so many points (one point for one record) in the line graph (where stamp is the x-axis and total

is the y-axis). To get around this, we only need to plot a small portion of sample points from the query. For example, we may just want to plot around 50 points uniformly selected from all the records. Since the table REACTIONS_BY_TIME is growing all the time, we need to know how many records are in the table when we uniformly choose sample points. The following simple method shows how to generate the sample points.

1. To simplify the query, another table with same structure of the REACTIONS_BY_TIME called REACTIONS_BY_TIME_SHORT is created which is used to store the sample records chosen from the big table.
2. The records in REACTIONS_BY_TIME_SHORT with SESSION_ID equal to the current session identifier is deleted first if there are any.
3. The total number of records rows of REACTIONS_BY_TIME is obtained by querying COUNT(timestamp).
4. An integer called steplength is calculated by the expression $\text{ceil}(\text{rows}/50)$ where ceil is the ceiling function.
5. The sample records are those records from REACTIONS_BY_TIME where timestamp is a multiple of steplength and stored in the REACTIONS_BY_TIME_SHORT table.
6. The query statement is changed as follows:

```
select timestamp "Time Step", total "Total Reactions"  
from reactions_by_time where session_id=:session_id;
```

The total time of operations in the above steps is less than 1 second. The generated line graph only needs to plot around 50 points, which greatly reduced the time to generated the whole reports page as shown in Figure 5.2.

5.3 Performance Comparison and Conclusion

We compare the insertion performance of the following three scenarios of insertions. Scenario 1 is the situation in that no indexes are created and no aggregations are generated. Obviously, this scenario will have the poorest query performance with best insertion performance. Scenario 2 is the situation in that indexes for sessionid, user_id, moleculeid and timestamp are created for query purpose, but no aggregations are generated. Scenario 2 will have better query performance than Scenario 1 but with the overhead of maintaining index structures and storages. Scenario 3 is the situation in that aggregations are generated without using any indexes. It will have the best query performance with the overhead of maintain aggregation tables.

Figure 5.5 shows the number of seconds needed in the simulations to generate the numbers of records and insert them into the tables for the above three scenarios. The simulations are running on a Pentium 3 dual 800 mHz Red Hat 7.2 machine with 1.5GB memory. The generated data is inserted into a database server, which is a Pentium 3 dual 733 mHz Win2K machine with 2GB memory over a 10MBPS Ethernet. In the figure, the x-axis is the number of records and the y-axis is the number of seconds needed to insert these records into the database. From the figure, we see that the insertion performance of Scenario 3 is almost as good as that of Scenario 1, while Scenario 2 has the poorest insertion performance, which will take at least 20% longer to insert the same number of records. Meanwhile, indexes need quite an amount of disk space to store the index structures. It is necessary that tables and indexes are on different disk drives, otherwise, IO contentions will occur since both table data and index data will be read at the same time. In our situation, indexes are not good choices for aggregation queries.

From the figure, we see that the insertion rate for the three scenarios are almost constants. Table 5.6 shows the comparison of the three scenarios with insertion

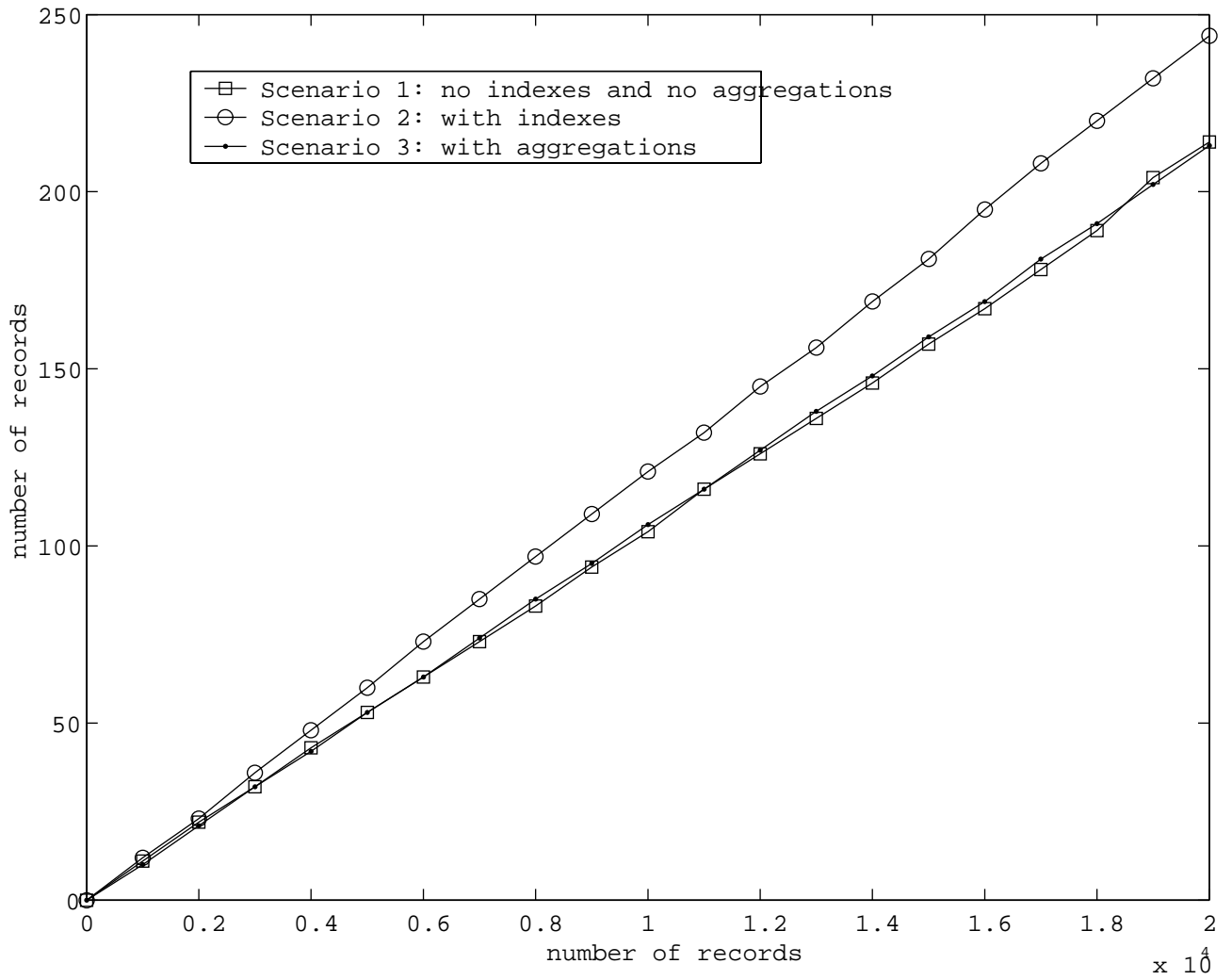


Figure 5.5. Insertion comparison of the three scenarios

performance and query performance when retrieving the same information from a table with 30 million rows. From the table, we see that aggregation tables only slow down the insertion performance a little, but the query performance gain is dramatic.

Table 5.6. Insertion and query performance comparison

Scenario	Insertion (seconds/row)	Query Time
1) No index, no aggregation	0.0106	> 1 hour
2) With index	0.0122	> 0.5 hour
3) With aggregations	0.0107	5 seconds

To get instant results of the aggregation queries from a large table with overloaded insertions, we need to generate temporary aggregation tables, since we cannot create indexes to speed up query due to the overhead of maintain the index structures. Often a database is normally queried after data is loaded or inserted using SQL*Loader or Import. In our situation, we need instant query of the database to view current statistics of the system, therefore, aggregation techniques are a good solution, also it is a simple solution. In next chapter, we will build our data warehouse both for query and data mining. Aggregation and transformation of data will be used to build the data warehouse. More query optimization techniques will be discussed.

CHAPTER 6

DATA WAREHOUSING FOR THE NOM SIMULATION

The NOM simulation generates a lot of data which is stored across the three database servers. The data is integrated, aggregated, transformed and then loaded into a data warehouse for ad hoc query and data mining.

As noted earlier, according to Inmon [22], a data warehouse is a “subject oriented, integrated, non-volatile and time-variant collection of data in support of management’s decisions.”. In the following sections, we show how to design, build and populate our data warehouse for both query and data mining.

6.1 Logical Design of The Data Warehouse

To make the data warehouse useful for the users, we need to load the raw data, possibly preprocessing it to resolve name and formatting inconsistencies (for example, the same molecule name with different structures). We then need to perform various aggregations to produce summary data in a form useful for the users. The design of the data warehouse will follow both of the following two approaches.

6.1.1 Detail And Summary Schema

Sometimes, it is possible that detailed data is not loaded to the data warehouse because of the large volume of detailed data produced by the simulation. But sometimes we may need to load detailed data in case we need to view specific detailed transactional data of a molecule, for example, the trajectory of a molecule

in the system. In the data warehouse built for this thesis, we loaded all the detailed data. Another decision will have to be made about retention, archival or purging of old detailed data. Because of limited storage, we need to purge some data as the disk space comes close to full.

In most cases, users will find the summarized data is much easier to navigate. In building a summarization, we can incorporate data from other related tables to avoid having to do joins from the summary. For example, the first sample query of last chapter will do joins against two tables NOM and REACTIONTYPE if we do not use the summary table REACTIONS_BY_TYPE.

A difficulty arises when providing data through summarizations. The difficulty is: what summarizations are anticipated? We don't know what questions our users will ask until they have worked with the data warehouse for a while. There is nothing to prevent adding more summary tables later after the design of the data warehouse. However, this approach will require that a sufficiently low level of detail to be available on which to base new summary tables. The new summary tables may need to be built based on detailed data which happens to be purged from the data warehouse.

Figure 6.1 shows one potential set of summary tables that might be derived from the detailed simulation data. The logical design of this type of schema can be implemented using Oracle. Oracle provides efficient parallel operations for populating the summary tables from the detailed data or from another lower level summarization. Building a summary from detailed tables can be efficiently performed using parallel CREATE TABLE AS SELECT.

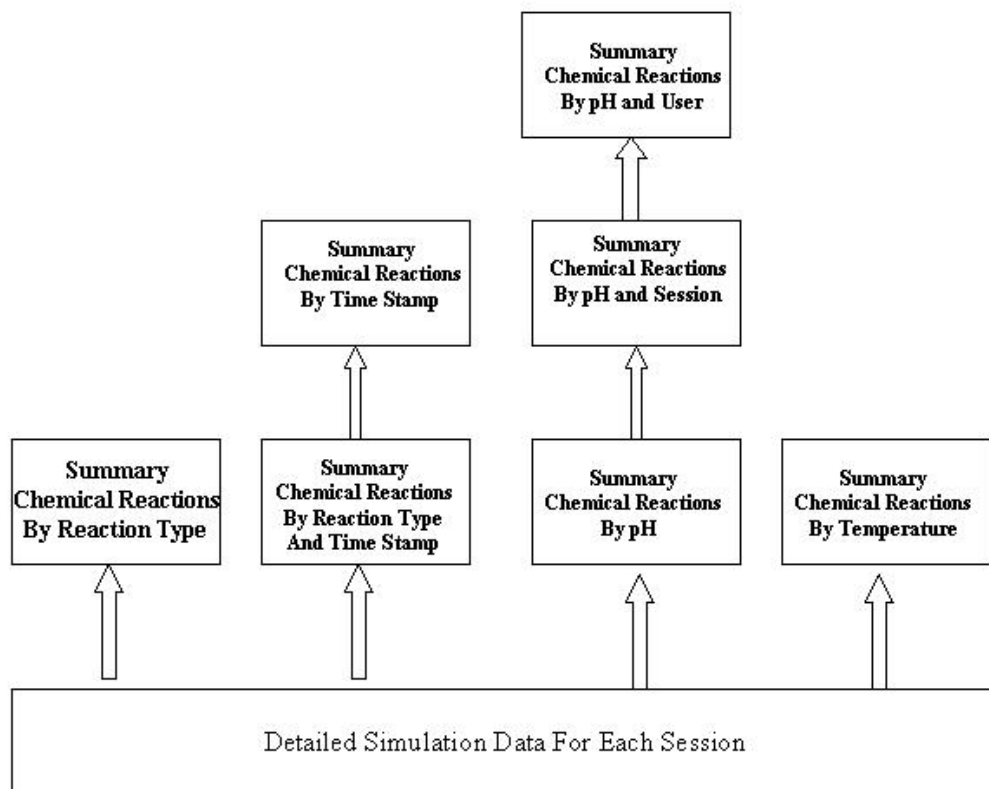


Figure 6.1. Examples of “Detail and Summary” Schema

6.1.2 Star Schemas

The primary references work on star schemas are Kimball's *The Data Warehouse Toolkit* [24] and *The Data Warehouse Life Cycle Toolkit* [25].

The relational star schema is derived from multidimensional database design. A star is designed as a central, large table of **facts**. When we think about a chemical reaction (fact), we might note that it occurred on a particular molecule or two, at a specific location, with a specific set of environments, in a specific session invoked by a specific user. Each of these ways of identifying the chemical reaction is a **dimension**.

Fact tables are commonly the largest collection of data in the data warehouse. To handle this volume of data, it is efficient to partition the fact table. Typically, the fact table is partitioned by user, or possibly by session. Each row in the fact tables has a column or a set of columns corresponding to the primary key columns of each of the dimension tables in the star. In addition to these foreign key columns, the fact table consists of one or more columns that describe the summaries, aggregations in a query.

Dimension tables are relatively small compared to the fact tables. Any descriptive attributes about the dimension will be stored in the dimension tables rather than the fact table.

The fact table has a foreign key relationship to each dimension table, i.e., each row inserted into the fact table will have values in various foreign key columns that correspond to primary key data in the dimension table. Rather than using B*Tree indexes on the foreign key columns to join dimension tables, Oracle's star transformation optimization will work on a separate bitmap index on each foreign key column. This allows the use of the very efficient bitmap merge operation to precisely determine the correct rows from the fact table. Further, the large fact

table may be partitioned on a frequently specified dimension. In many cases, a star schema query can be executed in comparable time to a corresponding query against a predefined summary table. The star schema design is much more flexible and saves a lot of space and advance processing effort. But it is not necessary to consider these two design alternatives as exclusive options. In our data warehouse, we design the star schema, but we also create summary tables based on common queries, such as statistics of chemical reactions.

More exotic variations of the star schema, known as **snowflake schemas**, extend the star schema model further. The snowflake branches correspond to different predefined levels of summarization in a traditional detail-summary warehouse schema. In this thesis, we will not use the snowflake schema.

Figure 6.2 shows a star schema of our data warehouse. Instead of predetermining which summary levels and GROUP BY columns are appropriate, the star schema allows users to select the dimensions that interest them and display whatever aggregate values are important.

6.2 Physical Design of The Data Warehouse

Physical data warehouse design requires translating the logical design of the detail-summary schema and star schema into actual database structures. Because we will use the Oracle database as the data warehouse, the physical design is specific to Oracle.

An Oracle database is organized into several tablespaces. Each tablespace occupies an amount of disk space through one or more operating system files. Each object such as table and index that requires storage space is assigned to a tablespace. The tablespace is the connection between the Oracle database structures and the physical disk files. The files associated with each tablespace are striped across multiple

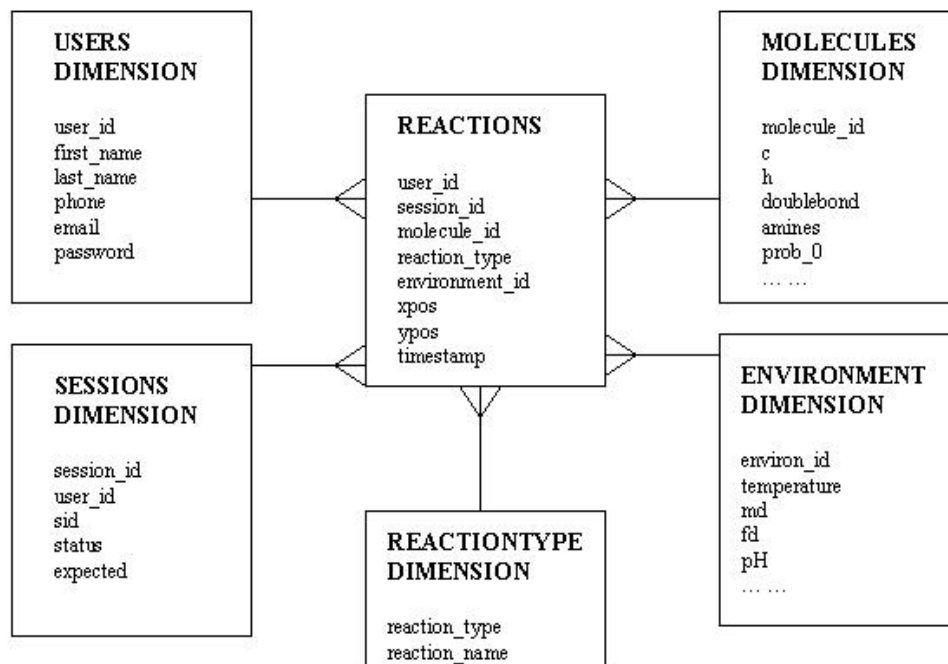


Figure 6.2. Star Schema

disk drives whenever possible.

To predict the space requirement for the data warehouse is very difficult. To predict the space requirement of tables, the method we use here is to load some data, then run the `ANALYZE` command to compute statistics that will provide a good estimate of the average length of a row and how many rows can be stored in each individual data block. Index storage is even harder to predict. To extrapolate the index's final size from sample data, the initially loaded sample must be uniformly distributed across the full range of data values.

Data warehouse tables have very different growth patterns. In many situations, it is critical to ensure that there is sufficient space available to accommodate the data to be loaded. Because the load jobs are scheduled nightly after new simulation data is produced, it is essential that the jobs can be completed without manual intervention in order to add datafiles to a tablespace. In Oracle, this can be implemented using the `AUTOEXTEND` clause as part of the file specification in the `CREATE TABLESPACE` and `ALTER TABLESPACE ... ADD DATAFILE` commands. Allowing data files to automatically grow provides some insurance against the midnight failure of a load or index job. In addition to the growth of the data warehouse, we also need to consider removing older data from the data warehouse.

The large `REACTIONS` table is partitioned for better query performance and easy administrative management. The table is partitioned based on `SESSION_ID`.

In trying to find the data of interest to a particular query, Oracle always has the option of reading all data in the table. For small tables, this is often the preferred way. In many situations, an index is needed to allow more rapid query of a single row or a small subset of the total table. Which columns need to be indexed is determined by knowing the data and anticipating the types of queries that will be issued by the users. The `WHERE` clause will determine which indexes can be used

by the query in an SQL statement. In section 6.5, we will discuss query optimization techniques specific to data warehouse.

Database constraints and triggers, commonly used in OLTP database design, have very limited use in the data warehouse. Performance considerations for the data warehouse are very different from those in the OLTP systems. Transactional processing generally focuses on the performance of a single update. For data warehouse, the data is initially entered into the database through large batch file loads as opposed to single row inserts, it is easier to examine the data integrity of the entire file as one batch step prior to loading.

6.3 Build The Data Warehouse

After planning and designing the data warehouse, the next step is to build the data warehouse. First of all, we need to create an Oracle database for the data warehouse. A number of parameters must be manipulated since they will have a direct effect on performance. These parameters must be tuned so they provide the best performance for our specific applications, for example, queries and data mining. We have to do some analysis and planning before we create the database. It is useful to have an estimate of the size of objects such as tables and indexes in the data warehouse so that we can ensure that our hardware architecture is adequate to support it.

We used the following methods to give us the rough estimate. First we determined how much space we will have in Oracle data blocks for our data. For our data warehouse, each Oracle data block has the size of 8KB. Oracle requires a small portion of each data block for some internal information, which is the block header. We can assume this portion is 200 bytes. Second we need to know how much space will be reserved for future updates and how much will be available

for initial inserts. This parameter is PCTFREE as introduced before. We can assume PCTFREE is the default value of 10. So the amount of space in each data block available for our data is equal to $\text{DATA BLOCK SIZE} - \text{BLOCK HEADER} - \text{PCTFREE}/100 * \text{DATA BLOCK SIZE}$. In our example, which is $8192 - 200 - 819 = 7073$ bytes. Next we determine how much space an average row in our table requires. Most Oracle data types are stored in variable length format so the size will be determined by the actual data loaded. To get a relatively precise estimate of the average length of a particular column's data, we load a sample of the data and then query the `AVG(VSIZE(column_name))` for each column. We have to add 1 byte for column overhead if the length of the column is under 255 bytes and 3 bytes if the column length is over 255 bytes. We also need to add 4 bytes for row overhead, which are used by ROWID which denote the address of the row in disk. The length of a row is the sum of the column lengths and column overhead plus the row overhead. This calculation is not precise because Oracle stores different types of data differently, but it provides a generally conservative estimate. We repeat this process for each fact table created in the data warehouse.

Another big reason why we want to estimate the space utilization for each data block is for query performance. Without careful estimation of the space utilization, it is often possible that one row of a table will span two data blocks. This is called chaining. To retrieve this row, 2 I/Os will be needed instead of one I/O, thus query performance will suffer.

Oracle stores database objects such as tables and indexes in tablespaces. We specify `AUTOEXTEND` in the `CREATE TABLESPACE` command, which helps us avoid loading errors that may occur during batch jobs. Once we have some idea of storage requirements, we can ensure that we have a sufficient number of disk drives available to support our data warehouse. We create a tablespace named `NOM`.

When creating this tablespace we try to spread tables across disk drives whenever possible. This greatly improve the performance if we use parallel query.

After creating database and tablespaces, we want to create our large fact table, for example, REACTIONS in the tablespace NOM so that we can take the advantage of parallel query when doing full table scans. Because our fact table REACTIONS is never updated, we don't have to reserve space for future updates. We will use all the available space in each data block. We use the following command to create the fact table.

```
CREATE TABLE REACTIONS
(USER_ID INTEGER,
SESSION_ID INTEGER,
MOLECULE_ID INTEGER,
REACTION_TYPE INTEGER,
ENVIRONMENT_ID INTEGER,
XPOS INTEGER,
YPOS INTEGER,
TIMESTAMP INTEGER)
TABLESPACE NOM
STORAGE (INITIAL 512M NEXT 512M MINEXTENTS 4 PCTINCREASE 0)
NOLOGGING PCTFREE 0
PARTITION BY RANGE (SESSION_ID)
(PARTITION SESS_100 VALUES LESS THAN ('100'),
PARTITION SESS_200 VALUES LESS THAN ('200'),
PARTITION SESS_300 VALUES LESS THAN ('300'),
PARTITION SESS_REST VALUES LESS THAN ('1000'),
);
```

This command results in the REACTIONS table receiving 4 extents at the time of table creation. The PCTINCREASE 0 clause tells Oracle that every data extent allocated to the REACTIONS table should be 512M. The NOLOGGING attribute informs Oracle to minimize the amount of redo information that is logged for this table. Finally, PCTFREE 0 makes the best possible usage of disk space and minimize the number of data blocks required. Rather than have to create and manage individual tables and then use a view to make them appear to be a single object,

Oracle allows a single table to be stored in multiple partitions. The order in which partitions are created is important. Partitions with lower values for the partition key must be declared before those for higher values.

After creating tables in the schema, we need to create indexes to speed up query and constraints to ensure data integrity. These work are not trivial at all. We need to know the data and potential queries before creating indexes. because the task of creating indexes and constraints it pretty tedious, we would like to skip it in the thesis. We may still mention some of the indexes and constraints when we discuss query optimization techniques in later sections.

6.4 Populate The Data Warehouse

There are various ways to load data into the data warehouse. Possible methods include using SQL*Loader, Export/Import, SQL*Plus copy command, the CREATE TABLE ... AS SELECT command, or other custom-written load programs such as JDBC. Our data warehouse in nature is different from commercial data warehouses, in which there is needed ways to deal with dirty data, missing data, etc. The data in our data warehouse is generated by the core simulation. We don't have to worry about dirty data or missing data. All we need to do is to extract, transform and load the data from the simulation databases to the data warehouse.

From all the methods of loading data, we use the CREATE TABLE ... AS SELECT command to copy data from the simulation databases. The fact tables and dimension tables are populated using the same method. Then we create some detail-summary schemas for some popular queries such as the two queries shows in last chapter.

After data is loaded it must be verified, indexed to make it available for query and data mining.

6.5 Query Optimization

One of the characteristics of data warehouse SQL queries is the presence of many tables in the SQL select statements. In standard star schema design, a central fact table is joined with numerous dimension tables.

We use the following techniques to improve the speed of data warehouse SQL statements:

- The *ordered* hint: The ordered hint specifies the optimal way to join the tables together. This bypasses the expensive parse phase of data warehouse SQL and ensures that the tables are always joined in the same order. Oracle must spend a great deal of time parsing n-way table joins (n tables are in the SQL statements) to determine the optimal order to join the tables. Large n-way table joins with seven or more tables can often take more than an hour to parse the SQL statement. This is because Oracle must evaluate all possible join orders. For example, with 9 tables, Oracle must evaluate 8!, or 40,320 possible join combinations. The ordered hint can be used to reduce the SQL parse time and Oracle joins the tables in the same order in which they appear in the *from* clause. Hence, the first table after the *from* clause becomes the driving table, and the driving table should be the table that returns the smallest number of rows. The ordered hint is commonly used in conjunction with other hints to ensure that multiple tables are joined in their proper order.
- The *star* hint: For queries of a fact table or a dimension table, the star hint can greatly improve the join speed of data warehouse queries. A permutation of the hash join, the star join technique builds a hash index on the fact table indexes. Bitmap indexes are required for all join columns on the fact table, and Oracle will initially use these bitmap indexes as a path to the fact table.

The SQL optimizer will then rewrite the original query, replacing the equi-join criteria with sub-queries using the IN clause. These sub-queries are used as sources of keys to drive the bitmap index accesses, using bitmap key iteration to access the dimension tables. Once the resulting bitmap-ROWID lists are retrieved, Oracle will use a hash join to merge the result sets. As illustrated in Figure 6.3, the performance improvement of the approach is a result of reducing the physical I/O. The indexes are read to gather the virtual table in memory, and the fact table will not be accessed until the virtual table has everything it requires to go directly to the requested rows via the composite index on the fact table.

- Partitioning: The use of partitioned tables and indexes can improve the speed of SQL statements. In a partition-wise join, join is divided into smaller joins that occur between each of the partitions on which the table reside, completing the overall join in less time. If you know the partition that contains your data, you can explicitly reference it in your SQL query. Since the tables are partitioned against the SESSION_ID, all data for each session is in only one partition, query against information for one session can be performed on one partition of the large table.

6.6 Conclusion

In this chapter, we described the design and construction of a data warehouse, which employs the star-schema and detail-summary schema data warehousing technologies. We also discussed query optimization methods in data warehouse environment. These techniques include ordered hint, star hint and partitioning. In the next chapter, data mining algorithms will be applied to the data warehouse.

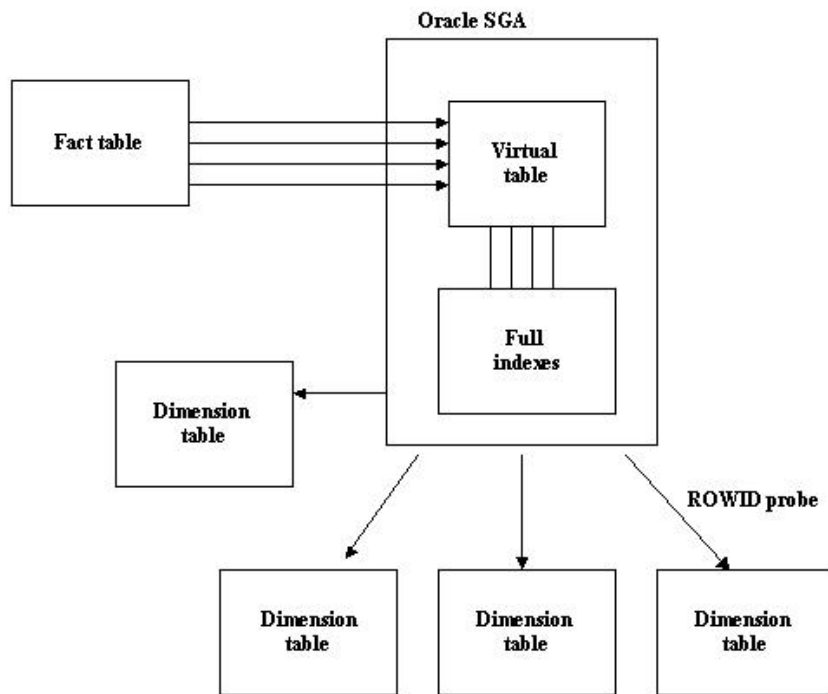


Figure 6.3. Star Join

CHAPTER 7

DATA MINING FOR THE NOM SIMULATION

Data mining refers to extracting or mining knowledge from a large amount of data. There are many other terms carrying a similar or slightly different meaning to data mining, such as knowledge discovery in databases (KDD), knowledge extraction, data/pattern analysis, data archaeology, data dredging and information retrieval [20]. In the previous chapter, we described the design and construction of our data warehouse. In this chapter, we apply data mining techniques to the data warehouse.

7.1 Introduction to Oracle Data Mining

As illustrated in Figure 7.1, data mining is a step in the process of knowledge discovery. In this thesis, we use data mining to find interesting patterns, then evaluate apply these patterns to new data.

Oracle9i Data Mining (ODM) has two components: Data Mining API and Data Mining Server (DMS). The data mining API is the component that allows users to write Java programs to mine data. The ODM API provides an early look at concepts and approaches being proposed for the emerging standard Java Data Mining (JDM). JDM follows Sun's Java Community Process as a Java Specification Request (JSR-73). JDM is based on several evolving data mining standards, including the Object Management Group's Common Warehouse Metadata (CWM),

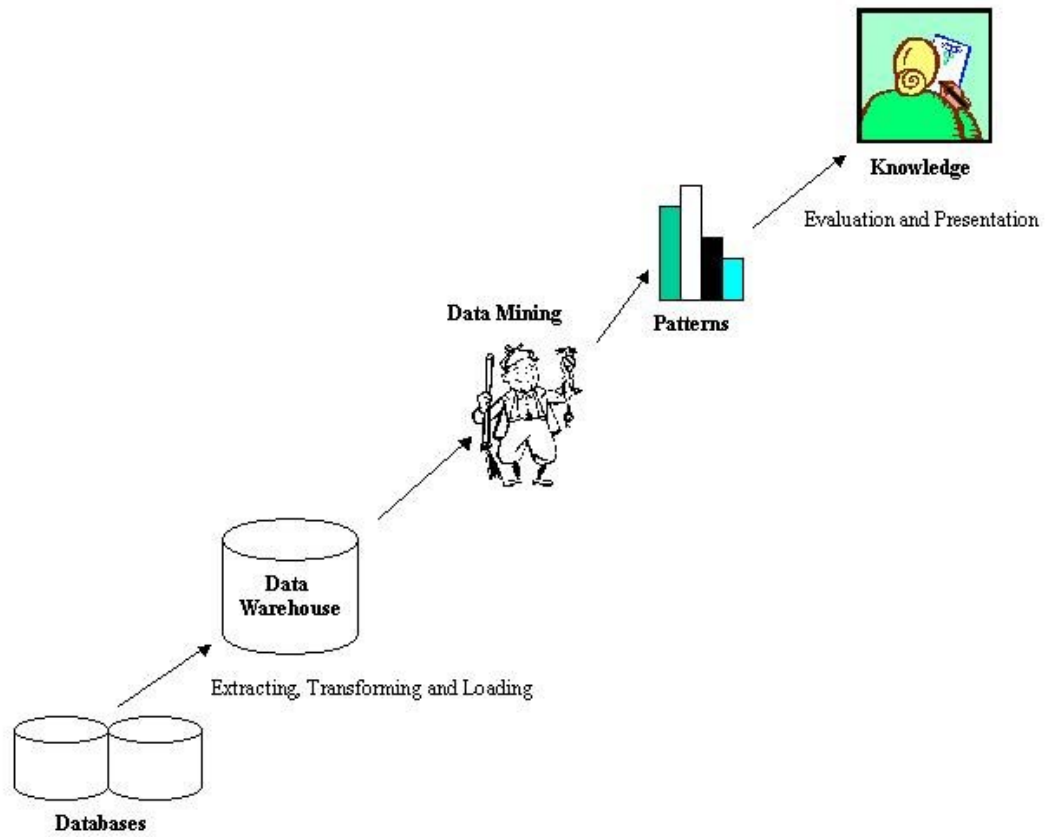


Figure 7.1. Data mining as step of knowledge discovery

the Data Mining Group's Predictive Model Markup Language (PMML), and the International Standards Organization's SQL/MM for Data Mining. The data mining server (DMS) is the server-side in-database component that performs the data mining operations within the 9i database, and thus benefits from its availability and scalability. The DMS also provides a metadata repository consisting of mining input objects and result objects, along with the namespaces within which these objects are stored and retrieved [5].

Data mining models are based on two kinds of learning: supervised and unsupervised. Supervised learning functions are typically used to classify and predict a value, unsupervised to typically used to find intrinsic structures or relationships in data. Oracle supports the following data mining functions:

- Classification (supervised)
- Clustering (unsupervised)
- Association Rules (unsupervised)
- Attribute Importance (supervised)

In this thesis, we only apply clustering to the data warehouse. Clustering is useful for exploring data. It is particularly useful when there are many attributes and no natural groupings. Clustering analysis identifies clusters embedded in the data. A cluster is a collection of data objects that are similar in some sense to one another. A good clustering method produces high-quality clusters to ensure that the inter-cluster similarity is low and the intra-cluster similarity is high; in other words, members of a cluster are more like each other than they are like members of a different cluster.

The process of data mining consists of the following steps: model building, model testing, computing lift (not a necessary step) and model applying.

As illustrated in Figure 7.2, the left side of the diagram shows the steps to perform before data can be supplied to the core algorithm. The pre-processing result is the build data table. The build data table and the mining function settings are shown as boxes with arrows pointing to the algorithm, indicating they are supplied as input for the algorithm. In the build data table processing step, the user may bin the data manually and produce bin boundary tables. ODM can perform the binning automatically. The algorithm then builds the model, using the pre-processed data, the mining function settings, and internal bin boundary tables and the core algorithm.

Classification models can be tested on new data with known target values to get an estimate of the accuracy of the models. A classification model contains a confusion matrix that allows a user to understand the type and number of classification errors made by the model. Applying a clustering model to new data produces, for each case, a predicted cluster identifier and the probability that the case belongs to that cluster. The test data must be in the same format and state of preprocessing as the data used to build the model.

In business world, the purpose of a simple targeting model is to identify a subgroup (target) from a larger population. The target members selected are those likely to respond positively to a marketing offer. A model is doing a good job if the response within the target is much better than average for the population as a whole. Lift is simply the ratio of these values: target response divided by average response. Let's assume that we want to predict whether a molecule will experience a chemical reaction. We know that only 1 percent of molecules will experience chemical reactions in average. Suppose that there are 1,000,000 molecules, then there are 10,000 molecules will do chemical reactions. If we can find a subset of molecules, say 100,000 molecules, but of which 2,000 molecules will experience chemical reactions,

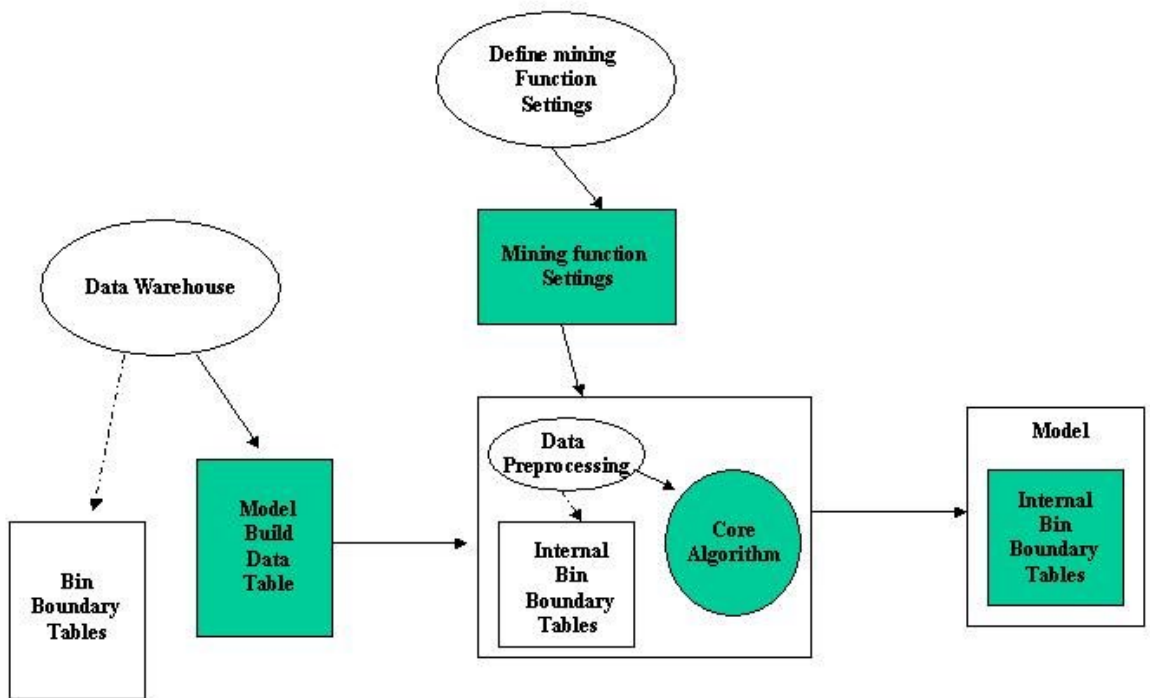


Figure 7.2. Model-Build Process

then there are 2 percent of molecules will experience chemical reactions among the subset, thus a lift of 2.

Figure 7.3 depicts the model-apply process. The left side of the diagram shows the steps to be performed before data can be supplied to the algorithm which scores the data. The “apply output” specification is a set of objects that determine the content of the result table. The algorithm then applies the model, using the pre-processed data, the “apply output” settings, the internal bin boundary tables, and the core algorithm. The result is a table with scores for each data element in the apply data table.

In next sections, we apply clustering algorithms to the data warehouse.

7.2 Clustering

In this section, an object refers to a row in a table, which stores the data to be clustered. There exists a large number of clustering algorithms in the literature. The choice of clustering algorithms depends on the type of data available and on the particular purpose and application [20]. A recent survey on clustering can be found in Murty, John and Flynn [7]. In general, major clustering methods can be classified into the following categories.

- **Partitioning:** Given n objects and a number k , where $k < n$, a partitioning method constructs k partitions of the data, where each partition represents a cluster. The partitioning methods classifies the n objects into k clusters, with the following requirements: (1) each cluster must contain at least one object, and (2) each object must belong to exactly one cluster. The goal of a partitioning method is to ensure that objects in the same cluster are close to each other, while objects of different clusters are far apart. To achieve global optimality, this method would require exhaustive enumeration of all possible

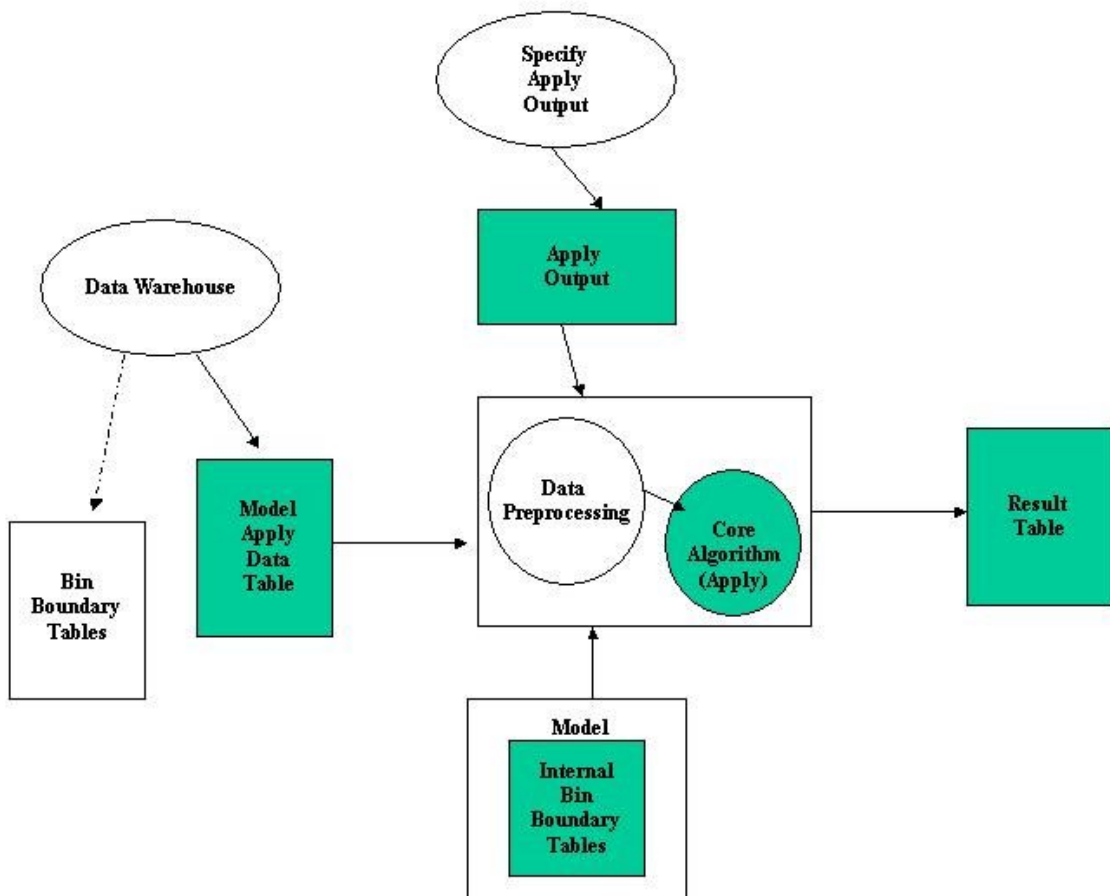


Figure 7.3. Model-Apply Process

partitions, which is computationally impossible. Most applications adopt one of the two following heuristic methods: (1) the **k-means** algorithm, where each cluster is represented by the mean value of the objects in the cluster [31], and (2) the **k-medoids** algorithm, where each cluster is represented by one of the objects located near the center of the cluster [11]. These heuristic methods work well for finding spherical shaped clusters in small or medium sized databases.

- **Hierarchical:** The hierarchical methods create a hierarchical decomposition of a given set of objects. Such a method can be classified as being either **agglomerative** or **divisive**, based on how the hierarchical decomposition is formed [11]. The agglomerative approach, which is also called the bottom-up approach, starts with each object forming a separate cluster. It successively merges the clusters close to one another, until all of the clusters are merged into one cluster, or until a termination condition holds. The termination condition could be the number of iterations or an error tolerant value. The divisive approach, also called the top-down approach, starts with one cluster containing all the objects. In each successive iteration, a cluster is split into smaller clusters, until eventually each object forms a cluster, or until a termination condition holds. Hierarchical methods suffer from the fact that no undo is possible when one iteration (either merge or split) is done. So a major problem of this method is that they cannot correct erroneous decisions. There are two approaches to improving the quality of hierarchical clustering: (1) perform careful analysis of object linkages at each hierarchical partitioning, such as in CURE [38] and Chameleon [16], or (2) integrate hierarchical agglomeration and iterative relocation by first using a hierarchical agglomerative algorithm and then refining the result using iterative relocation, as in BIRCH [41].

- **Density-based:** The idea of the density-based method is to continue growing the given cluster as long as the density (number of objects) in the neighborhood exceeds some threshold. That is, for each object within a given cluster, its neighborhood (an open ball centered at this object with a certain radius) has to contain at least a minimum number of objects. Such a method can be used to filter out noise and discover clusters of any shape. DBSCAN is a typical density-based method that grows clusters according to a density threshold [30]. OPTICS is a density-based method that computes an augmented clustering ordering for automatic and interactive cluster analysis [29].
- **Grid-based:** Grid-based methods quantize the object space into a finite number of cells that form a grid structure. The dimension of the space is equal to the number of attributes for the objects. All of the clustering operations are performed on the grid structure. The main advantage of this approach is its fast processing time, which only depends on the number of cells, independent of the number of objects. STING is a typical example of such method, which collects statistical information in grid cells [42]. CLIQUE and WaveCluster are two clustering methods that are both grid-based and density-based. WaveCluster is a multiresolution clustering approach that transforms the original feature space by wavelet transform [17]. CLIQUE is an integrated density-based and grid-based clustering method for clustering high-dimensional data [37]
- **Model-based:** Model-based methods attempt to optimize the fit between the given data and some mathematical model. Such methods are often based on the assumption that the data are generated by a mixture of underlying probability distributions. Model-based clustering methods follow two major

approaches: a statistical approach or a neural network approach [40].

To apply the clustering algorithms to our data warehouse, we create a table called POINTS which contains the data of two attributes from the REACTIONS table, xPOS and yPOS, which denote the position of a molecule in the simulation system. We standardize the points by re-scaling the points into the $[0, 1] \times [0, 1]$ square. Then we apply the Enhanced k-means algorithm and the O-Cluster algorithm to the POINTS table. The enhanced k-means algorithm is a mixture of the partitioning method and hierarchical method. The O-Cluster algorithm is a mixture of the grid-based and hierarchical method.

7.2.1 Enhanced k-Means Algorithm

The k-means takes the input parameter k , and partitions a set of n objects into k clusters so that the resulting intracluster similarity is high but the intercluster similarity is low. Cluster similarity is measured in regard to the *mean* value of the objects in a cluster, which can be viewed as the cluster’s *center of gravity*. The similarity of two objects are based on their distance. In general, it is a nonnegative number that is close to 0 when two objects are highly similar or near each other, and becomes larger the more they differ.

The k-means algorithm proceeds as follows. First it randomly selects k of the objects, each of which initially represents a cluster mean or cluster. For each of the remaining objects, it is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster mean. It then computes the new mean for each cluster. This process iterates until the criterion function converges. Typically, the **squared-error criterion** is used, defined as

$$E = \sum_{i=1}^k \sum_{p \in C_i} |p - m_i|^2, \quad (7.1)$$

where E is the sum of square-error for all objects in the table, p is the point in space representing a given object, and m_i is the mean of cluster C_i . This criterion tries to make the resulting k clusters as compact and as separate as possible.

A hierarchical clustering method works by grouping data objects into a tree of clusters. Oracle implements a hierarchical version of the k-means algorithm and use the top-down approach. The cluster with largest distortion (sum of distances to the cluster's centroid) is split to increase the number of clusters until the desired number of clusters is reached.

Because the k-means algorithm requires multiple passes through the data, it can be impractical for large data tables that don't fit in memory. In this case multiple expensive database scans would be required. ODM's enhanced k-means requires at most one database scan. For data tables that don't fit in memory, the enhanced k-means algorithm employs a smart summarization approach that creates a summary of the data table that can be stored in memory. This approach allows the enhanced k-means algorithm to handle data tables of any size. The summarization scheme can be seen as an adaptive sampling approach that generates more points for regions of the input space where it is harder to separate clusters.

7.2.2 O-Cluster Algorithm

The O-Cluster algorithm is used to cluster high-dimensional data in large databases. It works based on the following:

- Given a large set of multidimensional data points, the data space is usually not uniformly occupied by the data points. The algorithm identifies the sparse and the crowded areas in the space, thereby discovering the overall distribution patterns of the data set.
- A unit (which is a rectangular region in the object space) is dense if the fraction

of total data points contained in it exceeds an input model parameter, called sensitivity. A cluster is defined as a maximal set of connected dense units.

In the first step, the algorithm partitions the n -dimensional data space into non-overlapping rectangular units, where n is the number of attributes for the objects. It then identifies the dense units among them. This is done in a dimension-by-dimension phase, based on the following: If a k -dimensional unit is dense, then so are its projections in $(k-1)$ -dimensional subspaces. In the second step, the algorithm generates a minimal description for each cluster as follows. For each cluster, it determines the maximal region that covers the cluster of connected dense units. It then determines a minimal cover for each cluster.

7.2.3 Description of The Clustering Programs

This section shows how we apply the clustering algorithms to the model build data to build a clustering model and then use the model to score new data. Because the two algorithms, Enhanced k-means and O-Cluster, share some common steps, we specify them in the parameter files, instead of writing separate files. The program `ClusteringBuild.java` builds a clustering model, while the program `ClusteringApply.java` applies the model to score new data. We discuss the two programs in detail. We start with the model build program.

The program `ClusteringBuild` accepts two parameter files, `Global.property` which specifies the data mining server connection information such as data mining server URL, username, password, etc, and `ClusteringBuild.property` which specifies the inputs for the algorithms. More precisely, `ClusteringBuild.property` provides the inputs for a `PhysicalDataSpecification` instance, a `MiningFunctionSettings` instance, the model name, etc.

The following steps illustrate the details of building a clustering model:

1. Connect to the data mining server.

We first need to create an instance of `DataMiningServer`. This instance is used as a proxy to create connection to a Data Mining Server. It also maintains the connection. The data mining server performs the data mining operations. It also provides a metadata repository storing mining input objects and resulting objects. The following code shows this step:

```
//Create an instance of the DataMiningServer
//where DB_URL is of the form:
//jdbc:oracle:thin:@hostname:port:sid
//DB_URL, user_name and password are
//specified in Global.property
oracle.dmt.odm.DataMiningServer dms=
    new DataMiningServer('DB_URL',
        'user_name', 'password');

//get the connection
oracle.dmt.odm.Connection dmsConnection=dms.login();
```

2. Create a `PhysicalDataSpecification` object for the model build data.

Before the data mining server can use the data to build a model, it must know where the data is located and how the data is organized. This is done by creating a `PhysicalDataSpecification` instance where you indicate the location of the data and whether the data is in transactional format or nontransactional format. Before creating a `PhysicalDataSpecification` instance, the location of the table must be specified, which is done by creating a `LocationAccessData` instance, as illustrated in the following code:

```
//Create a LocationAccessData instance using the table_name
//POINTS and schema_name nom_mine
oracle.dmt.odm.LocationAccessData lad=
    new LocationAccessData('POINTS', 'nom_mine');
```

If the model build data is in nontransactional format, all the information

needed to create a PhysicalDataSpecification instance is in the LocationAccessData instance, as illustrated here:

```
//Create a PhysicalDataSpecification instance
//for a NonTransactionalDataSpecification instance
oracle.dmt.odm.data.PhysicalDataSpecification my_pds=
    new NonTransactionalDataSpecification(lad);
```

If the data is in transactional format, the role that the table columns play must be specified, as illustrated here:

```
//Create a PhysicalDataSpecification instance
//for a TransactionalDataSpecification instance
oracle.dmt.odm.data.PhysicalDataSpecification my_pds=
    new TransactionalDataSpecification(
        'CASE_ID', //column name for sequence id
        'ATTRIBUTES', //column name for attribute name
        'VALUES', //column name for value
        lad);
```

3. Create a MiningFunctionSettings object which specifies the algorithm settings.

To build a clustering model, we can use either the Enhanced k-means algorithm or the O-Cluster algorithm. For the k-means algorithm, we need to specify the number of iterations, or the error tolerance, or both. The algorithm terminates when either the number of iterations or the error tolerance is reached. For the O-Cluster algorithm, we need to specify sensitivity, which is used to compute a baseline density. Only regions exceeding the baseline density are considered as clusters. The following code shows how to create a MiningFunctionSettings object:

```
if ( my_AlgorithmType == 1 ) //k-means algorithm
{
    if(my_StoppingCriterion.compareToIgnoreCase(
        "errorAndIterations") == 0 )
        algorithmSetting = new KMeansAlgorithmSettings(
```

```

        my_Iterations, //the maximum number of iterations
        my_Error, //the error tolerant
        DistanceFunction.euclidean //use Euclidean distance
    );
else if(my_StoppingCriterion.compareToIgnoreCase(
    "error") == 0 )
    algorithmSetting = new KMeansAlgorithmSettings(
        my_Error,
        DistanceFunction.euclidean
    );
else if(my_StoppingCriterion.compareToIgnoreCase(
    "iterations") == 0 )
    algorithmSetting = new KMeansAlgorithmSettings(
        my_Iterations,
        DistanceFunction.euclidean
    );
}
else if ( my_AlgorithmType == 2 ) //0-Cluster algorithm
    algorithmSetting =
        new OClusterAlgorithmSettings(my_Sensitivity);

ClusteringFunctionSettings c1MFS =
    ClusteringFunctionSettings.create (
        dmsConnection, // the connection to the DMS
        algorithmSetting, // algorithm settings
        my_PhysicalDataSpecification,
        //the physical data specification
        ( my_AutoBinning == true ?
        //whether using manual discretization
        DataPreparationStatus.unprepared :
        DataPreparationStatus.discretized),
        my_Clusters,
        //the maximum number of clusters for k-means
        null, //array of categorical mining attributes
        null, //array of numerical mining attributes
        null //array of integral mining attributes
    );

c1MFS.validate(); // validate before store
c1MFS.store(dmsConnection, my_MiningSettingsName);

```

Because the MiningFunctionSettings objects are very complex objects, it is good practice to validate before performing the model build task. If the objects are validated, it should be stored in the data mining server for later use, for example, when we apply the model to score new data later.

4. Build the model.

Now all information needed to build a clustering model is captured in an instance of PhysicalDataSpecification and MiningFunctionSettings, the last step is to build the model. To build the model, we create an instance of MiningTask. A mining task can be stored in the DMS using the store method and executed at any time. Once the task is executing, query the current status information of a task by calling the getCurrentStatus method. This call returns a MiningTaskStatus object, which provides more details about the state. The following code shows this:

```
// Create mining task
MiningBuildTask miningbuildTast =
    new MiningBuildTask(
        my_PhysicalDataSpecification,
        //the physical data specification
        my_MiningSettingsName,
        //the name of mining settings, clMFS
        my_ModelName //name of the model
    );

// Store mining task
miningbuildTast.store(
    dmsConnection,
    my_BuildTaskName //name of the model build task
);

// Execute mining task
miningbuildTast.execute(dmsConnection);

// Wait for completion
MiningTaskStatus miningTaskStatus =
```

```

        miningbuildTast.waitForCompletion(dataMiningServerHandle);

// Check status
MiningTaskState miningTaskState =
    miningTaskStatus.getTaskState();

if(miningTaskState.isEqual( MiningTaskState.success ))
{
    System.out.println("Build Task suceessfully completed");
    System.out.println("Built (and stored) Clustering Model.
                        Name: " + my_ModelName);
}else if(miningTaskState.isEqual( MiningTaskState.error))
{
    System.out.println( "Build failed due to: " +
        miningTaskStatus.getStateDescription() );
} else
{
    System.out.println( "Unexpected error occured:  " +
        miningTaskStatus.getStateDescription() );
}

```

After building a model, you can obtain the information of the model using the ODM APIs. The program Clustering.java uses the ODM API to obtain information about the model we just built. It lists the information about the number of clusters, number of points in each cluster, the boundary of each cluster, the hierarchical structure of these clusters, etc.

Next, we discuss the program ClusteringApply.java, which applies the model built in ClusteringBuild.java to score new data. The new data is stored in a table POINTS_APPLY, which contains 4,000 records to be scored. The following steps show how ODM scores data using a model.

1. Connect to the data mining server.

This step is the same as the first step in building a model.

2. Create a PhysicalDataSpecification object for input data which is the data to

be scored.

This step is the same as the second step in building a model.

3. Create a `LocationAccessData` object for output data, which is a table to store the scoring results for the input data.

The result table is called `POINTS_CL_APPLY_RESULT`. The following code specifies writing to the output table.

```
//a LocationAccessData object for output table
//to store the model apply results
oracle.dmt.odm.LocationAccessData ladOut =
    new LocationAccessData(
        'points_cl_apply_result', 'nom_mine');
```

4. Create a `MiningApplyOutput` object for output data.

A `MiningApplyOutput` object captures the format of the scoring output. An instance of `MiningApplyOutput` specifies the data (columns) to be included in the apply output table that is created as the result of an apply operation. The columns in the apply output table are described by a combination of `ApplyContentItem` objects. These columns can be either from the input table or generated by the scoring task (for example, prediction and probability). In the `MiningApplyOutput` object, for each record in the table to be scored, there is a number of probabilities assigned to it which denote the probabilities for all the clusters. Among these probabilities, we picked the highest one, which means that for each record in the input table, it is assigned to the cluster with the highest probability. Since the `MiningApplyOutput` object is quite complex, we always validate it before performing the scoring task.

5. Score the data.

After gathering all the information for scoring the input table, which is con-

tained in the PhysicalDataSpecification object, the MiningApplyOutput object, the last step is to score the data. We create an MiningApplyTask object which can be stored in the data mining server. The current status information of a task can be queried by calling the getCurrentStatus(dmsConnection, taskName) method. The following code shows this:

```

MiningApplyTask task = new MiningApplyTask(
    // Physical Data Spec for input data
    my_InputPhysicalDataSpecification,
    my_ModelName, // Name of the model to be applied
    my_MiningApplyOutput, // MiningApplyOutput
    ladOutput, // Output table location
    // Result name in odm_apply_result table on the DMS
    my_ApplyOutputResultResultName);
task.store(dmsConnection, my_ApplyTaskName);
task.execute(dmsConnection);

MiningTaskStatus status =
task.waitForCompletion(dmsConnection);
System.out.println("    Apply Task Status: " +
                    status.getTaskState().getEnum());

```

Figure 7.4 shows 20 records in the POINTS_CL_APPLY_RESULT table, which has four columns, namely, MYPREDICTION, MYPROBABILITY, XPOS, YPOS, where XPOS, YPOS are the two columns from the POINTS_APPLY table, MYPREDICTION and MYPROBABILITY denote that the record should be assigned to the predicted cluster with the probability MYPROBABILITY.

7.2.4 Comparison of Enhanced k-means and O-Cluster

The clustering methodologies for enhanced k-means and O-Cluster algorithms are distance-based and grid-based respectively. The number of clusters created by

Oracle SQL*Plus

File Edit Search Options Help

SQL> /

MYPREDICTOR	MYPROBABILITY	XPOS	YPOS
8	1	.2692298	.31838146
7	.999999993	.60283025	.67409576
8	1	.24353355	.32321641
6	.998742421	.55165184	.3749809
8	1	.32056723	.43911267
7	.995819653	.63817123	.53058183
7	1	.54741483	.65221134
7	.999944358	.55317962	.515365
8	1	.30705166	.30945664
9	.99999988	.25124943	.68733497
6	.999438851	.57576427	.36904461

MYPREDICTOR	MYPROBABILITY	XPOS	YPOS
8	.999999995	.35594739	.43826327
7	.995819653	.63367435	.54459678
7	.999270623	.60786809	.56925711
6	.995990915	.51912221	.398702
7	.999999995	.48946769	.7178514
6	.999735914	.53355317	.34600943
7	1	.52713097	.75590523
5	1	.8059109	.54571319
8	1	.26032539	.33423874

20 rows selected.

SQL> |

Figure 7.4. Model-Apply Results Table

the model building process is user-specified for enhanced k-means algorithm. The number of clusters is automatically discovered by the O-Cluster algorithm. Both algorithms use the hierarchical clustering method. They assign scoring data to clusters probabilistically. Table 7.1 compares the time needed to build a model and the shapes of clusters. The model build table POINTS contains around 4 million records. These records are taken from the REACTIONS table, so the records are not necessary generated by one simulation. From the table, we see that O-Cluster has the advantage of less model build time. The number of clusters is user-specified for the enhanced k-means algorithm, while O-Cluster will discover the number of clusters automatically.

Table 7.1. Comparison of Enhanced k-means and O-Cluster)

Algorithm	Model Build Time	Cluster Shape	Number of Clusters
Enhanced k-means	34 minutes	Spherical	8
O-Cluster	14 minutes	Rectangular	15

7.3 Conclusion

In this chapter, we discussed the data mining programming using ODM. Especially, two clustering algorithms Enhanced k-means and O-Cluster are applied to the NOM simulation data and some interesting clusters are discovered and applied the models to new data. One way to explain these clusters is that “hot spots” exist in the simulated world, the movements and reactions of these molecules enable the molecules to form clusters. This discovery could be useful to determine the physical distributions of molecules in the evolution of soil, etc. Another interesting problem is to predict whether certain molecules will be adsorbed in a certain period of time. This is a typical application of classification. We plan to apply some classification methods such as Naive Bayes algorithm or decision tree algorithm to build a classifi-

cation model, which then can be used to predict whether certain types of molecules will be adsorbed or not in a certain time period. Other data mining algorithms are not used yet, we hope we can apply them to the data warehouse to discover more interesting patterns.

CHAPTER 8

CONCLUSIONS

8.1 Summary

In this thesis, we have presented an agent-based stochastic model to simulate natural organic matter (NOM). It acts as a starting point of a powerful tool which can be applied to aquatic ecosystem studies, soil and crop science, environmental protection, remediation in the surface and sub-surface, and global climate change prediction. Unlike current models, this model explicitly treats NOM as a heterogeneous mixture, so that distributions of physical, chemical and biological properties can be predicted.

The simulation system employs recent advances in web-based interfaces such as J2EE, and scalable web-based database management systems such as Oracle to improve the reliability and scalability of the stochastic simulations and to facilitate analysis of the resulting large datasets.

Currently, 6 computer systems are involved in the simulation system. To fully facilitate these resources, we employed the idea of load balance and fail over. Therefore, the simulation model is quite reliable and scalable. With the help of Oracle Reports, users can view the reports of their simulations online. Various techniques to improve query performance are discussed. These techniques include indexes, summary tables. To maintain index structures, the insertion performance is decreased about by 20%. Meanwhile, additional storage is needed by the indexes, which is

another overhead of indexes. We found that, in order to obtain the best query performance while the same data set is in a heavy use of insertion, we need to build necessary summary tables which are to be updated periodically.

A data warehouse is built to store aggregated and summarized data for the simulation databases. Detailed data is also stored in the data warehouse. Both star-schema and detail-summary data warehouse technologies are applied when building the data warehouse. Other considerations such as indexes, constraints, are not presented in the thesis, but they are carefully examined when building the data warehouse. To populate the data warehouse, we employed various tools such as SQL*Loader, Export/Import and Copy table data.

Finally, we applied some Oracle data mining algorithms to the data warehouse to discover interesting patterns. For example, we wrote Java programs to apply the Enhanced k-means and O-Cluster algorithms to the POINTS table. A clustering model is built for each algorithm. After comparing the two algorithms, we found that the O-Cluster algorithm has much better performance when building clustering models.

8.2 Conclusions

Through this thesis, we have presented an agent-based stochastic model which employed advanced web and RDBMS technologies typically used in e-business and e-commerce. We can apply these technologies to the field of scientific simulation. The simulation system we presented is quite reliable and scalable. It included the features of load balance and fail over. We also learned that we can apply data mining methods to the simulation data to discover interesting patterns which is useful for other scientists such as chemists, geologists and environmental scientists.

8.3 Future Work

There are several possible extensions of our current project. First of all, more features can be added into the core simulation. Currently, we only allowed ten most popular types of chemical reactions in the model. Obviously, there are far more reactions existing in the real world. We could add a feature to allow the users to build new reaction types.

Secondly, we could apply more data mining algorithms to the data warehouse to discover more interesting patterns. These patterns discovered could be useful in a variety of scientific applications.

BIBLIOGRAPHY

- [1] <http://java.sun.com/j2ee>.
- [2] <http://www.omg.org/technology/uml/>.
- [3] <http://www.santafe.edu/sfi/publications/Working-Papers/97-01-002.pdf>.
- [4] <http://www.swarm.org>.
- [5] *Oracle9i Data Mining*. Oracle Corporation, California, 2002.
- [6] G.R. Aiken, D.M. McKnight, R.L. Wershaw, and P. MacCarthy. *Humic substances in soil, sediment, and water - geochemistry, isolation and characterization*. John Wiley Sons, Berkeley, 1985.
- [7] M.N. Murty A.K. John and P.J. Flynn. Data clustering: A survey. In *ACM Comput. Surv*, 31, pages 264–323, 1999.
- [8] D. Bray and R.B. Bourret. Computer analysis of the binding reactions leading to a transmembrane receptor-linked multiprotein complex involved in bacterial chemotaxis. In *Mol. Biol. Cell*, 6, pages 1367–1380, 1995.
- [9] S.E. Cabaniss, Q. Zhou, P.A. Maurice, Y.P. Chi, and G.R. Aiken. A log-normal distribution model for the molecular weight of aquatic fulvic acids. In *Environ. Sci. Technol.* 34, pages 1103–1109, 2000.
- [10] IBM Almaden Research Center. Chemical kinetics simulator 1.0 user’s manual. In *IBM Corporation*, 1995.
- [11] L. Daufman and P.J. Rousseeuw. *Finding Groups in Data: An introduction to Cluster Analysis*. John Wiley and Sons, New York, 1990.
- [12] G. Dodge and T. Gorman. *Essential Oracle8i Data Warehousing*. John Wiley and Sons, New York, 2000.
- [13] P. Erdi and J. Toth. *Mathematical models of chemical reactions*. Manchester University Press, Manchester, 1989.
- [14] C.A.J.M. Firth. Stochastic simulation of cell signalling pathways. In *University of Cambridge*, 1998.
- [15] C.A.J.M. Firth and D. Bray. Stochastic simulation of cell signalling pathways. In *Computational Modeling of Genetic and Biochemical Networks*, MIT Press, 2001.

- [16] E.H. Han G. Karypis and V. Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. In *COMPUTER*, 32, pages 68–75, 1999.
- [17] S. Chatterjee G. Sheikholeslami and A. Zhang. Wavecluster: A multiresolution clustering approach for very large spatial databases. In *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*, pages 428–439, 1998.
- [18] D.T. Gillespie. A general methods for numerically simulating the stochastic time evolution of coupled chemical reactions. In *J. Comp. Phys.*, 22, pages 403–434, 1976.
- [19] M. Hall. *Core Servlets and JavaServer Pages*. Prentice Hall, New Jersey, 2001.
- [20] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publisher, New York, 2001.
- [21] H.F. Hemond. Acid neutralizing capacity, alkalinity and acid-base status of natural waters containing organic acids. In *Environ. Sci. Technol.* 24, pages 1486–1489, 1990.
- [22] W. Inmon. *Build the data warehouse*. John Wiley and Sons, New York, 1996.
- [23] K. Kalbitz, S. Solinger, J.H. Park, B. Mchalizik, and E. Matzner. Controls on the dynamics of dissolved organic matter in soils:a review. In *Soil Sci.165*, pages 277–304, 2000.
- [24] R. Kimball. *The Data Warehouse Toolkit*. Wiley-QED, New York, 1996.
- [25] R. Kimball. *The Data Warehouse Life Cycle Toolkit*. Wiley Computer Publishing, New York, 1998.
- [26] K.W. Kohn. Functional capabilities of molecular network components controlling the mamalian g1/s cell cycle phase transition. In *Oncogene 16*, pages 1065–1075, 1998.
- [27] J.A. Leenheer, G.K. Grown, P. MacCarthy, and S.E. Cabaniss. Models of metal-binding structure in fulvic acid from the suwannee river, geogia. In *Environ. Sci. Technol.* 32, pages 2410–2416, 1998.
- [28] K. Loney and G. Koch. *Oracle8i: The complete reference*. Osborne McGraw-Hill, Berkeley, 2000.
- [29] H.P. Kriegel M. Ankerst, M. Greunig and J. Sander. Optics: Ordering points to identify the clustering structure. In *In Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*, pages 49–60, 1999.
- [30] J. Sander M. Ester, H.P. Kriegel and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In *In Proc. 1996 ACM-SIGMOD Int. Conf. Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231, 1996.

- [31] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symp. Math. Statist, Prob. 1*, pages 281–297, 1967.
- [32] D.M. McKnight and G.R. Aiken. *Sources and age of humus*. Springer-Verlag, Berlin, 1985.
- [33] R.J. Muller. *Oracle Developer Starter Kit*. Osborne, Berkeley, 1999.
- [34] D.J.M. Park and B.E. Wright. Metasim: a general purpose metabolic simulator for studying cellular transformations. In *Comput. Prog. Biomed. 3*, pages 10–26, 1973.
- [35] E.M. Perdue, J.H. Reuter, and R.S. Parrish. A statistical model of proton binding by humus. In *Geochim, Cosmochim. Acta 48*, pages 1257–1263, 1984.
- [36] J. Price and L. Wald. *Oracle9i JDBC Programming*. Osborne McGraw-Hill, Berkeley, 2002.
- [37] D. Gunopulos R. Agrawal, J. Gehrke and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *In Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 94–105, 1998.
- [38] R. Rastogi S. Guha and K. Shim. Cure: An efficient clustering algorithm for large databases. In *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*, pages 416–427, 1998.
- [39] H.M. Sauro. Scamp: a general-purpose simulator and metabolic control analysis program. In *Comput. Appl. BioSci 9*, pages 441–450, 1993.
- [40] J.W. Shavlik and T.G. Dietterich. *Readings in Machine Learning*. Morgan Kaufmann Publisher, New York, 1990.
- [41] R. Ramakrishnan T. Zhang and M. Livny. Birch: An efficient data clustering method for very large databases. In *In Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'96)*, pages 103–114, 1996.
- [42] J. Yang W. Wang and R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *Proc. 1997 Int. Conf. Very Large Data Bases (VLDB'97)*, pages 186–195, 1997.
- [43] K. Wight R.J. Larson P.H. Masschelen W.F. Punch, A. Patton and L. Forney. A biodegradability evaluation and simulation system (bess) based on knowledge of biodegradation pathways. In *Biodegradability Prediction, Peijnenburg and Darborsky Editors, NATO ASI Series, Number 2, Vol 23*, pages 65–84, 1997.