# EXPERIMENT MANAGEMENT FOR THE PROBLEM SOLVING ENVIRONMENT WBCSIM

by

Jiang Shu

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

APPROVED:

Layne T. Watson, Chair
Frederick A. Kamke
Naren Ramakrishnan
Clifford A. Shaffer
Christopher North

August 10, 2009
Blacksburg, Virginia

**Key words**: Experiment management, problem solving environment, XML, computational steering, computing environment, wood-based composite materials, database management, visualization, optimization.

# EXPERIMENT MANAGEMENT FOR THE PROBLEM SOLVING ENVIRONMENT WBCSIM

by

Jiang Shu

(ABSTRACT)

A problem solving environment (PSE) is a computational system that provides a complete and convenient set of high level tools for solving problems from a specific domain. This thesis takes an in-depth look at the experiment management aspect of PSEs, which can be divided into three levels: 1) data management, 2) change management, and 3) execution management. At the data management level, anything related to an experiment (computer simulation) should be stored and documented. A database management system can be used to store the simulation runs for a PSE. Then various high level interfaces can be provided to allow users to save, retrieve, search, and compare these simulation runs. At the change management level, a scientist should only focus on how to solve a problem in the experiment domain. Aside from running experiments, a scientist may only consider how to define a new model, how to modify an existing model, and how to interpret an experiment result. By using XML to describe a simulation model and unify various implementation layers, changing an existing model in a PSE can be intuitive and fast. At the execution management level, how an experiment is executed is the main concern. By providing a computational steering capability, a scientist can pause, examine, and compare the intermediate results from a simulation. Contrasted with the traditional way of running a lengthy simulation to see the result at the end, computational steering can leverage the user's expert knowledge on the fly (during the simulation run) and provide new insights and new product design opportunities. This thesis illustrates these concepts and implementation by using WBCSim as an example. WBCSim is a PSE that increases the productivity of wood scientists conducting research on wood-based composite materials and manufacturing processes. It integrates Fortran 90 simulation codes with a Web based graphical front end, an optimization tool, and various visualization tools. The WBCSim project was begun in 1997 with support from United States Department of Agriculture, Department of Energy, and Virginia Tech. It has since been used by students in several wood science classes, by graduate students and faculty, and by researchers at several forest products companies. WBCSim also serves as a test bed for the design, construction, and evaluation of useful, production quality PSEs.

## ACKNOWLEDGEMENTS

I would like to express my greatest appreciation for the patience, support and guidance that I received from my advisor Dr. Layne T. Watson in the past nine years. Over these many years, he never gave up on me. Also, I want to thank Dr. Frederick A. Kamke for his help with my fourth journal paper related to computational steering. I want to thank the other three members of my advisory committee, Dr. Naren Ramakrishnan, Dr. Clifford A. Shaffer, and Dr. Christopher North for their sincere support and advice.

Furthermore, I would like to thank posthumously Balazs G. Zombori for his work on mat formation modeling, hot compression modeling, and composite material analysis modeling (documented in Chapter 2). Moreover, I thank Alex Verstak for his work on the telnet server (documented in Chapter 2) in WBCSim. I would also thank Dr. Clifford A. Shaffer and Dr. Calvin J. Ribbens for their effort in organizing research meetings to help students collaborate with each other.

This work was supported in part by a Virginia Polytechnic Institute and State University 1997 ASPIRES grant, USDA grant 97-35504-4697, NSF grants DMI-0422719, DMI-0355391 and DOE contract DE-AC04-95AL97273-91830.

## TABLE OF CONTENTS

## LIST OF FIGURES

# Chapter 1: INTRODUCTION

A problem-solving environment (PSE) is a computational system that provides a complete and convenient set of high-level tools for solving problems from a specific domain [84]. A PSE commonly addresses many issues: Internet accessibility to legacy codes, visualization, experiment management (EM), multidisciplinary support, recommender systems, collaboration support, optimization, high performance computing, preservation of expert knowledge, design extensibility, and pedagogical uses [105]. The research described here takes a specific, in-depth look at the EM aspect of PSEs.

## 1.1 Experiment Management Introduction

In this thesis, the terms "simulation", "model", and "experiment" are used extensively. Therefore, before any serious discussion, a clarification is needed. Naylor et al. [75] carefully defined the term "computer simulation" as a numerical technique for conducting experiments on a digital computer, which involves certain types of mathematical and logical models that describe the behavior of a system or its components over extended periods of real time. Certainly, a computing environment such as a PSE does not have to include computer simulations, nor any mathematical models. However, when providing a complete and convenient set of high-level tools for solving problems from a specific domain [84], PSEs often simulate certain processes and/or model the problems. For example, in WBCSim (described in Chapter 1.2), a hot-pressing model is used to simulate the heat and mass transfer during the hot-pressing manufacturing process for wood-based composite panels. Naylor et al. [75] further classifies models by five degrees of abstraction: 1) the process, activity, or situation itself, 2) a replication of the process, 3) a controlled, laboratory-type model of the process, 4) a completely synthetic extraction of essential elements of the process (such as a computer model), and 5) a closed analytical model. Traditionally, an experiment is interpreted as physical measurement of some certain variables. However, with the advent of high-speed computers in the 1950's, it became possible to model experiments on a digital computer. In the context of this thesis, the term "simulation" means a computer simulation; the term "model" means a computer model, and the term "experiment" (a numerical experiment on a computer) has the same meaning as the term "simulation".

Everyday, experimental scientists from various disciplines, such as physics, chemistry, and biology, whether using a PSE or other computing environments, perform numerical experiments on their computers. Each of these numerical experiments (hereafter referred to as an experiment) goes though similar life cycles even though these sciences have very little in common. Ioannidis et al. [53] identified three stages in a typical experiment life cycle: 1) design of experiment, 2) data collection, and 3) data exploration (shown in Figure 1).


Fig. 1.1. A typical numerical experiment life cycle.

A much more complex diagram of the experiment life cycle [52] further dissects the data exploration stage and orders the arcs in the diagram based on their frequencies.

In the experiment design stage, scientists need to think of the structure of an experiment and how to implement this experiment on a computer. Scientists also need to specify which control variables enter the experiment as the input, and what will be collected as the results. In the data collection stage, the experiments are performed, and the output data are gathered. The size of the data generated from experiments may vary greatly, from being only several megabytes as in most small laboratories, to being several terabytes as in the NIH Human Genome project [36]. In the data exploration stage, scientists validate and study the data collected. Various techniques and tools can be used for analyzing, visualizing data, or comparing one experiment to another, as well as searching for a trend among several experiments in order to optimize some experiment results. A satisfactory design of an experiment is rarely achieved in a single attempt. This process can undergo many iterations, which typically includes the execution of some experiments and analysis of the obtained data, before the design reaches its final form [52].

Scientists in these fields need an adequate tool to manage those numerical experiments. Besides the possible evolving nature of the designing experiment stage, an experiment management system should focus on data collection and data exploration. Each experiment consists of a model, some input, and some results. Therefore, a representation of the space of all these experiments is needed. The possible combinations of experiment input (for a fixed model) form a multi-dimensional experiment space, where there is one dimension for each input variable and one point for each experiment.

The common method used to explore the experiment space is a query facility. This query facility takes a well-defined query (it could be retrieving a particular experiment, or listing all the experiments on a particular date), and returns the corresponding results. Furthermore, techniques for quantitatively and automatically comparing two or more experiments are needed. Scientists need to figure out what to compare, how to compare, and what questions the comparison can answer. The ultimate goal of the data exploration is to enhance an experiment (such as parameter tuning) and its performance (such as automated optimization). At this level, an EM system may predict an experiment result based on the historical data stored in the experiment space [35].

In this thesis, experiment management is divided into three levels: data management (Chapter 3), change management (Chapter 4), and execution management (Chapter 5), which correspond in some ways to "data collection", "design of experiment", and "data exploration" (from Figure 1).

## 1.2 WBCSim Introduction

WBCSim is a PSE that increases the productivity of wood scientists conducting research on wood-based composite (WBC) materials and manufacturing processes. It integrates Fortran 90 simulation codes with a Web based graphical front end, an optimization tool, and various visualization tools. The WBCSim project was begun in 1997 with support from USDA, Department of Energy, and Virginia Polytechnic Institute & State University (VPI&SU). It has since been used by students in several wood science classes, by graduate students and faculty, and by researchers at several forest products companies. WBCSim also serves as a test bed for the design, construction, and evaluation of useful, production quality PSEs.

Goel et al. [41], [42] described an early version of WBCSim. Over the years, WBCSim has evolved in many ways. More advanced simulation models such as a mat formation model and a hot compression model are now incorporated into the system [91]. An experiment management tool [90] is also included now. This EM tool uses a database management system (DBMS) to store simulation input and results at the server end, provides helpful graphical interfaces to facilitate user access at the client side, and then uses various scripts to connect all of these components. Recently, XML (Extensible Markup Language) was used to unify the implementation layers of WBCSim [89]. A XML datasheet is tailored for each model (WBCSim has five models). The WBCSim interface layer, server scripts, and database management system all use this XML datasheet to improve the usability and maintainability of the three layers—client, server, developer—comprising the WBCSim system. Furthermore, with a reasonable number of changes and a small set of components, a computational steering capability is now added to WBCSim. Computational steering can effectively reduce computational time, make research more efficient, and open up new product design opportunities.

## 1.3 Organization

Chapters 2–5, based on four journal papers ([91], [90], [89], [88]), show the evolution of WBCSim since 2002. The chapters occur in the order the papers were written, not in the order they were published (the first paper wasn't published until 2006 due to editorial delay for a special issue). Chapter 2 describes all five simulation models in great detail. It emphasizes two new and more advanced models: a mat formation model and a hot compression model. At that point in time, WBCSim still used Java applets through Web browsers and a telnet server. Chapter 3 elaborates on an experiment management component for WBCSim. This

experiment management component provides graphical interfaces (again Java applets) to allow users to save, retrieve, and compare simulation runs (input and results) by using a database management system (DBMS). Chapter 4 explains how XML is used to unify the implementation layers of WBCSim. A XML datasheet is tailored for each model. A simple change in the XML can cause cascading changes at various implementation layers, thereby significantly simplifying the development effort and improving maintainability. This chapter also shows new PHP Web interfaces for WBCSim. Chapter 5 demonstrates WBCSim's new computational steering capability. Computational steering allows scientists to adjust parameters of the computation on-the-fly and explore "what if" analysis [38]. Chapter 6 summarizes all these changes/enhancements throughout the years. Chapter 7 outlines some future directions for the experiment management research and WBCSim, and also draws conclusions.

Chapters 2–5 correspond to four standalone journal papers written at different times, so there are some overlaps in the sections and figures. Some sections from different papers/chapters may describe the same thing, for example, the five simulation models. However, a careful reading reveals how WBCSim has evolved over the years. Some figures may be the same or look similar, such as the WBCSim architecture diagrams. Some figures may have the same caption, for example, Figures 2.3, 4.3, and 5.1 are all captioned "The HC model user interface". However, these figures are different and illustrate how WBCSim changed from a Java applets interface to a PHP web interface, and a different aspect of WBCSim—computational steering.

# Chapter 2: The Modeling Environment WBCSim*

## 2.1 Introduction

Scientists often conduct experiments involving many parameters, and cannot afford the time and cost to pursue every combination of those parameters. Therefore, they rely on computer modeling and simulation to assist them in exploring the parameter space. However, most of the time, the modeling and simulation codes are developed in-house and have many limitations (relevant commercial codes may not exist or may not be affordable). Often not only is the code tied to a fixed computing environment, making it hard to access or exchange with other research groups, but it is also very common that the code is not integrated with any visualization and optimization tools. Moreover, since the developers of a code are often the users of that system, the result is poor documentation and an unsophisticated user interface, so that only the developers themselves can utilize the system efficiently. This situation obviously hinders the productivity of many research groups. This chapter describes a computing environment WBCSim that increases the productivity of wood scientists conducting research on wood-based composite (WBC) materials and manufacturing processes. WBCSim integrates Fortran 90 simulation codes with a Web-based graphical front end, an optimization tool, and various visualization tools.

Goel et al. [41] described an early version of WBCSim in 1997. Since then, WBCSim has evolved, taking different approaches to its architecture, adding more sophisticated models, and switching from experiment-oriented to manufacturer-oriented. However, the original goals remain the same: (1) to increase the productivity of WBC research and manufacturing groups by improving their software environment, and (2) to continue serving as an example for the design, construction, and evaluation of small-scale problem solving environments (PSEs). In this chapter, a detailed description of the current system architecture, five different WBC models, and a typical scenario of usage are discussed, along with optimization and visualization features.

Goel et al. [41] give a more complete description of what constitutes a PSE and why WBCSim is a PSE. In general, a PSE provides an integrated set of high-level facilities that support users engaged in solving problems from a proscribed domain [40]. Also a PSE commonly addresses the following issues: Internet accessibility to legacy code, visualization, experiment management, multidisciplinary support, collaboration support, optimization, high performance computing, usage documentation, preservation of expert knowledge, recommender systems, and integration [86]. WBCSim qualifies as a PSE because it makes legacy simulation codes available via the WWW, is equipped with visualization and optimization tools, is multidisciplinary in supporting different disciplinary simulation

* WBCSim: an environment for modeling wood-based composites manufacture, J. Shu, L.T. Watson, B.G. Zombori, F.A. Kamke, *Engineering with Computers*, 21 (2006) 259–271.

models, and has experiment management, collaboration support, and high performance computing being added.

The chapter is organized as follows. Chapter 2.2 reviews some related work in PSE and WBC computer-based systems. Chapter 2.3 describes the five simulation models currently supported by WBCSim. Chapter 2.4 elaborates on the WBCSim user interface, the model interface similarities, and also the unique aspects of each. Chapter 2.5 explains the various architecture layers of WBCSim, and also the new WBCSim telnet communication connection scheme. Chapter 2.6 demonstrates a typical simulation scenario. Chapter 2.7 describes the WBCSim optimization tool. Chapter 2.8 explains the visualization tools involved. Finally, Chapter 2.9 outlines some future directions for WBCSim.

## 2.2 Related Work

There are many problem-specific PSEs developed for various application domains, and several computer-based mathematical models were developed to solve particular problems in the wood-based composites industry. However, no work known to the authors addressed the integration of mathematical models with a PSE environment. WBCSim is a novel approach to fill this gap and provide a valuable tool for the wood-based composites industry.

Goel et al. [41] describe some early PSEs such as ELLPACK and its descendents [8] for solving two and three dimensional elliptic partial differential equations (PDE), SCIRun [77] developed to interactively compose, execute, and control a large-scale computer simulation, and Linear System Analyzer [14] for manipulating and solving large-scale sparse linear systems of equations. Since then, many new PSEs have been introduced. Gismo [18], created at Washington University, is an object oriented Monte Carlo package for modeling all aspects of a satellite's design and performance. It has played a significant role in the design of the Gamma Ray Large Area Space Telescope, the successor to the Compton Gamma Ray Observatory that was launched into space in 1991 for exploring the gamma ray portion of the electromagnetic spectrum in astrophysics. VizCraft [39], developed at Virginia Polytechnic Institute and State University, provides a graphical user interface to a widely used suite of analysis and optimization codes to aid aircraft designers during conceptual design of a high-speed civil transport. VizCraft combines visualization and computation, encouraging the designer to think in terms of the overall problem-solving task, not simply using the visualization to view the computation's results. There is also a computing environment developed by Chen et al. [24] that combines particle systems, rigid-body particle dynamics, computational fluid dynamics, rendering, and visualization techniques to simulate physically realistic, complex dust behaviors useful in interactive graphics applications for education, entertainment, or training.

An emerging practice in the wood composites industry is to use mathematical models to describe the pertinent relationships between various manufacturing parameters and final

composite properties. The most influential stages of composites production are the mat formation and the hot-compression processes, therefore the modeling effort is concentrated on these two areas. However, most of the mathematical models were implemented as command line based programs with standard text input and output, and only a few attempts were made to create more user friendly environments.

A mat formation model is required to establish the critical relationships between the structure of the composite and the dynamic change of certain physical properties during the mat consolidation. One of the most notable examples is commercially available mat formation software developed by Forintek Canada Corporation. The software, based on Dai's mathematical model [27], incorporates geometric probability theory and simulation techniques for describing the characteristics observable on the surface of the flake mat. The basis of the model is an idealized, randomly formed, flake layer network, where the dimensions of the flakes are uniform, and the number of the flakes is limited in a manner that the sum of the areas of the flakes is equal to the area of the layer. Given these assumptions, several properties of the flake network are mathematically definable using probability distributions and random field theory. The composite mat is constructed from these single layers of flakes. The model can calculate the horizontal density distribution of the mat, the flake-to-flake contact area, and the void volume fraction. The commercial software version of the model is only accessible for a fee.

Another mat formation model created by Lu [67], called WinMat, works as a stand alone application in all Windows operating systems. Besides the major objective of WinMat to generate commands for an industrial robot, which is capable of putting together mat structures from uniform size flakes, several additional features were included in the program. It can create a single or a three-layer OSB structure from flakes either with fixed dimensions or with dimensions described by a normal, uniform, Poisson, or binomial distribution. Additionally, the orientation of the flakes can follow a uniform or a vonMises distribution. Although the interface to the program is easy to use, several aspects of the model are deficient. The density of the flakes is constant, not allowing mats built from mixed wood species. Additionally, the number of the deposited flakes in each layer is controlled by the actual flake number instead of the more realistic cumulative flake weight.

Less computer based than the above, another mathematical modeling approach to characterize the spatial structure of randomly formed wood flake composites is referred to as the edge model [65], [63], [64]. This model subdivides the mat into imaginary flake columns of finite size. The geometrical properties of the columns are determined by experiments and measured at the edge of the mat. Probability density functions are then fit to the data. The columns are reproduced in the simulation model by sampling the fitted probability distributions. Although the edge modeling approach is unique, no attempt was made to include a graphical user interface or advanced result visualization.

Zombori et al. [112] developed a more realistic model of the mat structure, where the mat formation includes the geometry of the wood elements as random variables with certain limitations imposed on the orientation of the elements. Additionally, a density value is assigned to each individual strand, allowing the simulation of the horizontal density distribution of composites made out of mixed species. The mat formation process model was developed based on data obtained from commercial strand measurements and was validated by comparison of the simulated horizontal density distribution to a measured horizontal density distribution. None of the previous models could predict the real structure and horizontal density distribution of a commercial, three-layer OSB mat, based on geometry and density measurements of industrial strands.

A hot-compression model describes the heat and mass transfer and adhesive cure during the consolidation of a mat of wood elements into a composite panel. The first of a series of models was published by Humphrey and Bolton et al. [10], [11], [12], followed by a more sophisticated model, including steam injection, by Haselein [46]. This was further improved by Thömen [94], who modeled the internal environment in a continuous hot-press.

All these models had inherent limitations; either they were one-dimensional, or gross simplifications were made about the transfer mechanisms. Additionally, no attempt was made to integrate the mathematical heat and mass transfer models within a PSE. The hot-compression model of Zombori [111] allowed a comprehensive two-dimensional description of the heat and mass transfer phenomena during the mat consolidation. All conceivable simultaneous heat and mass transfer mechanisms were considered, together with phase balancing sorption isotherms. By numerically solving the governing partial differential equations, the evolution of moisture and temperature profiles in the vertical midplane of the board could be predicted.

The oriented strandboard mat formation and hot-compression simulation models provided in the PSE WBCSim are a logical continuation of the previously described models. Generally, both of the models implement a wider range of functionalities than any of the mathematical models used in the wood-based composites industry. WBCSim supports these models with intuitive graphical user interfaces, from a large selection of model input variables to sophisticated visualization and optimization techniques.

## 2.3 Simulation Models

WBCSim currently supports five simulation models that help wood scientists studying wood-based composite material manufacturing. Each of these models is described next.

### 2.3.1 Rotary Dryer Simulation (RDS)

The rotary dryer simulation model assists in the design and operation of the most common type of system used for the drying of wood particles [58], [59]. The rotary dryer is an integral part of the processes to manufacture particleboard and strandboard products. It

consists of a large, horizontally oriented, rotating drum (typically 3 to 5 m in diameter and 20 to 30 m in length). The wet wood particles are mixed directly with hot combustion gases, in a co-current flow pattern, at the inlet to the rotating drum. The gas flow provides the thermal energy for drying, as well as the medium for pneumatic transport of the particles through the length of the drum. Interior lifting flanges serve to agitate and produce a cascade of particles through the hot gases.

The RDS model consists of coupled material and energy balance equations for each segment along the length of the drum. Each drum segment is defined by the cascading pattern of particle travel. The segment, or cycle, begins when a particle drops off a lifting flange and falls to the bottom of the drum. This is followed by travel along the periphery of the drum, where the particle is caught by a lifting flange. The segment ends when the particle attains its maximum angle of repose and tumbles off of the lifting flange. The user must supply the inlet conditions of the hot gases and wet wood particles, as well as the physical dimensions of the drum and lifting flanges, flow rates, and thermal loss factor for the dryer. The RDS model predicts the particle moisture content, temperature, gas composition, and energy consumption.

### 2.3.2 Radio-Frequency Pressing (RFP)

The radio-frequency pressing model [83] was developed to simulate the consolidation of wood veneer into a laminated composite. The energy needed for cure of the thermosetting adhesive is supplied by a high-frequency electric field. Radio-frequency pressing is commonly used for thick composites and for nonplanar laminated composites. The model may be used to help design alternative pressing schedules.

The RFP model consists of a collection of nonlinear PDEs that describe the heat and mass transfer within the veneer layers. The primary variables are temperature and moisture content. The moisture content is further divided into three phases: bound water, liquid water, and water vapor. These water phases must satisfy a criterion of local thermodynamic equilibrium as represented by a nonlinear algebraic equation. The model is one-dimensional, with a fixed resistance to heat and mass flux at the boundary. The RFP model predicts the time-dependent temperature and moisture content profiles in the veneer layers, as well as the extent of adhesive cure. Among the user-supplied input data are the initial density, thickness, moisture content, and temperature of the veneer, as well as the electric field strength.

### 2.3.3 Oriented Strandboard Mat Formation (OSB)

The mat formation model creates the three-dimensional spatial structure of a layered wood-based composite (e.g., oriented strandboard and waferboard) and calculates certain mat properties by superimposing a mesh on the mat structure. The model is based on a Monte Carlo simulation technique. It is assumed that the dimensions of the constituents (e.g.,

strands) of the mat follow certain probability distribution functions. The parameters of the probability distribution functions have to be determined on a subsample by measurement. The user inputs the number of layers. The total number of strands is determined by specifying that number, or specifying the weight, volume, or area of each layer. The probability distribution functions of the length, width, thickness, and density of the strands are given by the user.

Output from the mat formation model includes the horizontal density distribution, proportion of void space, and potential bonded surface area. Output data is available in a numerical format and two- or three-dimensional visualizations. The output from the mat formation model is required for the hot compression model.

### 2.3.4 Hot Compression (HC)

The hot compression model simulates the mat consolidation and adhesive cure that occurs during industrial hot-pressing of wood-based panels. The model is based on fundamental engineering principles and uses the output from the mat formation model to establish the starting spatial structure of the mat. Six primary variables are considered: mat density, air content, vapor content, bound water content, and temperature within the mat, and the extent of the cure of the adhesive system characterized by the cure index. The heat is transported by conduction and bulk flow, while the water phases are transported by bulk flow and diffusion. A nonlinear viscoelastic relationship was used to describe the compression behavior of the mat. This relationship separates the geometric nonlinear response of the cellular structure of the wood elements from the linear viscoelastic response of the wood cell wall polymers. The behavior of the cellular structure is modeled with a modified Hooke's Law. The viscoelastic properties of the flakes are described by the time-temperature-moisture equivalence principle of polymers.

The resulting differential-algebraic system of equations is solved by a semicontinuous finite difference method. The spatial derivatives of the conduction terms are discretized according to a central difference scheme, while the spatial derivatives of the bulk flow terms are discretized according to an upwind scheme. The resulting ordinary differential equations (ODEs) in the time variable are solved by DDASSL, a freely available differential-algebraic system solver. The model predicts temperature, moisture content, partial air and vapor pressures, total pressure, relative humidity, extent of adhesive cure, and the spatial density profiles within the mat. A set of three-dimensional graphical profiles illustrate the evolution of these variables with time, in the thickness and width dimensions of the mat. The model assists users in the understanding of the interacting mechanisms involved in a complex production process. The model may also be used to optimize the hot-pressing parameters for improved quality of wood-based panel products, while minimizing processing cost.

### 2.3.5 Composite Material Analysis (CMA)

The composite material analysis model was developed to assess the stress and strain behavior and strength properties of laminated materials (e.g., plywood and fiber-reinforced composites). The user defines the layer sequence of the laminate. The graphical user interface was designed to allow easy specification of the material, thickness, and orientation at each layer. The mechanical and failure properties of the layer materials may be selected from a predefined list. The model can perform design and analysis functions, where the user either defines the loading condition, or defines the deformation.

The calculations are based on the classical lamination theory (CLT) and the Tsai-Wu failure criteria. The "Design" and "Analysis" models calculate the induced stresses and strains in the primary laminate $(x, y)$ and principal fiber $(1, 2)$ directions within each layer based on the CLT, and check for the integrity of the layers of the laminate by the Tsai-Wu failure criteria. In the design mode, the model calculates the stresses and strains caused by the combination of different loading conditions, such as tension, moment, torque, or shear. In the analysis mode, the normal and shear stresses, together with the strains and curvatures induced by a user-defined deformed shape, are calculated. The model predicts the tensile strength, bending strength, and shear strength of the composite material.

## 2.4 User Interfaces

WBCSim is Web-based; therefore, its user interface is composed of a Web browser and Java applets. A user launches the WBCSim Web page from a browser window, and then invokes applets from the Web page. The very first applet allows the selection of a simulation model. From that point on, all user interaction with the system is via applets. There are some common features among all the model interfaces (see Fig. 2.1). At the top left corner of a model interface, there is a "Usage Instructions" button, whose pop-out window explains the model behavior at an abstract level, and also gives general directions about how to use the model. The middle region of each model interface is model-dependent, allowing a user to specify simulation parameters in any number of text boxes, drop down lists, radio buttons, or plain buttons, which possibly open pop-out windows for editing even more simulation parameters. Near the bottom of the interface, there is a long text box where a user can describe the simulation; this description is saved when the simulation data is saved. At the very bottom of an interface, there is a row of buttons, which control model-level actions for a simulation. Among those buttons are the four: "Retrieve Problem", "Run Simulation", "Store Problem", and "Dismiss". The "Store Problem" button is used to store the current set of input values (along with the simulation description text), which can be retrieved later using the "Retrieve Problem" button. Clicking on the "Run Simulation" button fires input values defined in the interface through a telnet connection to the WBCSim server, which in turn calls a UNIX or Perl script to execute compiled FORTRAN code, and additional

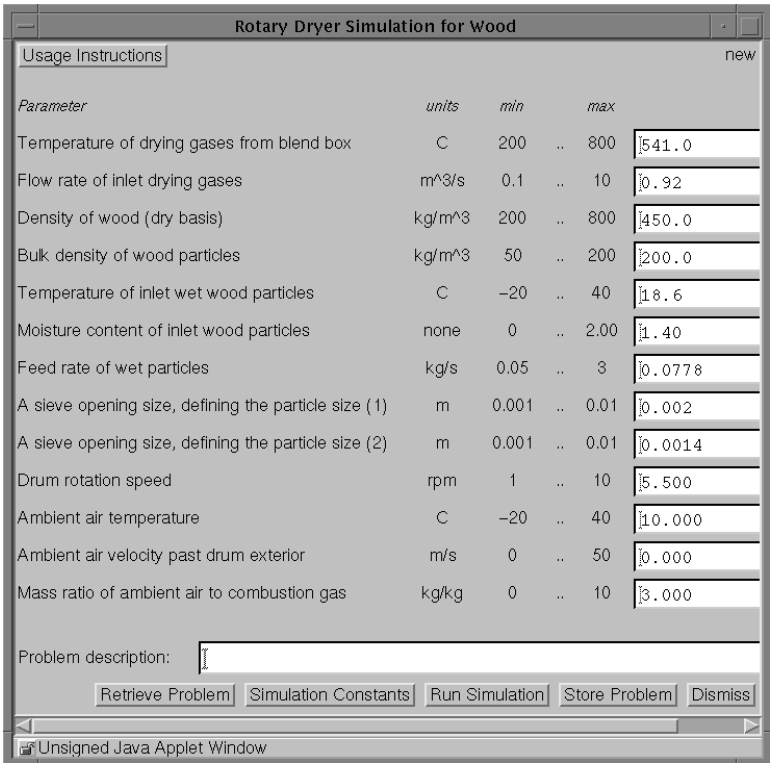


Fig. 2.1. The RDS model user interface.

optimization or visualization codes. (The details of this communication process will be described in Chapter 2.5.) The "Dismiss" button dismisses the interface without saving any input. Other than these four main control buttons, there are some other buttons varying from model to model that may appear in the same row, such as the "Simulation Constants" button (lists some simulation parameters not accessible by the user), and the "Set Default" button (sets all the model variables to a predefined set of values). The results from running a simulation model are usually available in both textual (normally tables of numbers) and graphical (VRML files, GIF files) forms, and displayable in browser windows.

The RDS (Fig. 2.1) and RFP models have the simplest user interfaces in WBCSim. The user simply enters values for various input parameters through text boxes, then runs the simulation. The CMA model has evolved since [41]. Depending on the context, "model" refers to the computer simulation code itself, or the interface to that code. The previous CMA model [41] supported five calculations: "analysis", "design", "tensile strength", "bending strength", and "shear strength". When the user selected any of the five, the user was locked to that calculation and could not switch from one calculation to another without returning to the beginning. Also, in each calculation interface, many parameters were fixed or assumed, whereas the current CMA model interface provides text boxes for those previously fixed parameters. The current CMA model has a new calculation "general strength", and

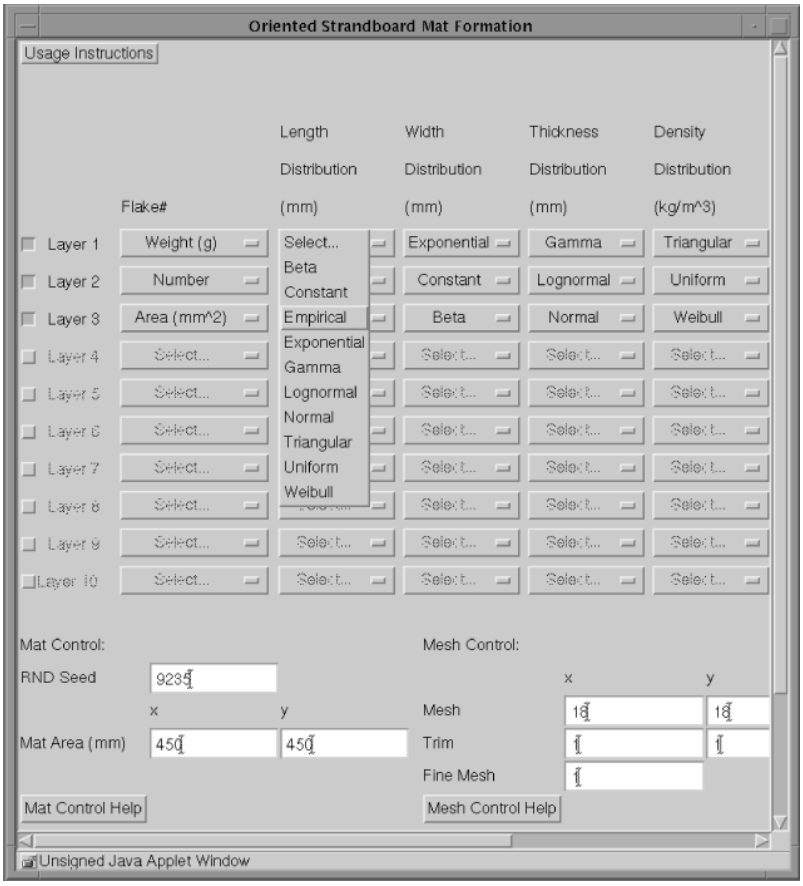


Fig. 2.2. The OSB model user interface.

integrates all the calculations into one interface, allowing the user to select a dynamically loaded calculation from a drop down list. The current CMA model also includes two sets of defaults, structure and loading (both in drop down lists). This new integrated interface allows the user to cross-reference different sets of defaults by setting one structure parameter set and then running any calculation with any loadings.

The OSB model (Fig. 2.2) is similar to the CMA model with rows of input data representing layers of the wood composite. The user activates a layer by selecting the leftmost checkbox for that layer. Also as in the CMA model, the OSB model provides a set of defaults rather than a single default like most of the other models. However, the rest of the OSB model is more complex than that of CMA, since it involves many pop-out windows when the user chooses an item from a drop down list or clicks on a button. For example, selecting the "Empirical" distribution in the "Length Distribution" drop down list, a pop-out window containing forty-two text boxes appears, allowing the user to define the number of intervals and each interval's starting and ending points. The OSB interface

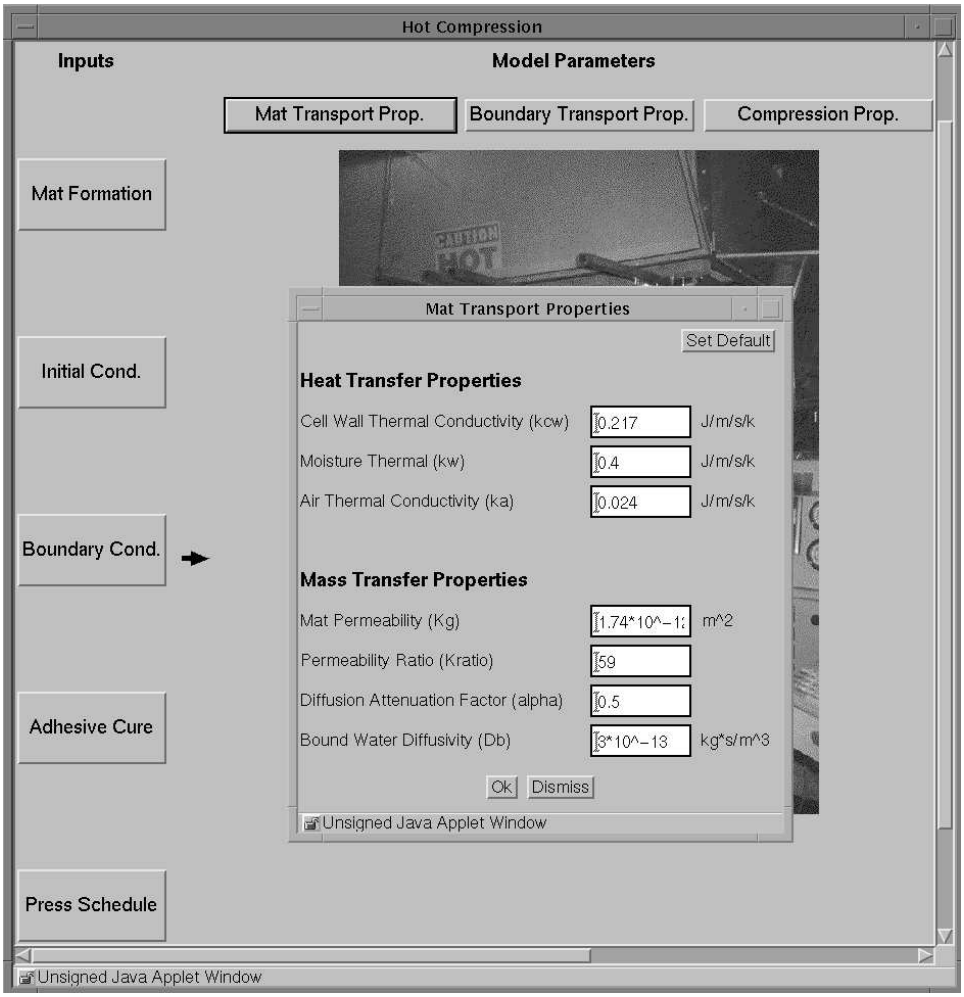


Fig. 2.3. The HC model user interface.

can also save the values defined in pop-out windows, which means those pop-out windows can be closed and then opened again for editing.

The HC model (Fig. 2.3) interface differs from the other models' interfaces by dividing the simulation parameters into three groups partitioned into columns: (1) model input, (2) model execution specification, and (3) model output. The first column contains buttons for specifying the parameters related to material, initial condition, boundary condition, adhesive cure, and press schedule. The second column defines the HC model execution by specifying mat transport properties, boundary transport properties, and compression properties. The last column specifies the output such as color or black and white, the time interval for generating frames for animation, and other parameters determining which data/image files are generated.

## 2.5 Software Architecture

The current software architecture of WBCSim follows the three-tier model described by

Goel et al. [41], for which the tiers are (1) the developer layer—legacy simulation codes and various optimization and visualization tools running on the server, (2) the client layer—user interface, and (3) the server layer—telnet server and a custom shell. These layers are shown in Fig. 2.4.

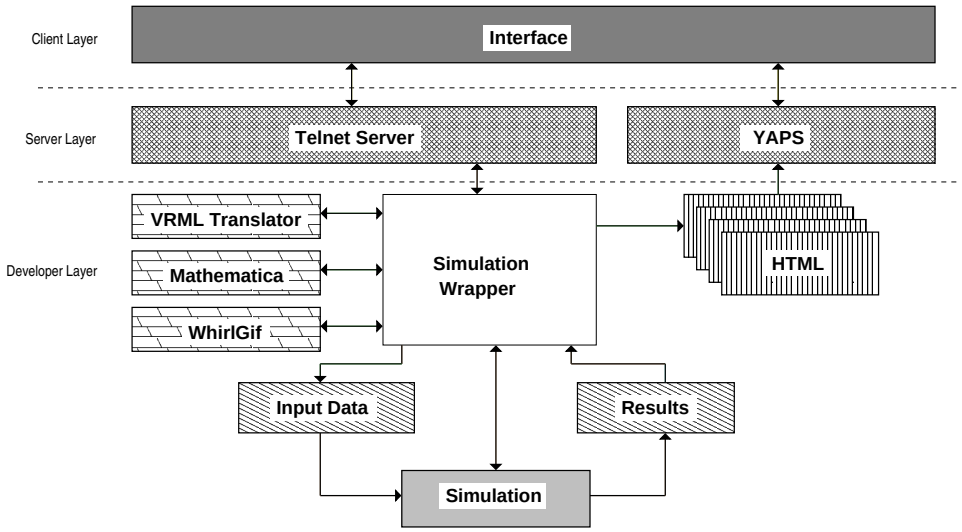


Fig. 2.4. WBCSim architecture overview.

### 2.5.1 Developer Layer

As its name suggests, the developer layer consists of legacy programs created by researchers to model WBC materials and manufacturing processes. These legacy programs are the heart of WBCSim. In general, WBCSim supports legacy programs written in any programming language as long as the program takes its input parameters from UNIX stdin. The input parameters can be a few numbers or a large data file. In particular, WBCSim supports five FORTRAN 77 and Fortran 90 simulation programs corresponding to the five models described in Chapter 2.3.

While each Fortran program has its own input format (stdin could be redirected to a file), the server layer communicates data with the developer layer via strings of parameters separated by white space. Therefore, in order to cope with this string format, each legacy program is "wrapped" with a customized Perl script. The script receives this string of parameters from the server, and converts those parameters into an appropriate format for that Fortran program. Then the script calls the legacy program into action feeding it the input, invokes any required optimization and visualization tools, and finally packs all the Fortran output in HTML files and passes their URLs to the server. With this architecture, the developer layer is independent of the other layers, which makes the process of designing, and integrating new simulation codes relatively easy.

The developer layer also includes optimization and visualization tools to maximize the simulation's value to the user. Optimization and visualization tools will be described in Chapters 2.7 and 2.8, respectively.

### 2.5.2 Client Layer

The client layer is the only layer visible to end-users and typically the only layer running on the local machine. Besides the user interfaces described in Chapter 2.4, the client layer also contains viewers for one of the visualization tools—the VRML Translator. WBCSim requires a VRML 2.0 viewer for the RFP model. The VRML viewer serves as a plug-in to the Web browsers.

The client layer also handles the communication with the server layer. After the user enters all the necessary parameters and triggers the "Run Simulation" button, the client sends those parameters along with a request for executing the corresponding simulation to the server layer via a telnet connection. When the simulation terminates, the client takes the URLs returned by the server layer, and directs them into the user's browser. Of course, the client may also send requests for save and retrieve operations.

The telnet connection method replaces the old Javamatic server and socket communication paradigm described in [41], and facilitates setting up WBCSim guest accounts on the server machine that do not require full account privileges. The guest account and normal user accounts (for commercial, paying users) permit users to save and track their different simulation runs, and protect their data from being removed or overwritten by others. The client side of this telnet connection manages the pool of connections and provides a means for executing remote commands on the server.

### 2.5.3 Server Layer

The server layer, by separating the legacy simulation codes from the user interface, is the key to how WBCSim can run a text-only application from a Web browser. The server layer consists of two components, a telnet server and a custom shell to facilitate this server-client communication.

The telnet server is not a replacement for a standard telnet server, implementing only enough of the telnet protocol to work with the WBCSim telnet client. The telnet server supports guest and regular logins and all the operations provided by the previous Javamatic server, which could direct execution of multiple simulations and accept multiple requests from the client concurrently [41].

Yet Another PSE Shell (YAPS) is a simple Perl script that the client invokes when it logs in via the telnet connection. Therefore, the client talks to this shell instead of the UNIX login shell of the account. Among the commands that YAPS supports, the following four are most important.

(1) The *store* command is used to permanently store a problem (input parameters, text description, simulation output) on the server. Upon request, a temporary archive file is created. The problem description and simulation parameters are written to this file. Notice that it is possible to store problem parameters without running the simulation, in which case there is no simulation output.

(2) The *remove* command removes one or more stored problems. It actually removes files from the file system, so the results are irreversible.

(3) The *load* command retrieves all stored problems from a directory. This command prints a collection of problem IDs (each problem definition or simulation run has a unique problem ID), parameters, descriptions, and URLs, which could be picked up by the telnet client.

(4) The *clean* command deletes temporary files. Normally cleanup is requested when the "simulation results" window is dismissed. However, cleanup requests are only executed if there is an idle connection. If no idle connection exists, cleanup requests are queued until a connection becomes available. The goal of this policy is to clean up only when doing so incurs no extra connection. Making an extra connection involves a lengthy bootstrap process and possibly passing the user password again.

There is no command to run a simulation because the simulation wrapper can be invoked directly by YAPS based on the requests coming from the telnet client.

## 2.6 Simulation Scenario

Describing a typical usage scenario of WBCSim is instructive. Consider research into the properties of oriented strandboard products, which would use the OSB model. First, a scientist collects a subset of the face and core layer flakes from an industrial production line. The scientist measures the geometry of the flakes and the weight by using an image analysis system and a scales, respectively. The flake property data sets are then used to estimate the statistical probability density functions of the flake properties. Then, the scientist opens a browser window and launches WBCSim. The guest account is the default login option, which stores all the input and output data in a temporary directory that will be cleaned periodically. The scientist selects the OSB model from a dropdown list of all the available models, and launches the OSB model interface (shown in Fig. 2.2). By specifying the probability density functions of the flake properties obtained offline earlier, the scientist can recreate the spatial structure of multiple layers of flakes, each with various parameters defining the number of flakes, length distribution, width distribution, thickness distribution, density distribution, orientation, flake color, and number of flakes shown. The latter two are solely for visualization.

Next, the input parameters (Boolean, numeric, or alphanumeric) are sent to the WBCSim server as a long string (parameters are separated by white spaces) via the telnet connection.

Then the OSB wrapper converts this string into a data file designated for the OSB Fortran 90 simulation program. Also, since the OSB simulation code has its own text-based user interface taking input from stdin, a temporary file is generated to contain all the appropriate commands, and stdin is redirected to this temporary data file for the simulation. Then the OSB wrapper calls the OSB simulation code with this temporary file as stdin along with the properly formatted data file.

When the simulation code is executing, the OSB wrapper listens to the simulation output stream for strings indicating execution milestones. The OSB wrapper uses the standard error stream for sending these messages to the client, because Java buffers the standard output stream until the process terminates, while the contents of the standard error stream are sent immediately. The OSB wrapper also sends other messages (such as when visualization or optimization tools are invoked) to the client, displaying the simulation status to the scientist. However, due to network delay, a group of messages generated at different times can arrive at the client at the same time, in which case the client will only display the latest message and discard any old ones.

When the OSB Fortran 90 simulation program terminates, the OSB wrapper calls Mathematica to read the simulation results and generate plots with various Mathematica commands such as ListContourPlot, ListPlot3D, MultipleListPlot, and Graphics3D. Mathematica also converts these internal graphics data structures into GIF format so they can be viewed in a browser. Fig. 2.5 shows a three-dimensional visualization of a three-layer random flake mat created with Graphics3D by Mathematica. Finally, the OSB wrapper embeds these GIF files in HTML files and returns the URLs of these HTML files to the client.

Upon closing the interface windows, the scientist has a choice of storing (archiving) this particular run's input values and results, or discarding this run (the default). Based on the user's request, the YAPS shell will take the proper actions as described in Chapter 2.5.

## 2.7 Optimization

Scientists typically use a PSE like WBCSim iteratively with some design or system behavior goal in mind. Their exploration is by trial and error, guided by intuition that improves as more computational experiments are performed. Mathematical optimization is a powerful tool for automating this tedious search process. Currently the optimization tool provided with WBCSim is the program DOT (Design Optimization Tool) [96], based on sequential quadratic programming and the method of feasible directions. DOT is a robust sophisticated FORTRAN 77 subroutine for nonlinear constrained optimization, and is widely used in engineering.

WBCSim supports two models, RDS and RFP, that are linked to DOT. Each has a different interface for optimization than for analysis. Comparing the interface without

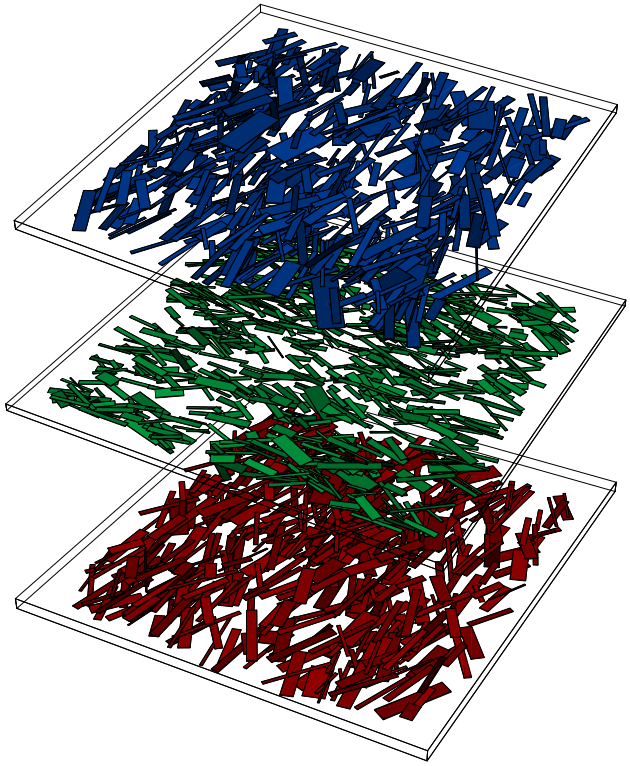

Fig. 2.5. Visualization of the three-layer random flake mat: the area of the mat is $450 \times 450$ mm. Only a fraction of the flakes are shown, and the shading designates the density of the flakes.

the optimization option to the one with optimization, the former provides only one text box for every simulation parameter, while the latter provides three, one for specifying the initial value for that parameter, one for specifying the maximum value, and one for specifying the minimum value. The optimization interface also allows the user to specify which parameters are fixed and which are to be varied. The optimization interface also supplies a number of predefined objective functions (typically computed physical quantities, such as temperature or pressure), for which the user can specify either maximization or minimization. The optimization interfaces necessarily provide parameters to control the convergence of the optimizer, such as "Maximum absolute change in the objective function", "Maximum relative change in the objective function", "Relative finite difference step for gradients", and "Maximum number of iterations".

## 2.8 Visualization

In principle, WBCSim supports the addition of any visualization tools to convert simulation output to a format meaningful to the user. However, it does require the user to have the corresponding viewer for the format of that visualization result installed and

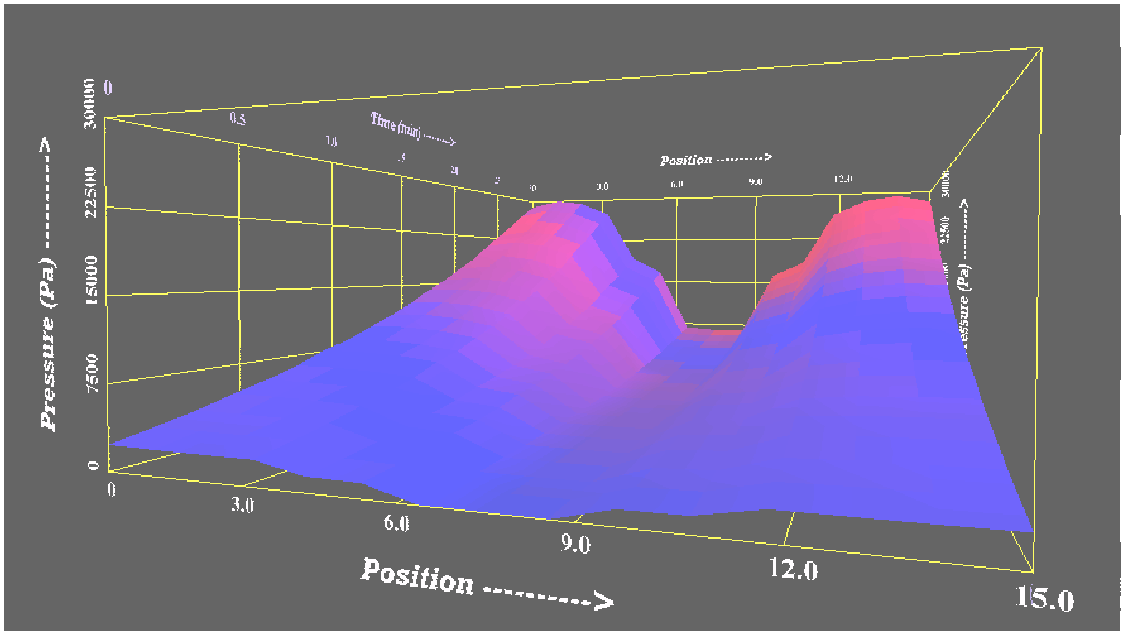


Fig. 2.6. VRML pressure graph showing the variation of pressure with increasing distance from the laminate surface and with time (the receding axis).

accessible from the Web browser. An example is the VRML viewer described in Chapter 2.5.2.

WBCSim currently use three visualization tools: VRML [4], Mathematica [108], and the UNIX utility WhirlGif. With a VRML viewer, the output converted to VRML can be viewed from various directions in the three-dimensional viewspace. Mathematica is only used to generate static two or three-dimensional graphs of simulation results. WhirlGif is a GIF to animation converter, which simply picks up the GIF format frames generated by Mathematica and converts them into an animation file, also in GIF format.

VRML was chosen to interactively visualize some WBCSim simulation output, because VRML is a well-accepted standard for three-dimensional visualization, available on many platforms, easy to use, and easily generated. A custom VRML translator was built to provide better control over the description of the three-dimensional output than third party translators could provide. WBCSim generates VRML code for the RFP model output. In the RFP model, the output data is a matrix of numbers where each row contains two independent variables and the corresponding dependent variable. This three-dimensional data is conveniently modeled as a VRML ElevationGrid. Each row in the matrix can be represented either as a point or a square that is colored based on its height, where blue denotes the lowest point on the grid, red the highest, and the rest are colored by linearly interpolating between blue and red (HSB) hues, as shown in Fig. 2.6.

Mathematica was chosen to generate static two or three-dimensional graphs of WBCSim simulation results, because Mathematica has sophisticated graphics tools, and is commonly

used by wood scientists for graphic representations. When Mathematica was first used in conjunction with WBCSim, the Mathematica commands were hard coded (in Perl) in the wrapper. Every time a simulation was invoked that required Mathematica's services, the wrapper would write the Mathematica commands into a text file embedded with the dynamically generated filenames for the GIF output files (using the simulation ID) . Then, the wrapper would give this file as input to Mathematica, which would generate the GIF files accordingly. However, as more complex simulations such as CMA were added to the system, the Mathematica input file became large and unwieldy. Compounding this size problem was the fact that every time the Mathematica code changed, the Perl wrapper also had to be changed. The current WBCSim approach keeps the Mathematica code as a blueprint external to the wrapper. Then in every simulation run, the wrapper makes a copy of that blueprint code, uses *sed* to insert the dynamic graphics output filenames, and then feeds it directly to the Mathematica kernel. Fig. 2.5 (explained in Chapter 2.6) is an image file generated by Mathematica from the OSB model.

Since some simulation results, such as the CMA model's stress and strain distributions, are time series, WBCSim also provides time animation of those results by using WhirlGif. WhirlGif is a small UNIX utility for generating a single animated GIF image file from a sequence of GIF image files. The reason Mathematica is not used for generating those animation files is that the time to generate an animation in Mathematica is significantly longer than the time by using WhirlGif. Because the number of frames may change from one simulation run to another, the wrapper reads the number of frames from a data file. Then the wrapper constructs a command like "`WhirlGif -loop 1000 frame1.gif frame2.gif frame3.gif frame4.gif` ... > `animation.gif`", where `frame*.gif` are the frame images generated by Mathematica. Then this WhirlGif command is fired off to the UNIX shell by the wrapper to generate the animation file. By embedding each frame into a HTML file, another option is also provided whereby the user can use hyperlinks such as "Next" and "Previous" to view the frames one at a time.

## 2.9 Future Work and Conclusions

Of the PSE characteristics mentioned in Chapter 2.1, experiment management, collaboration support, and high performance computing are the most appealing at the current stage of WBCSim development.

### 2.9.1 Experiment Management

Any given simulation run is just an evaluation of a single point in a multidimensional space, and optimization can search this space for an optimal point with respect to an objective function [86]. While WBCSim supports automated optimization, often it is not possible to precisely articulate an objective function, and there may be multiple conflicting objectives. Furthermore, design trends and data patterns are unlikely to be revealed by

a few isolated runs or by automatic optimization. What is required is that the results of simulation runs be stored automatically in a systematic way, which permits annotating the parameters that define a run as well as its results, and further allows searching, comparison, and other data mining of a data base of numerical experiments. Toward this end, an open source database—Postgres—is currently being integrated with WBCSim. A sophisticated experiment management subsystem should significantly enhance the utility of a PSE like WBCSim.

### 2.9.2 Collaboration Support

Even though the current telnet server and YAPS allow user/guest logins and multiple users at multiple workstations working on WBCSim simultaneously, more sophisticated real-time collaboration support (e.g., Sieve [54]) is needed. This would allow multiple geographically distributed users to jointly modify the same model interface, and concurrently view and annotate the same simulation output. Other features indirectly supporting collaboration are user control over where their data is stored (locally on the client machine vs. remotely on the server), and support for exchanging data with other researchers.

### 2.9.3 High Performance Computing

Simulation codes used by wood scientists often require access to significant computing resources such as a parallel supercomputer. WBCSim is no exception. The more advanced models in WBCSim, such as the HC Fortran 90 core or its visualization/optimization tools, can take hours to run on a fast (AXP 21064) workstation. By utilizing either parallel or distributed computing, WBCSim can reduce the turnaround time to minutes, making interactive use feasible even for complex three-dimensional models. A planned next step is to use a 200 node Beowulf cluster as a compute engine for WBCSim, transparent to the user, when a requested simulation can be estimated to require such resources.

### 2.9.4 Conclusions

WBCSim has evolved steadily from a prototype PSE intended as a tool for computer science PSE research and a Web-based interface to a few legacy computer programs, to a manufacture-oriented near commercial quality PSE seriously used by wood science researchers in industry and academia. Since interesting computational capabilities are still lacking (cf. Chapters 2.9.1–2.9.3), WBCSim will remain an object of computer science research for some time to come. Yet the interfaces, models, and output visualizations are now good enough to be used as production tools by the wood scientists. The directions in which computer scientists would like to take WBCSim (collaboration, data mining, grid computing) are quite different from the directions the wood scientists would prefer for WBCSim (new models, refining existing models, more interface and visualization options). The present layered architecture of WBCSim supports this dichotomy well.

From the user perspective, by providing a Web-based graphical interface to legacy command-line FORTRAN programs, WBCSim leverages the accessibility of the Web to make these simulations easily usable by scientists and engineers away from their laboratories. Via the concept of remote compute cycles WBCSim frees users from software installation on the client machine. The optimization tool and the simulation output visualizations increase the intellectual utility of the legacy simulation codes.

From the developer perspective, the simulations are wrapped and isolated from users. This has the advantage that if there are changes to a legacy simulation program, such as bug fixes, recompilation with new libraries or a new compiler, or implementing a different algorithm, the new program can be installed in WBCSim without additional work. Adding a different visualization tool or a new optimization algorithm is easy since only wrappers need to be modified.

The original stated goal of WBCSim, to provide "an integrated set of facilities allowing wood scientists to concentrate on high-level problem solving rather than on low-level programming details and application scheduling/execution, allowing users to define, record, and modify problems, and visualize and optimize simulation results," now seems eminently achievable.

# Chapter 3: Data Management via a DBMS*

## 3.1 Introduction

Over the past few years, increased attention has been given to the way scientists generate, store, and manage experimental data. Usually, scientists are easily able to generate and store several megabytes of data per experiment. However, they often lack adequate experiment management tools that are not only powerful enough to capture the complexity of the experiments but, at the same time, are natural and intuitive to the non-expert [53]. Therefore, scientists often rely on tagged folders and directory hierarchies to separate and organize experiment data; some scientists have even gone back to the use of paper notebooks to track their data. This situation obviously hinders the productivity of many experimental groups.

In order to address this problem, this chapter describes an experiment management component developed for the WBCSim problem solving environment (PSE). This experiment management component integrates a web-based graphical front end, server scripts, and a database management system to allow scientists to easily save, retrieve, and perform customized operations on experimental data.

WBCSim is a problem solving environment that increases the productivity of wood scientists conducting research on wood-based composite (WBC) materials and manufacturing processes. It integrates Fortran 90 simulation codes with a Web-based graphical front end, an optimization tool, and various visualization tools. The WBCSim project was begun in 1997 with support from USDA, Department of Energy, and Virginia Polytechnic Institute & State University (VPI&SU). It has since been used by students in several wood science classes, by graduate students and faculty, and by researchers at several forest products companies. User feedback has resulted in numerous changes to the interface and underlying models, and was the major impetus for adding experiment management. Replacing the batch file mode of use by the Web interface, and supporting optimization for manufacturing process design, have had a major impact on the productivity of wood scientists using the analysis codes in WBCSim. Goel et al. [41], [42] describe an early version of WBCSim. In 2002, Shu et al. [91] described how WBCSim has further evolved by taking different approaches to its architecture, adding more sophisticated models, and switching from an experiment-oriented to a manufacturer-oriented approach. However, the application's original goals remain the same: (1) to increase the productivity of WBC research and manufacturing groups by improving their software environment, and (2) to continue serving as an example for the design, construction, and evaluation of small-scale problem solving environments.

---

* An experiment management component for the WBCSim problem solving environment, J. Shu, L.T. Watson, N. Ramakrishnan, F.A. Kamke, B.G. Zombori, *Advances in Engineering Software*, 35 (2004) 115–123.

WBCSim primarily serves as an example for the design, construction, and evaluation of small-scale PSEs. In general, a PSE provides an integrated set of high-level facilities that support users engaged in solving problems from a prescribed domain [40]. Also, a PSE commonly addresses the following issues: Internet accessibility to legacy codes, visualization, experiment management, multidisciplinary support, collaboration support, optimization, high performance computing, usage documentation, preservation of expert knowledge, recommender systems, and integration [86]. WBCSim qualifies as a PSE for the following reasons: it makes legacy simulation codes available via the World Wide Web; it is equipped with visualization and optimization tools; it is multidisciplinary; and it will soon be augmented with experiment management, collaboration support, and high performance computing.

Despite its multifunctional ability of running simulation experiments and generating and storing data, WBCSim needs an adequate tool to manage its execution and experiment data, and to provide additional features such as experiment searching, comparison, and other data mining techniques. This chapter describes efforts to develop such an experiment management component for WBCSim. In short, the intent is to implement a database management system (DBMS) to store simulation inputs and outputs at the server end, provide helpful graphical interfaces to facilitate user access at the client side, and then use some scripts to connect all of these various components.

The chapter is organized as follows. Chapter 3.2 reviews some related work in PSEs, WBC computer-based systems, and some experiment management (EM) systems. Chapter 3.3 briefly describes WBCSim. Chapter 3.4 gives the motivation for developing the EM component. Chapter 3.5 elaborates the architecture of WBCSim and how the EM component fits in. Chapter 3.6 demonstrates a typical scenario using this EM component. Chapter 3.7 outlines some future directions for this EM component and draws conclusions.

## 3.2 Related Work

There are a dozen or so problem-specific PSEs developed for various application domains, and several computer-based mathematical models were developed to solve particular problems in the wood-based composites industry. However, no work known to the authors addresses the integration of WBC mathematical models with a problem solving environment. WBCSim is meant to fill this gap and provide a valuable tool for the wood-based composites industry.

Goel et al. described some early PSEs [41]. Since then, several new PSEs have been introduced. Gismo [18], created at Washington University, is an object-oriented Monte Carlo package for modeling all aspects of a satellite's design and performance. It has played a significant role in the design of the Gamma Ray Large Area Space Telescope, the successor to the Compton Gamma Ray Observatory that was launched into space in 1991 to explore the gamma ray portion of the electromagnetic spectrum in astrophysics. VizCraft [39],

developed at VPI&SU, provides a graphical user interface for a widely-used suite of analysis and optimization codes that aid aircraft designers during the conceptual design of high-speed civil transport. VizCraft combines visualization and computation, encouraging the designer to think in terms of the overall problem-solving task, not simply using the visualization to view the computation's results. Also, a computing environment developed by Chen et al. [24], that combines particle systems, rigid-body particle dynamics, computational fluid dynamics, rendering, and visualization techniques to simulate physically realistic, complex dust behavior has been shown to be useful in interactive graphics applications for education, entertainment, or training.

In the wood composites industry, mathematical models may be used to describe the pertinent relationships among various manufacturing parameters and final composite properties. The most influential stages of composites production are the mat formation and the hot-pressing processes; therefore, the modeling effort is concentrated on these two areas.

A mat formation model is required to establish the critical relationships between the structure of the composite and the dynamic change of certain physical properties during mat consolidation. One of the most notable examples of such an approach is the commercially available mat formation software developed by Forintek Canada Corporation. The software, based on Dai's mathematical model [27], incorporates geometric probability theory and simulation techniques for describing characteristics observable on the surface of the flake mat. Another mat formation model, created by Lu [67], called WinMat, works as a stand-alone application in all Windows operating systems. The major objective of WinMat is to generate commands for an industrial robot that is capable of putting together mat structures from uniform size flakes. WBCSim uses a more realistic model of the mat structure developed by Zombori et al. [112], in which mat formation includes the geometry of the wood elements as random variables. In this model, certain limitations can be imposed on the orientation of the elements.

A hot-pressing model describes the heat and mass transfer during the hot-pressing process of wood-based composite panels. The literature is extensive on the modeling of the internal conditions that are present during the hot-pressing stage of wood-based composite manufacture. Zombori et al. [111] surveyed the hot-pressing model literature. However, important modeling efforts to date were either not computer based or did not include an integrated environment.

The topic of experiment management has become very popular in the research community. Workflow management systems (WFMSs) created at the University of Wisconsin [1] use a database management system (DBMS) to store task descriptions and implement all workflow functionality in modules that run on top of the DBMS. The Desktop Experiment Management Environment (DEME), also called Zoo [53], which comes from the same university, emphasizes generic experiment management technology. Zoo has been used by

many domain scientists in fields as diverse as soil sciences to biochemistry, demonstrating that new technology can be continuously transferred among laboratories. At the same time, feedback from installed software can be tested and evaluated in real-life settings that can also affect research directions and decisions. The Site-Specific Systems Simulator for Wireless Communications [101] at VPI&SU uses a subset of XML-based markup languages to support a high performance execution environment, experiment management, and reasoning about model sequences. PYTHIA-II [47] is a modular framework and system that combines a general knowledge discovery in database methodology and recommender system technologies to give advice about scientific software/hardware artifacts. It provides all of the facilities needed to set up database schemas for testing suites and associated performance data in order to test suites of software.

## 3.3 WBCSim

WBCSim is intended to increase the productivity of wood scientists conducting research on wood-based composite materials by making legacy file-based FORTRAN programs that solve scientific problems in the wood-based composites domain widely accessible and easy to use. WBCSim currently provides Internet access to command-line driven simulations developed by the Wood-Based Composites (WBC) Center at VPI&SU. WBCSim leverages the accessibility of the World Wide Web to make simulations with legacy code available to scientists and engineers away from their laboratories.

WBCSim currently supports five simulation models that help wood scientists studying wood-based composite material manufacturing.

(1) Rotary dryer simulation (RDS). The rotary dryer simulation model assists in the design and operation of the most common type of system used for the drying of wood particles [58], [59].

(2) Radio-frequency pressing (RFP). The radio-frequency pressing model [83] was developed to simulate the consolidation of wood veneer into a laminated composite using high frequency energy.

(3) Oriented strand board mat formation (OSB). The mat formation model [91] creates a three-dimensional spatial structure of a layered wood-based composite (e.g., oriented strand board and waferboard); it also calculates certain mat properties by superimposing a mesh on the mat structure.

(4) Hot compression (HC). The hot compression model [91] simulates the mat consolidation and adhesive cure that occurs during industrial hot-pressing of wood-based panels.

(5) Composite material analysis (CMA). The composite material analysis model [91] was developed to assess the stress and strain behavior and strength properties of laminated materials (e.g., plywood and fiber-reinforced composites).

The current software architecture of WBCSim follows the three-tier model described by Goel et al. [41], in which the tiers are (1) the client layer—user interface, (2) the server layer—a telnet server and a custom shell, and (3) the developer layer—legacy simulation codes and various optimization and visualization tools running on the server. Chapter 3.5 elaborates the details of the three layers while explaining how the experiment management component fits into this architecture.

WBCSim is equipped with an optimization algorithm DOT (Design Optimization Tool) [96] and various visualization tools: VRML [4], Mathematica [108], and the UNIX utility WhirlGif. The reader is referred to [91] for an in-depth treatment of these tools.

## 3.4 Rationale for Experiment Management

Before the experiment management component was added, WBCSim had a file-based save and retrieve system that operated as follows. Upon saving, the input parameters were packed into a file with a filename identified by the current run ID. This file, along with the possible output files (data files generated by the FORTRAN code, pictures generated by the Mathematica or VRML translator, and HTML files generated by the scripts), was stored in a permanent directory (depending on the login account). An optional description of the simulation was also stored. Upon retrieving the simulation, a user chose to either load the input parameters in the proper model interface or load the stored simulation results.

This approach is not sufficient as WBCSim evolves from being experiment-oriented to being manufacturer-oriented. Manufacturers like to see how the product properties could be optimized while the profit is maximized. However, any given simulation run is just an evaluation of a single point in a multidimensional space, and optimization can only search this space for an optimal point with respect to an objective function [86]. While WBCSim supports automated optimization, often it is not possible to precisely articulate an objective function, and there may be multiple conflicting objectives. Furthermore, design trends and data patterns are unlikely to be revealed by a few isolated runs or by automatic optimization. What is required is that the results of simulation runs be stored automatically in a systematic way. This approach would permit annotating the parameters that define a run as well as its results, and would further allow searching, comparison, and other data mining of a database of numerical experiments.

Toward this end, a special tailored experiment management component is being added to WBCSim, which consists of customized user interfaces, server scripts, and an open source database management system (DBMS)—Postgres. The details of this component are explained in Chapter 3.5—WBCSim EM Architecture. This experiment management component not only supports all of the previous features from the file-based save and retrieve system, it also significantly increases WBCSim user productivity and usability in the following ways.

(1) The new user interface allows a user to filter the stored simulation runs by their description or date.

(2) The new user interface provides customized comparison functions that can compare multiple simulation runs.

(3) The server script searches the database for each simulation run. If the simulation has been run before and saved, the script retrieves the simulation results from the database rather than run the FORTRAN program, which normally requires a longer execution time.

(4) The database stores simulation inputs and outputs. Only distinct inputs and outputs are saved. In case there is a duplicate simulation run (with a distinctive annotation), its inputs and outputs are not stored but will point to those from the previously stored simulation run.

## 3.5 WBCSim EM Architecture

The current software architecture of WBCSim follows the three-tier model described by Goel et al. [41], in which the tiers are (1) the client layer—user interface, (2) the server layer—telnet server and custom shell, and (3) the developer layer—legacy simulation codes and various optimization and visualization tools running on the server. These layers are shown in Fig. 3.1. Here, only the elements related to the experiment management component are discussed. The reader is referred to [91] for further details.

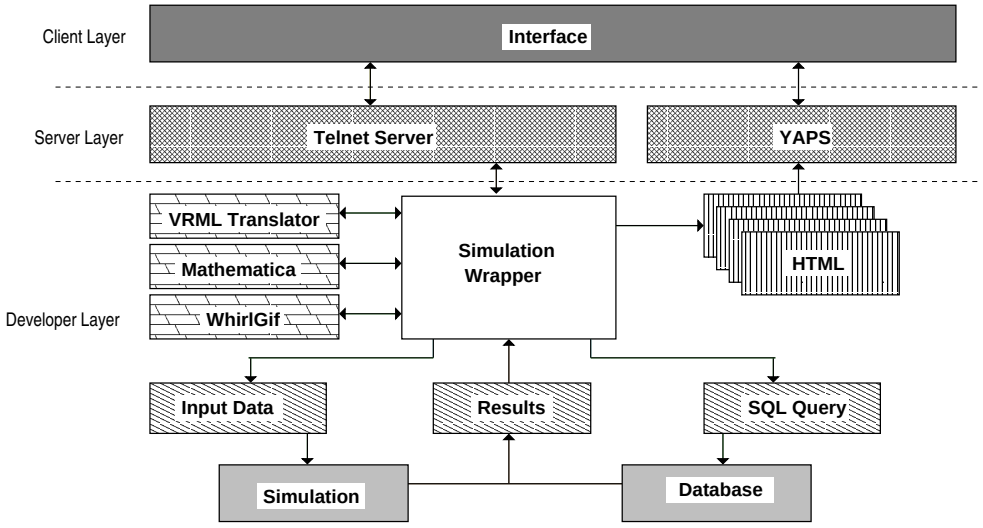


Fig. 3.1. WBCSim architecture overview.

### 3.5.1 Client Layer

The client layer is the only layer visible to end-users and typically the only layer running on the local machine. The main part of the client layer is the user interface.

WBCSim is Web-based; therefore, its user interface is composed of a Web browser and Java applets. A user launches the WBCSim Web page from a browser window, and then invokes applets from the Web page. The very first applet allows the selection of a simulation model. From this stage, all user interaction with the system is via applets. The user interface varies depending on which model is invoked. However, all models have a row of buttons at the very bottom of the interface (shown in Fig. 3.2) that control model-level actions for a simulation. Among those buttons are two, "Store Problem" and "Retrieve Problem", which are related to the experiment management component. When the "Store Problem" button is triggered, the current set of input values (along with the simulation description, which can be specified in the long text box near the bottom of the interface) is sent to the server and stored. The interface will be updated with a run ID at the upper right-hand corner to indicate that the save occurred. The "Retrieve Problem" button is used to retrieve stored simulation runs.

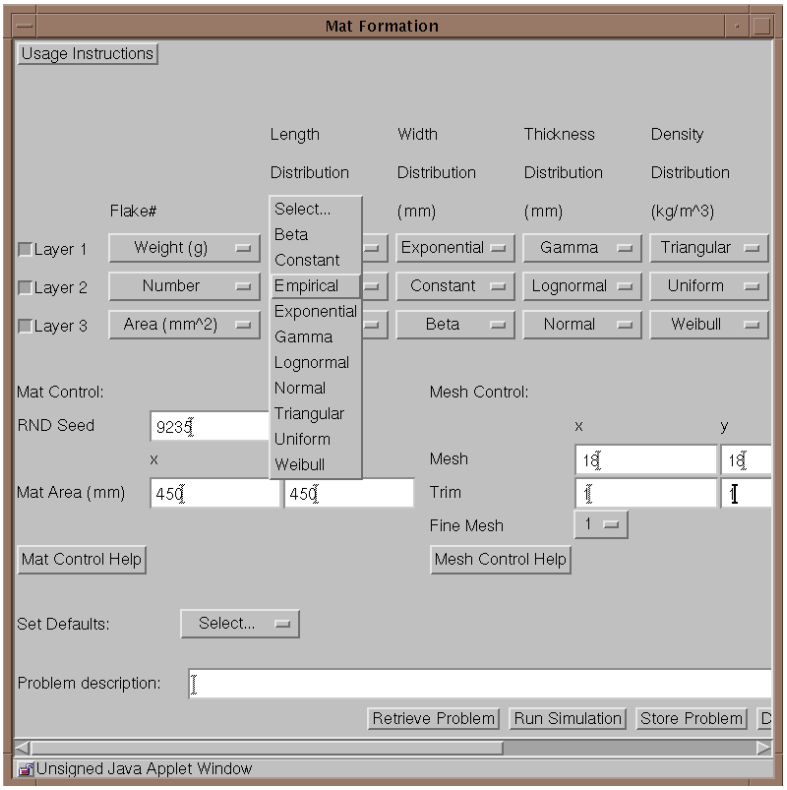


Fig. 3.2. The OSB model user interface.

Upon activation of the "Retrieve Problem" button, a window (shown in Fig. 3.3) will open with a list of all the simulation runs for this particular model stored in the database. This window is the portal for a user to access the stored simulation runs. At the top of this window, there is some text describing the use of this window. There are also a set of dropdown lists, and text fields and buttons, which allow the user to filter the simulation runs displayed. By selecting either "Description" or "Date" from the "Options" dropdown list, the user can narrow the filter process to either field. Then the user can input a regular expression (in UNIX grep style) in the text field, and press the "Filter" button to execute the filter process, which will reload this window with the simulation runs that meet the filter criteria. The next portion of the window is a scrollbar panel with all of the simulation runs listed. Each simulation takes a row, which has a "Show Input" button, "Show Output" button, a check box, the date that it was stored, and its description. The "Show Input" button launches the model interface with the set of stored input values. The user can make changes to those parameters, and run the simulation again. Clicking on the "Show Results" button opens another window with buttons that point to the simulation results available in both textual (normally tables of numbers) and graphical (VRML files, GIF files) forms, displayable in browser windows. The checkbox allows the user to either delete the simulation run from the database (by clicking the "Remove" button at the very bottom of the window), or apply the comparison functions (by selecting an item from the "Compare Marked Runs" dropdown list). Each model of WBCSim has its own comparison functions. For example, the OSB model has three, which allow the user to compare "Void Fractions/Contact Area", "Density", and "Coefficient of Variation" among the simulation runs marked.

If "Void Fractions/Contact Area" is selected from the "Compare Marked Runs" dropdown list, a window (shown in Fig. 3.4) opens with more options that allow comparisons among "space/mat", "lumen/mat", "void/mat", "lumen/flake", and "contact area". The user can also specify the interval of the Y axis in order to narrow the comparison.

Depending on the choices made in the detail comparison window, the user gets a result window, which has links pointing to the results that can be displayed in a browser window.

The user interfaces described above are solely for the experiment-management component. The interfaces for the models are described in [91]. The client layer also contains viewers for one of the visualization tools: the VRML translator. WBCSim requires a VRML 2.0 viewer for the RFP model. The VRML viewer functions as a plug-in for the Web browsers.

The client layer also handles communication with the server layer. After the user triggers the "Store Problem" button, the client sends those parameters via a telnet connection along with a request for storing them to the server layer via a telnet connection. The "Retrieve Problem" and "Compare" buttons act similarly.

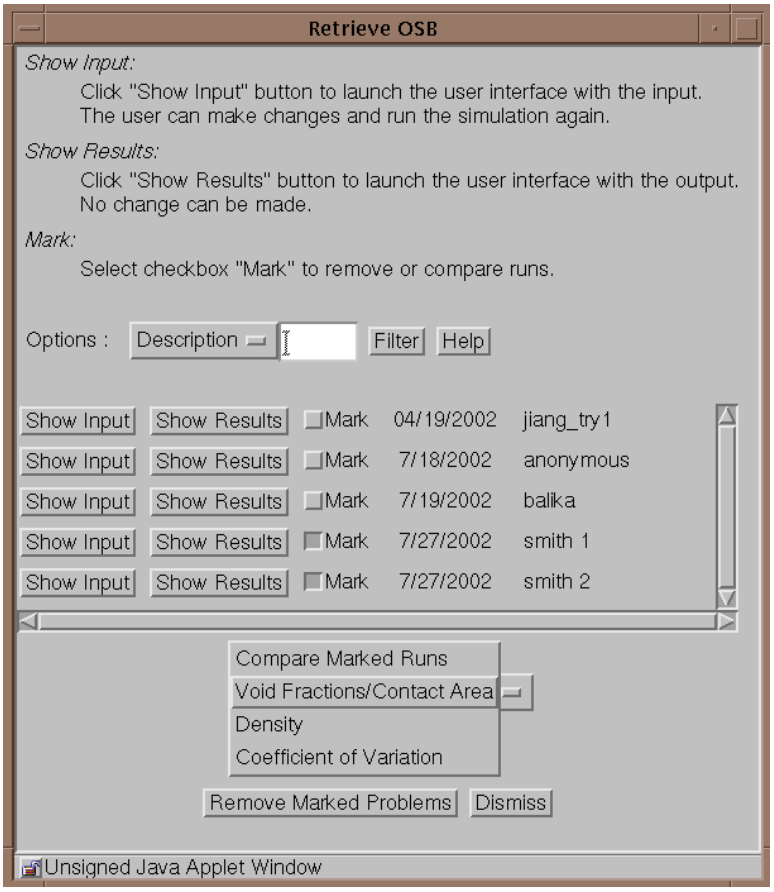


Fig. 3.3. The OSB retrieval user interface.

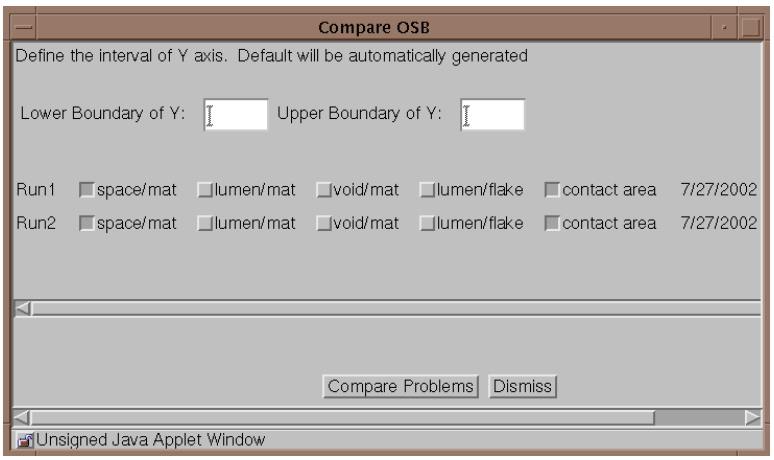


Fig. 3.4. The OSB Void Fractions/Contact Area comparison detail user interface.

### 3.5.2 Server Layer

By separating the legacy simulation codes from the user interface, the server layer functions as the key to how WBCSim can run a text-only application from a Web browser.

The server layer consists of two components: a telnet server and a custom shell to facilitate server-client communication.

The telnet server is not a replacement for a standard telnet server; it implements only enough of the telnet protocol to work with the WBCSim telnet client. The telnet server supports guest and regular logins and all of the operations provided by the previous Javamatic server, which could direct execution of multiple simulations and concurrently accept multiple requests from the client [41].

Yet Another PSE Shell (YAPS) is a simple Perl script that the client invokes when it logs in via a telnet connection. The client talks to this shell instead of the UNIX login shell of the account. The common commands supported by YAPS are described in [91].

### 3.5.3 Developer Layer

As its name suggests, the developer layer consists of legacy programs created by researchers to model wood-based composite materials and manufacturing processes. These legacy programs are the heart of WBCSim. In general, WBCSim supports legacy programs written in any programming language as long as the program takes its input parameters from UNIX stdin. The input parameters can be a few numbers or a large data file. In particular, WBCSim supports five FORTRAN 77 and Fortran 90 simulation programs, corresponding to the five models described in Chapter 3.3.

While each Fortran program has its own input format (stdin could be redirected to a file), the server layer communicates data with the developer layer via strings of parameters separated by white space (spaces, tabs, newlines). In order to cope with this string format, each legacy program is "wrapped" with a customized Perl script. The script receives this string of parameters from the server, and converts those parameters into an appropriate format for the Fortran program. Then the script calls the legacy program into action, feeding it the input, invoking any required optimization and visualization tools, packing all Fortran output in HTML files, and passing their URLs, first to the server layer, and then to the client layer.

With the addition of the experiment management component, the wrapper performs one more operation before it calls the Fortran program to execute the simulation. The wrapper checks if a simulation with the current set of inputs has already been executed. If the simulation was executed before and the results were saved, the wrapper constructs the output data files from the database instead of running the Fortran program. The wrapper then feeds the output data files into any required optimization and visualization tools. This operation is transparent to the user; it can significantly reduce the response run time if the output is found in the database.

There are other wrappers associated with the save, retrieve, and compare functions from the user interface at the client layer. Like the simulation wrapper, these wrappers receive

data from the server layer, and convert those input values into an appropriate format for the database. Moreover, they also perform some extra checks in order to optimize database usage and storage. For example, the save wrapper checks if the current set of values is already saved. If that is the case, only a pointer to that set of values is saved. If the current set of the input that the user is saving is matched with the previous stored inputs whose output was not saved, the wrapper automatically updates those previously stored inputs in order to point to the current set of output values.

Postgres is the database system being used. Postgres is a sophisticated object-relational database management system (DBMS) that supports almost all SQL constructs, including subselects, transactions, and user-defined types and functions. Moreover, it is open source. In this architecture, the wrappers communicate with Postgres by Pgsql-perl5, which is an interface between Perl version 5 and Postgres. This interface uses the Perl version 5 application programming interface for C extensions, which calls the Postgres programmer's interface LIBPQ.

The developer layer also includes optimization and visualization tools to maximize the simulation's value to the user [91].

## 3.6 Scenario

Describing a typical usage scenario of WBCSim is instructive. Consider research into the properties of oriented strand board products, which would use the OSB model. First, a scientist samples the face and core layer flakes from an industrial production line. The scientist measures the geometry and the weight of each flake. The flake property data sets are then used to estimate the statistical probability density functions of the flake properties. Then, the scientist opens a browser window and launches WBCSim. The guest account is the default login option, which stores all the input and output data in a temporary directory that is purged periodically. The scientist selects the OSB model from a dropdown list of all the available models, and launches the OSB model interface (shown in Fig. 3.2). By specifying the probability density functions of the flake properties obtained offline earlier, the scientist can create a statistically valid model instance of the spatial structure of multiple layers of flakes, each with various parameters defining the number of flakes, length distribution, width distribution, thickness distribution, density distribution, orientation, flake color, and number of flakes shown. The latter two are solely for visualization.

Next, the input parameters (Boolean, numeric, or alphanumeric) are sent to the WBCSim server as a long string (parameters are separated by white space) via a telnet connection. Then, the OSB wrapper converts this string into a SQL query, and checks if this set of input parameters is stored in the database.

If the output does not exist in the database, the OSB wrapper converts this input string into a data file designated for the OSB Fortran 90 simulation program. Since the OSB

simulation code has its own text-based user interface taking input from stdin, a temporary file is generated to contain all of the appropriate commands. Stdin is then redirected to this temporary data file for the simulation. The OSB wrapper then calls the OSB simulation code with this temporary file as stdin along with the properly formatted data file. When the simulation code is executing, the OSB wrapper monitors the simulation output stream for strings indicating execution milestones. The OSB wrapper uses the standard error stream for sending these messages to the client, because Java buffers the standard output stream until the process terminates. The contents of the standard error stream are sent immediately. The OSB wrapper also sends other messages (such as when visualization or optimization tools are invoked) to the client, displaying the simulation status for the scientist. However, due to network delay, a group of messages generated at different times can arrive at the client at the same time, in which case the client only displays the latest message and discards any old ones.

If the output exists in the database, the OSB wrapper extracts the outputs from the database and places them into data files in the same format as they would have been written by the OSB Fortran 90 simulation code.

When those output data files are ready (no matter whether they come from the Fortran 90 simulation program or the database), the OSB wrapper calls Mathematica to read those data files and generate plots with various Mathematica commands such as ListContourPlot, ListPlot3D, MultipleListPlot, and Graphics3D. Mathematica also converts these internal graphics data structures into GIF format so that they can be viewed in a browser. Fig. 3.5 shows a three-dimensional visualization of a three-layer random flake mat created with Graphics3D by Mathematica. Finally, the OSB wrapper embeds these GIF files in HTML files and returns the URLs of these HTML files to the client.

Upon closing the interface windows, the scientist has a choice of storing this particular run's input values and results, or discarding this run (the default). If the scientist chooses to store this run, he enters a description such as "Smith 2" and presses the "Store Problem" button. The entire interface is then updated with the run ID to indicate that the save occurred. To further compare this run with any previous runs, the scientist can click the "Retrieve Problem" button to launch the retrieve problem interface (shown in Fig. 3.3). The scientist can use the filter (described in Chapter 3.5) to narrow the list of simulation runs displayed in the scrollbar panel. By marking two simulation runs, for example, "Smith 2" and "Smith 1", the scientist can compare their "Void Fractions/Contact Area" by selecting that item from the "Compare Marked Runs" dropdown list. The comparison detail window (shown in Fig. 3.4) allows the scientist to choose among "space/mat", "lumen/mat", "void/mat", "lumen/flake", and "contact area" to view the comparison result. Fig. 3.6 shows the comparison result created with MultipleListPlot by Mathematica. The scientist

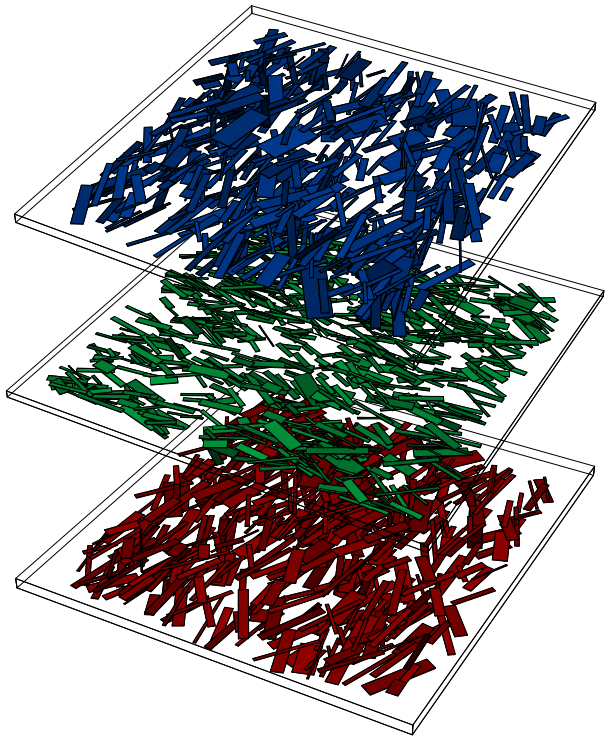

Fig. 3.5. Visualization of the three-layer random flake mat: the area of the mat is $450 \times 450$ mm. Only a fraction of the flakes are shown, and the shading designates the density of the flakes.

can go back to the comparison detail window (shown in Fig. 3.4) to make other selections or change the Y axis interval to get different results or views.

## 3.7 Future Work and Conclusions

Shu et al. [91] describe the planned addition of experiment management (EM), collaboration support, and high performance computing, which are most appealing given the current state of WBCSim development. While WBCSim is being developed on several different fronts, the development track for the EM component intends to support all simulation models, be more DBMS centric, and be general enough for use in other PSE projects.

### 3.7.1 Support All Models

Currently, this EM component is only available for two models, RDS and OSB, among the five models described in Chapter 3.3. RDS was used as the first application of this EM component, since RDS has the smallest number of input and output parameters. Application of the EM component to the OSB model has proved that it can handle the complex OSB model as well. EM support for the other three models (RFP, HC, and CMA) is in progress. Each model will have its own customized comparison functions.

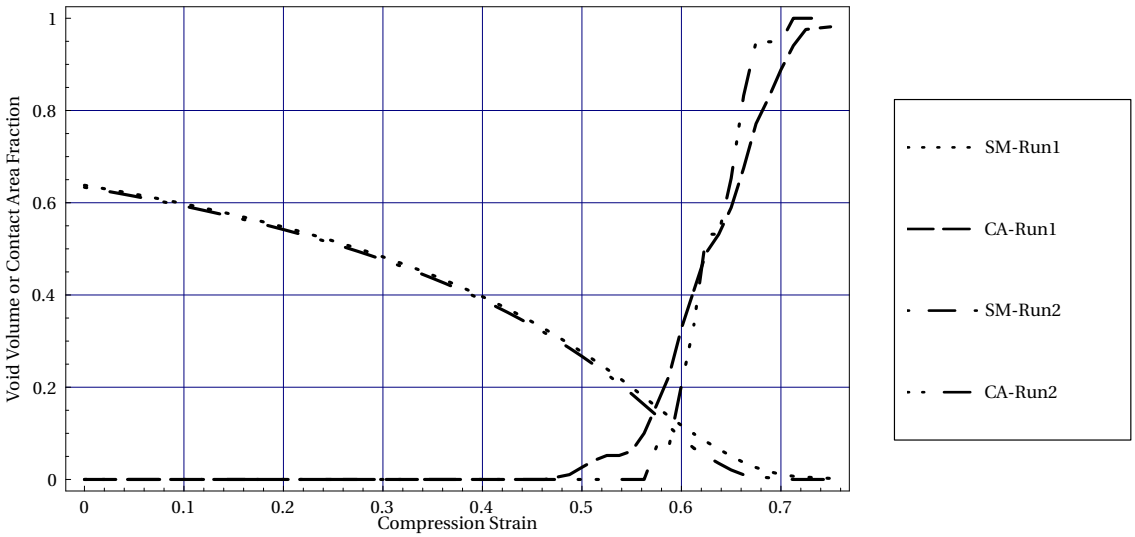


Fig. 3.6. The OSB Void Fraction/Contact Area comparison result from two simulation runs. It shows "space/mat" (SM) volume and "contact area" (CA) from those two simulation runs.

### 3.7.2 More DBMS Centric Architecture

Chapter 3.5 describes the current architecture of WBCSim along with the EM component's role. However, a more database management system-centric architecture (DBMSCA) could provide much more scientific workflow management system functionality. The DBMSCA can define a workflow for the conduct of a WBCSim simulation: preparing the input file, validating the input, invoking the simulation code or submitting a job, formatting files, monitoring the status of a job, and retrieving results. The DBMSCA can then define a schema for the workflow, use a workflow management system that summarizes, keeps track of the status of different jobs, determines what is to be done, triggers upon abnormal conditions (e.g., divide by zero, file not found), etc. A (Postgres) user-defined function looks for records specified in the query. If these records exist, it gets them from the database. If they do not, it "places" some new records into a table called "requests". The workflow manager periodically scans the "requests" table and schedules jobs. When experiments are finished, the manager places appropriate entries in the output tables. This idea has been explored in the $S^4W$ PSE for wireless system design [99].

### 3.7.3 Generalization

The EM component is specially designed for WBCSim. At Virginia Polytechnic Institute and State University, there are currently PSE groups working on cell cycle modeling, wireless communication systems, aircraft design, microarray experiments, and land use change analysis. The usefulness of the EM component for other PSE projects is currently being explored.

### 3.7.4 Conclusions

WBCSim has evolved steadily from a prototype PSE intended as a tool for computer science PSE research and a Web-based interface for a few legacy computer programs, to a manufacture-oriented near commercial quality PSE that is seriously used by wood science researchers in industry and academia. Since interesting computational capabilities are still lacking [91], WBCSim will remain an object of computer science research for some time to come. Yet the program's interfaces, models, and output visualizations are now good enough to be used as production tools by wood scientists. The directions in which computer scientists would like to take WBCSim (collaboration, data mining, grid computing) are quite different from the directions that wood scientists would prefer for WBCSim (new models, refining existing models, more interface and visualization options). The present layered architecture of WBCSim supports these divergent development directions well.

With the addition of the EM component, WBCSim now has a much more efficient tool to manage its simulation execution and experiment data. From the user perspective, the EM component allows a user to easily save, retrieve, compare simulation runs, and further investigate the inter-relationships among multiple parameters. From the developer perspective, the simulation runs are stored in a more systematic manner—a database that permits further annotation and data mining possibilities. Adding a different compare function (comparison between simulation outputs) is easy, since only (high level script) wrappers need to be modified.

The original stated goal [41] of WBCSim was to provide "an integrated set of facilities allowing wood scientists to concentrate on high-level problem solving rather than on low-level programming details and application scheduling/execution, allowing users to define, record, and modify problems, and visualize and optimize simulation results". This now seems much closer with the addition of an experiment management capability.

# Chapter 4: Change Management via XML*

## 4.1 Introduction

A problem solving environment (PSE) is a computational system that provides a complete and convenient set of high-level tools for solving problems from a specific domain [84]. A PSE commonly addresses many issues: Internet accessibility to legacy codes, visualization, experiment management, multidisciplinary support, recommender systems, collaboration support, optimization, high performance computing, preservation of expert knowledge, design extensibility, and pedagogical uses [105].

A PSE often has many development cycles. Features addressing the above issues are added step by step depending on various interests/feedback, priorities, and/or the dynamic nature of the problem. During the process of adding more features, much information could be duplicated at various implementation layers. This situation obviously makes the system hard to maintain and change, and hinders the productivity of developers and scientists.

This chapter addresses this problem of information duplication by describing how a design approach based on specifications via XML can be used to unify the implementation layers for the problem solving environment WBCSim. A XML datasheet is tailored for each model (WBCSim has five models). The WBCSim interface layer, server scripts, and database management system all use this XML datasheet to improve the usability and maintainability of the three layers—client, server, developer—comprising the WBCSim system. XML is the acronym for Extensible Markup Language, which allows a developer to create customized tags, supporting the definition, transmission, validation, and interpretation of data between applications and between organizations. The contribution of this chapter is a design approach that permits changes in one place (a simulation model) to be automatically propagated to other places in the PSE system, thereby facilitating customization and maintenance. This could have been done with any domain-specific language that is easily parsed, validated, and composed, but XML was chosen because of the plethora of available tools supporting XML. The design approach described here is generic, and as such generalizes to any other PSE implemented in layers like WBCSim. The propagation tools themselves are, of necessity, specific to the simulation models and interfaces in WBCSim. The layered architecture of WBCSim and the design approach using XML-based specifications are valuable ideas for the construction of future problem solving environments.

WBCSim is a problem solving environment that increases the productivity of wood scientists conducting research on wood-based composite (WBC) materials and manufacturing processes. It integrates Fortran 90 simulation codes with a Web-based graphical front end, an

* Unification of problem solving environment implementation layers with XML, J. Shu, L.T. Watson, N. Ramakrishnan, F.A. Kamke, C. North, *Advances in Engineering Software*, 39 (2008) 189–201.

optimization tool, and various visualization tools. The WBCSim project was begun in 1997 with support from USDA, Department of Energy, and Virginia Polytechnic Institute & State University (VPI&SU). It has since been used by students in several wood science classes, by graduate students and faculty, and by researchers at several forest products companies. User feedback has resulted in numerous changes to the interface and underlying models, and was the major impetus for using XML as a mechanism for unification. Replacing the batch file mode of use by the Web interface and supporting optimization for manufacturing process design have had a major impact on the productivity of wood scientists using the analysis codes in WBCSim.

Goel et al. [41], [42] described an early version of WBCSim. In 2002, Shu et al. [91] described how WBCSim had further evolved by taking different approaches to its architecture, adding more sophisticated models, and switching from an experiment-oriented to a manufacturer-oriented approach. In 2004, Shu et al. [90] described a WBCSim experimental management component, which integrates a Web-based graphical front end, server scripts, and a database management system to allow scientists to easily save, retrieve, and perform customized operations on experimental data. However, the application's original goals remain the same: (1) to increase the productivity of WBC research and manufacturing groups by improving their software environment, and (2) to continue serving as an example for the design, construction, and evaluation of small-scale problem solving environments.

WBCSim primarily serves as a test bed for the design, construction, and evaluation of small-scale PSEs. WBCSim qualifies as a PSE for the following reasons: it makes legacy simulation codes available via the World Wide Web; it is equipped with visualization and optimization tools; it is multidisciplinary; it will soon be augmented with experiment management, collaboration support, and high performance computing.

Concomitant with its multifunctional ability of running simulation experiments and generating and storing data, WBCSim is quite difficult to modify and maintain. Upon making a change such as adding an input parameter to a model, a developer needs to change the proper interface code, wrapper code, visualization code, and database schema. This series of changes can sometimes be tedious and tricky in order to maintain the integrity of the whole system. This chapter describes efforts to use XML to resolve this problem and reduce redundancy. Succinctly, all implementation layers access the same XML datasheet for a particular model. A simple change in the XML can cause cascading changes at various implementation layers, thereby significantly simplifying the development effort and improving maintainability.

The chapter is organized as follows. Chapter 4.2 reviews some related work in PSEs and XML usage in PSEs. Chapter 4.3 briefly describes WBCSim. Chapter 4.4 gives the motivation for applying XML in WBCSim. Chapter 4.5 elaborates on the WBCSim user interface, the model interface similarities, and also the unique aspects of each. Chapter 4.6

explains the architecture of WBCSim and how the XML fits in. Chapter 4.7 demonstrates typical scenarios of using and changing WBCSim. Chapter 4.8 outlines some future directions for WBCSim and draws conclusions.

## 4.2 Related Work

A PSE is a relatively new computing paradigm that was first introduced in problem domains such as partial differential equations (ELLPACK [41] and its descendents [8] for solving two- and three-dimensional elliptic partial differential equations) and linear algebra (Linear System Analyzer [14] for manipulating and solving large-scale sparse linear systems of equations). There is also SCIRun [77], developed to interactively compose, execute, and control (computational steering) large-scale scientific computations.

Since then, many new PSEs have been introduced. Gismo [18], created at Washington University, is an object oriented Monte Carlo package for modeling all aspects of a satellite's design and performance. It has played a significant role in the design of the Gamma Ray Large Area Space Telescope, the successor to the Compton Gamma Ray Observatory that was launched into space in 1991 to explore the gamma ray portion of the electromagnetic spectrum for astrophysics research. VizCraft [39], developed at Virginia Polytechnic Institute and State University, provides a graphical user interface for a widely used suite of analysis and optimization codes that aid aircraft designers during the conceptual design of a high speed civil transport. VizCraft combines visualization and computation, encouraging the designer to think in terms of the overall problem solving task, not simply using the visualization to view the computational results. Also, a computing environment developed by Chen et al. [24], which combines particle systems, rigid body particle dynamics, computational fluid dynamics, rendering, and visualization techniques to simulate physically realistic, complex dust behavior, has been shown to be useful in interactive graphics applications for education, entertainment, or training.

Watson et al. [105] present a thorough summary of the key attributes of a PSE (described in Chapter 4.1), and also compare a PSE with other similar computing environments—a decision support system (DSS) and a geographic information system (GIS). Roughly, a DSS emphasizes the functionality (analysis, planning, and decision making), a GIS emphasizes the nature of data and information (spatial), while a PSE emphasizes the problem domain (such as wood-based composite materials).

There are many problem specific PSEs developed for various application domains. Some of those PSEs utilize XML specifications for some components of the PSE. The XML technology is often applied in PSEs in two ways: 1) describing entities such as models, components, and services, and 2) describing relationships between entities such as calls, messages, and connectivity. Often XML is used in conjunction with other technologies such as SOAP, CORBA, and Java.

Rana et al. [80] describe a PSE for scientific computations and large-scale simulations. This environment is based on the Java programming language, with components being Java or CORBA objects defined with XML interfaces. Youn et al. [110] describe Application Web Service Toolkits, which consist of a set of core services built using the Web services model and application metadata services. A PSE can be built from these core services, which also support multiple versions of services. XML is used to describe, access, and discover those core services in a distributed computing paradigm. PROTEUS [19] is a grid based PSE for composing, compiling, and running bioinformatics applications on the grid. One of the main components—metadata repository—uses XML documents to represent installed software components and heterogeneous data sources. Beca [6] focuses on the collaboration aspect of a PSE by using a technology called Shared Place, which allows construction of virtual places on the Web by merging collaborative tools with Web page content. Shared Place Definition Language (SPDL), an XML based language, describes properties of a virtual place: Place appearance, behavior of collaboration components present in the Place, and access to the Place resources.

BSML [100] is a markup language proposed to formally specify data interchange assumptions underlying scientific codes. It addresses a problem similar to that studied in this chapter but focused on a PSE for wireless system design [93]. In contrast to the tasks considered here, BSML is aimed at validation (ensuring that data is available in the appropriate format to execute the model), binding (associating a data stream with a computational model), and conversion (suitably massaging data if format conversions are required). There are numerous other efforts, primarily in the area of grid computing (see, e.g., [79]), aimed at using XML to declaratively specify problem solving tasks, scheduling runs, and managing scientific data.

The use of XML datasheets here to capture model descriptions and invocation formats is similar in goals to the APIs standardized in the TeraGrid Science Gateway [29] and OGSA (Open Grid Services Architecture) [34] projects. Whereas the APIs in these projects are designed for inter-operability between third party codes (e.g., a molecular dynamics simulation can be connected to a profiling tool or a visualizer), the specifications here are designed to ease the addition of new models in WBCSim; hence the primary consideration here is to avoid redundancy among the many implementation layers in the PSE and improve maintainability. This enables a newly developed model to be incorporated and accessible by the end users. As the scope of wood-based composite simulation in WBCSim grows, it is anticipated that the model specifications would be wrapped into remote invocation APIs suitable for inclusion in large, grid computing environments.

## 4.3 WBCSim

WBCSim is intended to increase the productivity of wood scientists conducting research on wood-based composite materials by making legacy file based FORTRAN programs that

solve scientific problems in the wood-based composites domain widely accessible and easy to use. WBCSim currently provides Internet access to command line driven simulations developed by the Wood-Based Composites Center at VPI&SU. WBCSim leverages the accessibility of the World Wide Web to make simulations with legacy code available to scientists and engineers away from their laboratories.

WBCSim currently supports five simulation models that help wood scientists studying wood-based composite material manufacturing.

(1) Rotary dryer simulation (RDS). The rotary dryer simulation model assists in the design and operation of the most common type of system used for the drying of wood particles [58], [59].

(2) Radio-frequency pressing (RFP). The radio-frequency pressing model [83] was developed to simulate the consolidation of wood veneer into a laminated composite using high frequency energy.

(3) Oriented strand board mat formation (OSB). The mat formation model [91] creates a three-dimensional spatial structure of a layered wood-based composite (e.g., oriented strand board and waferboard); it also calculates certain mat properties by superimposing a mesh on the mat structure.

(4) Hot compression (HC). The hot compression model [91] simulates the mat consolidation and adhesive cure that occurs during industrial hot-pressing of wood-based panels.

(5) Composite material analysis (CMA). The composite material analysis model [91] was developed to assess the stress and strain behavior and strength properties of laminated materials (e.g., plywood and fiber-reinforced composites).

The current software architecture of WBCSim follows the three tier model described by Goel et al. [41], in which the tiers are (1) the client layer—user interface, (2) the server layer—Web server and a PHP module, and (3) the developer layer—legacy simulation codes and various optimization and visualization tools running on the server. Chapter 4.6 elaborates the details of the three layers while explaining how the XML technology fits into this architecture.

WBCSim is equipped with an optimization algorithm DOT (Design Optimization Tool) [96] and various visualization tools: VRML [4], Mathematica [108], and the UNIX utility WhirlGif. The reader is referred to [91] for an in-depth treatment of these tools.

## 4.4 Rationale for Applying XML Technology

Before the introduction of XML technology into WBCSim, the WBCSim Web interface (originally in Java, now in PHP) was hard-coded. At the beginning of each model interface code, there are sections to declare and initialize the model input parameters, which are sequentially used to construct the user interface, and store the input values. The simulation wrappers (in Perl) are used to envelop the FORTRAN simulations and connect various

tools (such as the database, visualization tools, and optimization tools). These wrappers also have sections in their code for the input parameters, output parameters, and other corresponding scripts involved (such as Mathematica scripts for graphics). Moreover, the database (Postgresql) also has database table schemas for each model to define its input and output parameters and files.

This architecture results from adding features step by step. A typical PSE has many development cycles in which different features are added at different times depending on developer interest, user feedback, priorities, and the dynamic nature of the problem. WBCSim, for example, had its interface rewritten from Java into PHP; optimization tools, visualization tools, and the database were all added at different times and by different developers. The problem is that much information (such as input parameters) is duplicated at various implementation layers. When making a change at the interface layer, such as adding an input parameter to a model, a developer needs to locate the proper code to define the information about the parameter (such as name, physical units, minimum, maximum, and default), and then pinpoint the place to add the parameter (normally in a HTML table). Then, the corresponding simulation wrappers and the database table schemas need to be carefully changed as well, in order to maintain the integrity of the entire system. Therefore, it is very hard for a developer to change and maintain such a system, especially when the developer needs to communicate often with wood scientists requesting changes. Misunderstanding and wasted programming effort can occur, making the process even more inefficient.

What is required is that the information about the model be stored at one place so that all the components at various implementation layers can easily access such information. This approach would permit changes at one place to cause cascading changes in the entire system. Toward this end, a XML datasheet is tailored for each model in WBCSim. The WBCSim interface, server scripts, and database management system all use this XML datasheet to improve the usability and maintainability of the three layers—client, server, and developer—comprising the WBCSim system. The details of XML usage are explained in Chapter 4.6—WBCSim Architecture. This approach significantly simplifies development time and improves maintainability in the following ways.

(1) The information about a model is stored in its XML datasheet. All components from various implementation layers refer to this same datasheet. This eliminates any duplicate information.

(2) Changing the XML datasheet will cause cascading effects, which will alter the interface, wrappers, and database accordingly.

(3) The XML datasheet is flexible and simple enough to allow wood scientists to easily define and directly modify a model by themselves.

(4) The XML datasheet can be verified via its XML Schema (a standard to describe and validate data in an XML environment) to ensure its integrity and (syntactic) correctness.

(5) A set of XML datasheets can be labeled as one version of the system to support disaster recovery. A scientist could recover an old model if a newer process model is lost (from hardware crashes).

## 4.5 User Interface

In 2002, Shu et al. [91] described a Java applet interface for WBCSim. That Java applet interface (Browser plug-in technology) has since been replaced by a PHP (server scripting technology) interface. Java applets require a user to have a Java-enabled browser. Since different browsers may have different versions of the Java plug-in, the developers stuck to a very early version of Java (Java 1.0) to ensure that the applets worked on all user computers (especially old computers at industrial manufacturing sites). This old version of Java not only has a very limited set of graphical user interface APIs, but also is very buggy. Moreover, this old Java applet interface has many pop-up windows in a very strict hierarchy (parent and child relationship). If a user accidentally closes the wrong window, the whole application could crash. It was apparent there was a need to consolidate the Web interface while simultaneously getting away from the plug-in technology (browser/computer dependent). After carefully consideration and assessment, PHP was chosen to replace Java applets in WBCSim. PHP is the acronym for Hypertext Preprocessor, which is an open source, server-side, HTML embedded scripting language used to create dynamic Web pages. Other similar server scripting technologies are JSP (Sun) and ASP (Microsoft). Upon any HTTP request, the server runs PHP scripts and sends plain HTML code back to the client, where no plug-in is needed. While implementing this new PHP interface, most of the old pop-up windows were incorporated into frame-based Web pages. Studies with student users prove the new interface significantly improves WBCSim's usability and is much more user friendly.

Today a user launches the WBCSim Web page from a browser window, which contains a list of the process simulation models that WBCSim supports. Under each model, a user can either choose to enter (define parameters for) a simulation model by clicking "Use This Model as Guest" or read more information about the model by clicking "Detailed Description". Upon choosing a particular simulation model, a user has the choice of creating a new simulation, or investigating the stored simulation runs. There are additional choices at the right top corner of the page to allow a user to go back to the home of the current model, switch models, refresh a page, enable frames, submit comments, and logout; there is also a question mark icon that leads to help. These additional choices are always at the top and the bottom (in case the Web page is too long) of the page to help a user. There are two kinds of stored simulation runs—incomplete ones and complete ones. For incomplete

simulation runs, a user can resume and finish off the simulation. For complete simulation runs, a user can view the result of that simulation, change the input values of the simulation to run it again, and perform an "Advanced Option". In the "Advanced Option", a user is allowed to filter the simulation runs displayed. By selecting either "Title" or "Date" radio buttons, a user can narrow the filter process to either field. Then a user can input a regular expression (in UNIX grep style) in the text field, and press the "Filter" button to execute the filter process, which will reload this Web page with the simulation runs that meet the filter criteria. The "Advanced Option" also allows a user to apply comparison functions by selecting an item from the "Compare Option" dropdown list. Each process model in WBCSim has its own comparison functions. For example, the OSB model has three, which allow a user to compare "Void Fractions/Contact Area", "Density", and "Coefficient of Variation" among the simulation runs selected. If "Void Fractions/Contact Area" is selected from the "Compare Option" dropdown list, a new page is loaded with more options that allow comparisons among "space/mat", "lumen/mat", "void/mat", "lumen/flake", and "contact area". A user can also specify the interval of the y-axis in order to zoom the comparison graph. To the right of these stored simulation runs, there is information about model specific alerts and announcements and a user profile.

If a user decides to create a new simulation, he will actually see the specific model interface. There are some common features among all the model interfaces (see Fig. 4.1). Under the "WBCSimulator" title, there is a series of names and arrows to show the current navigation stage. In the left frame, there are many links corresponding to the steps necessary to run a particular process simulation model. When these links are clicked, the corresponding Web pages will show up in the right frame. The first link is "Introduction" that explains the model behavior at an abstract level, and also gives general directions about how to use the model. The next link is "Set Name", which allows a user to name (describe) the simulation. Following "Set Name", there are model-dependent links to allow a user to specify necessary simulation parameters in the right frame with any number of text boxes, dropdown lists, radio buttons, or plain buttons. At the bottom of the left frame, there are links to "Run Simulation", "Save Inputs", "Save Results" and "Delete". The "Run Simulation" link will display a Web page to confirm and send the HTTP request to execute the proper PHP code with input values defined in the interface. This will cause a UNIX or Perl script to run compiled FORTRAN code, and any additional optimization or visualization codes. The "Save Inputs" link will only save the input values in the database as an incomplete simulation run, while the "Save Results" link will save both inputs and outputs in the database as a complete simulation run. The "Delete" link will delete the current simulation run from the database and any associated files. The "View Results" link will appear right under "Save

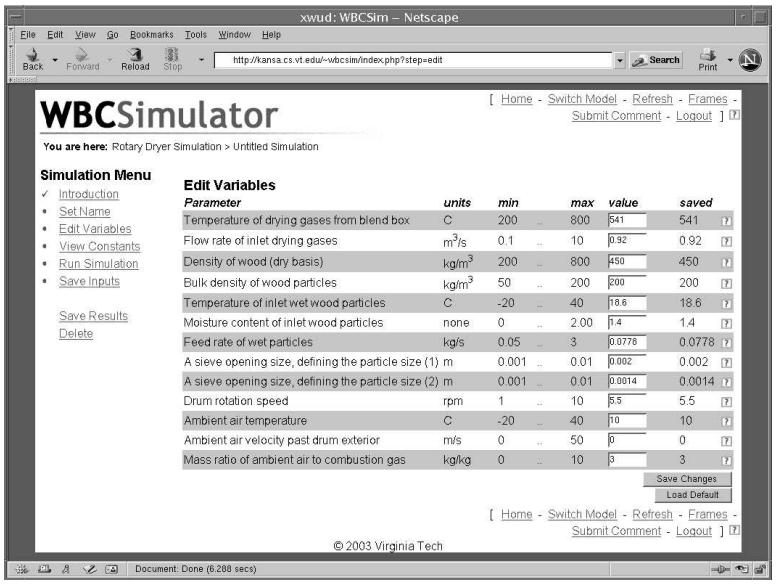


Fig. 4.1. The RDS model user interface.

Inputs" if a simulation is executed. The "View Results" will show the simulation results in both textual (normally tables of numbers) and graphical (VRML files, GIF files) forms.

The RDS (Fig. 4.1) and RFP models have the simplest user interfaces in WBCSim. There are only few model-dependent links such as "Edit Variables", "Edit Wood Layer Properties", and "View Constants". A user can simply enter values for various input parameters through text boxes and then run the simulation. The CMA model has evolved to support six calculations: "Design", "Analysis", "General Strength", "Tensile Strength", "Shear Strength", and "Bending Strength". A user can select a dynamically loaded calculation from a dropdown list in "Select Submodel". The CMA model includes two sets of defaults—structure and loading—in "Input Structure" and "Input Loading", which allow a user to cross reference different sets of defaults by setting one structure parameter set and then running any calculation with any loadings. The CMA model has rows of input data representing layers of the wood composite. A user can activate a layer by selecting the left most checkbox for that layer.

The OSB model (Fig. 4.2) is similar to the CMA model with input data representing layers of the wood composite. However, these layers in the OSB model are incorporated into the left frame, because there are too many properties for each layer. A user can enter the number of layers at "Edit Sim Variables." After clicking save, the links corresponding to each individual layer are displayed in the left frame menu. A user can further define the layer properties through these links. For example, clicking "Edit Layer 1", a user can select desired distributions for "Length", "Width", "Thickness", and "Density", along with "Orientation", "Flake Color", and "Number of Flakes to Display" for the first layer. Clicking the "Distribution" link, a user can further define the actual (probability density) distributions.

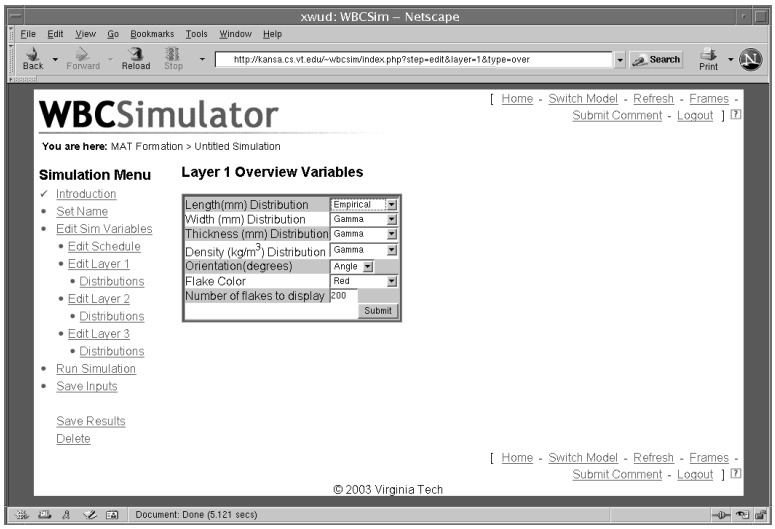


Fig. 4.2. The OSB model user interface.

For example, if selecting the "Empirical" distribution in the "Length Distribution" dropdown list of link "Edit Layer 1", a user will see there are forty-one text boxes under "Length Distribution" in the link "Distribution", allowing a user to define the number of intervals and each interval's starting and ending points. Also, as in the CMA model, the OSB model provides a set of defaults rather than a single default like most of the other models.

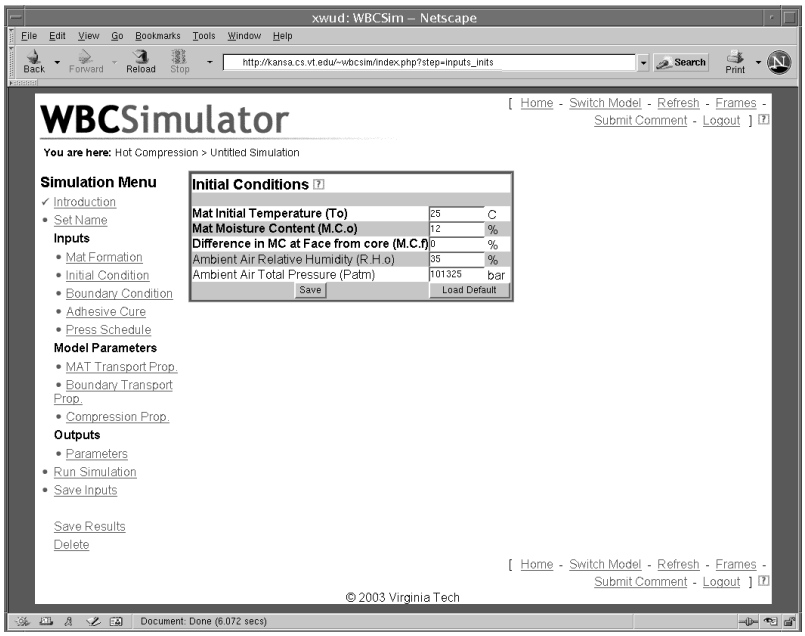


Fig. 4.3. The HC model user interface.

The HC model (Fig. 4.3) interface differs from the other models' interfaces by dividing the simulation parameters into three groups in the left frame menu: (1) model inputs, (2) model

parameters (execution specification), and (3) model outputs. The first group contains links for specifying the parameters related to the material, initial condition, boundary condition, adhesive cure, and press schedule. The second group defines the HC model execution (numerical solution of a nonlinear partial differential equation) by specifying mat transport properties, boundary transport properties, and compression properties. The last group specifies the output, such as color or black and white, the time interval for generating frames for animation, and other parameters determining which data/image files are generated.

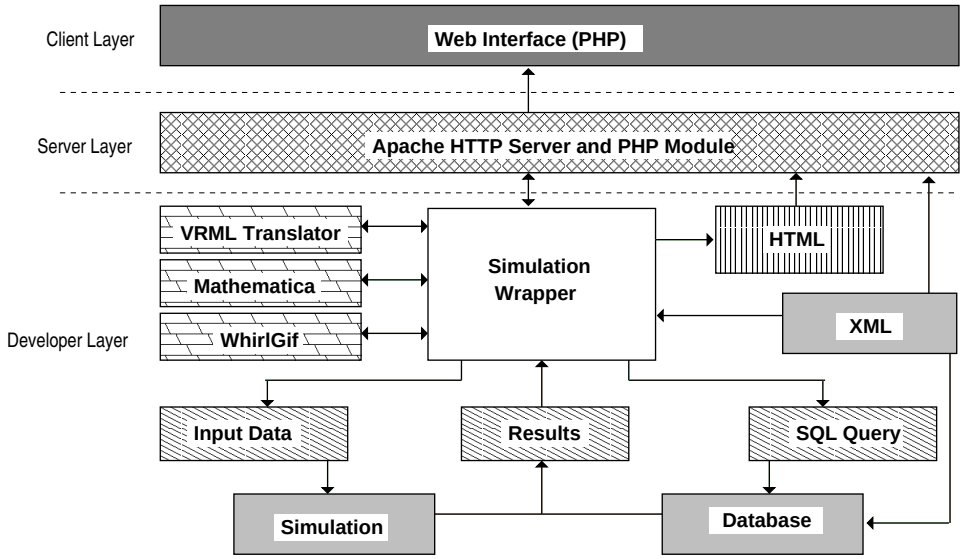


Fig. 4.4. WBCSim architecture overview.

## 4.6 WBCSim Architecture

The current software architecture of WBCSim follows the three-tier model described by Goel et al. [41], in which the tiers are (1) the client layer—user interface, (2) the server layer—Apache Web server and a PHP module, and (3) the developer layer—legacy simulation codes and various optimization and visualization tools running on the server. These layers are shown in Fig. 4.4.

### 4.6.1 Client Layer

The client layer is the only layer visible to end-users and typically the only layer running on the local machine. The main part of the client layer is the user interface, which is described in Chapter 4.5.

The user interface is generated from PHP and XML. As described earlier, each model has a XML datasheet, which uses a set of customized tags to describe the model. The size of the XML datasheets in WBCSim varies from 109KB (around 7000 lines) to 18KB (800 lines) depending on how complicated the model input is. A typical reason for a large

XML datasheet is that the XML datasheet contains information about many input/output parameters, and many sets of default values. Fig. 4.5 shows the actual XML datasheet for the RDS model. Some common tags among all the models are:

```
<simulation>
       <simulation_name></simulation_name>
       <simulation_name_abbreviation></simulation_name_abbreviation>
       <simulation_image></simulation_image>
       <short_description></short_description>
       <long_description></long_description>
       <announcement></announcement>
       <parameter_section>
              <parameter>
                     <name></name>
                     <unit></unit>
                     <min></min>
                     <max></max>
                     <default></default>
                     <help_text></help_text>
              </parameter>
       </parameter_section>
</simulation>
```

At the beginning, the PHP script reads this XML datasheet and uses a XML parser to parse the information into an array structure. Then different parts of the PHP script use this array to construct interfaces, set parameter defaults, store user inputs, and assemble the inputs into a string (the corresponding simulation wrapper will feed this string to the FORTRAN simulations). The first row of input parameters in the browser window shown in Fig. 4.1 is actually generated from a portion of the XML file shown in Fig. 4.5. Therefore, adding a parameter in the XML datasheet will simply add an element in the dynamic array. Then, the PHP code referring to this array will automatically update the interface when executed.

The client layer also contains viewers for one of the visualization tools, the VRML translator. WBCSim requires a VRML 2.0 viewer for the RFP model. The VRML viewer serves as a plug-in to the Web browsers. A user has other substitute visualization results if the VRML plug-in does not exist.

The client layer handles communication with the server layer by sending HTTP requests. When a user clicks a link, a URL appended with necessary action commands and parameters is sent to a Web sever in the server layer, where the corresponding PHP scripts, along with other possible Perl scripts, visualization tools, and database operations, are executed.

### 4.6.2 Server Layer

By separating the legacy simulation codes from the user interface, the server layer functions as the key to how WBCSim can run a text-only application from a Web browser. The server layer consists of two components: an Apache HTTP server and a PHP module.

```
<?xml version="1.0"?>
<simulation>
  <simulation_name>Rotary Dryer Simulation</simulation_name>
  <simulation_name_abbreviation>RDS</simulation_name_abbreviation>
  <simulation_image>../img/Rotary_Dryer.gif</simulation_image>
  <short_description>This model simulates the drying behavior of wood particles in a rotary
    dryer.</short_description>
  <long_description>The rotary dryer simulation model was developed as a tool to assist in
    the design of drying systems for wood particles, such as used in the manufacture of
    particleboard and strandboard products. The rotary dryer is used in about 90 percent
    of these processes. It consists of a large, horizontally oriented, rotating drum (typically
    3 to 5 m in diameter and 20 to 30 m in length). The wet wood particles are mixed
    directly with hot combustion gases at the inlet. The gas flow provides the thermal
    energy for drying, as well as the medium for pneumatic transport of the particles
    through the length of the drum. Interior lifting flanges serve to agitate and produce a
    cascade of particles through the hot gases. This process uses a co-current
    flow.<br><br>The RDS model consists of a series of material and energy balance
    equations, which are defined for each cascade of wood particles. A cascade cycle begins
    when a particle drops off a lifting flange and falls to the bottom of the drum. This is
    followed by travel along the periphery of the drum, when the particle is caught by a
    lifting flange. The cascade ends when the particle attains its maximum angle of repose
    and tumbles off of the lifting flange. The heat and mass flows between cascade cycles,
    and the distance of travel along the length of the drum for each cycle, are determined
    by algebraic equations. The user must supply the inlet conditions of the hot gases and
    wet wood particles, as well as the physical dimensions of the drum and lifting flanges,
    flow rates, and thermal loss factor for the dryer. The RDS model predicts the moisture
    content and temperature of the wood particles for each cascade in the drum, and
    predicts the gas phase composition and temperature at each cascade.<br><br>Kamke,
    F.A.; Wilson, J.B. (1985) Computer simulation of a rotary dryer: retention time,
    American Institute of Chemical Engineers J., 32(2), 263-268.<br><br>Kamke, F.A.;
    Wilson, J.B. (1985) Computer simulation of a rotary dryer: heat and mass transfer,
    American Institute of Chemical Engineers J., 32(2), 269-275.<br><br></long_description>
  <announcement></announcement>
  <parameter_section>
    <title>Variable</title>
    <parameter>
      <name>Temperature of drying gases from blend box</name>
      <unit>C</unit>
      <min>200</min>
      <max>800</max>
      <default>541</default>
      <help_text>Combustion gases from burner are mixed with ambient air, or recycled
        air, in the blend box. The airflow from the blend box is a mixture of these gases.
        This is the inlet air temperature to the rotating drum.<br><br><center><img
        src=../img/RDSHELP1.jpg alt=RDSHELP1.jpg border=0></center></help_text>
    </parameter>
```

⋮

Fig. 4.5. The RDS model XML datasheet, an excerpt from the file `rds.xml`.

In 2002, Shu et al. [91] described a telnet connection method, which replaced the old Javamatic server and socket communication paradigm described in [41], and facilitated setting up WBCSim guest accounts on the server machine that do not require full account

privileges. The guest account and normal user accounts (for commercial, paying users) permit users to save and track their different simulation runs, and protect their data from being removed or overwritten by others. However, this telnet connection needs the client to use a special network port to communicate with the server. In today's world of highly tightened security, it is problematic to ask a user to open/monitor a special port on his computer in order to use WBCSim. A standard Apache HTTP server with a PHP module only requires the standard HTTP port 80, which is the standard and normally open/monitored port at the client computer for Web browsing.

The PHP module serves as a plug-in to the Apache HTTP server. When the HTTP server receives a request for a PHP file, the PHP module will parse and execute that PHP file and return HTML back to client. This PHP module supports sessions. A session is a series of related interactions between a single client and the Web server, which can take place over an extended period of time. By using sessions, the PHP module can concurrently accept multiple requests from different clients and direct executions of multiple simulations.

### 4.6.3 Developer Layer

As its name suggests, the developer layer contains legacy programs created by researchers to model WBC materials and manufacturing processes. These legacy programs are the heart of WBCSim. In general, WBCSim supports legacy programs written in any programming language as long as the program takes its input parameters from UNIX stdin. The input parameters can be a few numbers or a large data file. In particular, WBCSim supports five FORTRAN 77 and Fortran 90 simulation programs corresponding to the five models described in Chapter 4.3.

While each Fortran program has its own input format (stdin could be redirected to a file), the server layer communicates data with the developer layer via strings of parameters separated by white space (spaces, tabs, newlines). In order to cope with this string format, each legacy program is "wrapped" with a customized Perl script. The script receives this string of parameters from the PHP module at the server, and converts those parameters into an appropriate format for the Fortran program. Then the script calls the legacy program into action, feeding it the input, invoking any required optimization and visualization tools, packing all Fortran output in HTML files, and returning HTML files, first to the server layer, and then to the client layer. With this architecture, the developer layer is independent of the other layers, which makes the process of designing, and integrating new simulation codes relatively easy.

Originally, these wrappers had a set of input parameters hard coded in the code. Today these wrappers use the XML datasheet to convert the string of input parameters into the appropriate format for the Fortran program, similar to how the interface layer will read/parse the same XML file, and utilize array structures (described in Chapter 4.6.1).

Shu et al. [90] described some additional operations in those wrappers to support the experiment management component. There are also other wrappers associated with the save, retrieve, and compare functions from the user interface at the client layer. Like the simulation wrapper, these wrappers receive data from the server layer, and convert those input values into an appropriate format for the database. Originally, this conversion process was also hard coded. However, work converting WBCSim to use the same XML sheet to construct those necessary queries and database tables is currently underway.

Other than wrappers, the developer layer also includes optimization and visualization tools to maximize the simulation's value to the user [91]. Since the wrappers prepare the inputs for and read the outputs from these optimization and visualization tools, these tools do not interact with the XML datasheets directly.

Fig. 4.6 shows the XML schema for the XML datasheet in Fig. 4.5. The schema defines and constrains the values in an XML sheet.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="simulation">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="simulation_name" type="xs:string"/>
      <xs:element name="simulation_name_abbreviation" type="xs:string"/>
      <xs:element name="simulation_image" type="xs:string"/>
      <xs:element name="short_description" type="xs:string"/>
      <xs:element name="long_description" type="xs:string"/>
      <xs:element name="announcement" type="xs:string"/>
      <xs:element name="parameter_section">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="parameter">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="name" type="xs:string"/>
                  <xs:element name="unit" type="xs:string"/>
                  <xs:element name="min" type="xs:decimal"/>
                  <xs:element name="max" type="xs:decimal"/>
                  <xs:element name="default" type="xs:decimal"/>
                  <xs:element name="value" type="xs:decimal"/>
                  <xs:element name="help_text" type="xs:string"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Fig. 4.6. XML schema for the RDS model XML datasheet.

## 4.7 Scenarios

### 4.7.1 Usage Scenario

Describing a typical usage scenario of WBCSim is instructive. Consider research into the properties of oriented strand board products, which would use the OSB model. First, a scientist samples the face and core layer flakes from an industrial production line. The scientist measures the geometry and the weight of each flake. The flake property data sets are then used to estimate the statistical probability density functions of the flake properties. Then, the scientist opens a browser window and launches the WBCSim Web page, which is dynamically generated from PHP and XML. The guest account is the default login option, which stores all the input and output data in a temporary directory that is purged periodically. The scientist selects the OSB model from a list of all the available models by clicking "Use This Model as Guest". The scientist then launches the OSB model interface (shown in Fig. 4.2) by clicking "New Mat Formation". By specifying the probability density functions of the flake properties obtained offline earlier, the scientist can create a statistically valid model instance of the spatial structure of multiple layers of flakes, each with various parameters defining the number of flakes, length distribution, width distribution, thickness distribution, density distribution, orientation, flake color, and number of flakes shown. The latter two are solely for visualization.

Next, upon user requests, the input parameters (Boolean, numeric, or alphanumeric) are collected in the PHP code at the server as a long string (parameters are separated by white space). Then, the PHP module calls the OSB wrapper to convert this string into a SQL query, and checks if this set of input parameters is stored in the database.

If the output does not exist in the database, the OSB wrapper converts this input string into a data file destined for the OSB Fortran 90 simulation program by referencing the corresponding OSB XML datasheet. Since the OSB simulation code has its own text-based user interface taking input from stdin, a temporary file is generated to contain all of the appropriate commands. Stdin is then redirected to this temporary data file for the simulation. The OSB wrapper then invokes the OSB simulation code with this temporary file as stdin along with the properly formatted data file. When the simulation code is executing, the OSB wrapper monitors the simulation output stream for strings indicating execution milestones. The OSB wrapper uses the standard error stream for storing these messages into a log file, which a user can also access from the Web by clicking "Check Status".

If the output exists in the database (a simulation with this exact same input data has been run earlier, and the output saved), the OSB wrapper extracts the output from the database and places it into data files in the same format as it would have been written by the OSB Fortran 90 simulation code.

When those output data files are ready (no matter whether they came from the Fortran 90 simulation program or the database), the OSB wrapper calls Mathematica to read those data files and generate plots with various Mathematica commands such as ListContourPlot, ListPlot3D, MultipleListPlot, and Graphics3D. Mathematica also converts these internal graphics data structures into GIF format so that they can be viewed in a browser. Fig. 4.7 shows a three-dimensional visualization of a three-layer random flake mat created with Graphics3D by Mathematica. Finally, the OSB wrapper embeds these GIF files in HTML files and returns these HTML files to the PHP module. These HTML files are passed back to the client, where a user can click "View Results" to access them. Shu et al. [90] further described scenarios involving a comparison between results from multiple simulation runs.

### 4.7.2 Typical Change Scenario

Describing a typical change scenario for WBCSim is effective to demonstrate how XML unifies the components in the three implementation layers. Consider adding an input parameter—"Difference in Moisture Content at Face from Core"—in the "Initial Conditions" section for the HC model. A scientist initiates this change by adding this parameter in the Fortran simulation code (its input, its algorithm, and its output). In order to change the rest of the system, this scientist can download a copy of the HC XML datasheet from the Web (for example, http://.../xml/hot.xml). After locating the parameter section defining "Initial Conditions", the scientist can duplicate the tags and contents of an existing parameter as a template, and then edit and add this new parameter's full name, short name, physical units, and default value. This scientist can then submit this updated XML datasheet to a developer, who can verify the syntax of this datasheet by running it against a XML Schema. After properly backing up the old version of the XML datasheet, the new one can be plugged in. Then, the interface at the client layer and the wrappers at the developer layer will access this new XML datasheet instantly when a new session is started. The corresponding interface will have this new parameter as a textbox; the wrappers will prepare this new parameter as an additional input in the input data file for the HC simulation code. In the future, necessary scripts can even backup and alter the database tables as well, which is yet to be implemented in WBCSim.

The above scenario has many assumptions. Obviously, the XML datasheet only supports a set of customized tags (such as the ones listed in Chapter 4.6.1). If it is a simple change like above, a wood scientist can easily insert the new input parameter by using existing tags, and obey the XML structure. If a change requires extra tags (to define some special information related to a parameter or a set of parameters), certainly a wood scientist needs to consult with a developer to coordinate the modification of the XML schema (such as adding more tags and embracing more complex XML structures). The above scenario also assumes that the output stays the same after adding the new input parameter. That is

seldom the case. Therefore, a wood scientist would also need to locate the output related sections within the XML datasheet to add possible additional output information, so that the wrappers can parse such information from the XML datasheet describing how to deal with the additional output.

How to perform such change management automatically is an active area of research in database systems and, needless to say, is quite critical in PSE research as well. By explicitly encoding modeling and execution assumptions in a declarative format such as XML, the ability is reserved to revisit WBCSim system design at a later stage and provide more sophisticated change management facilities. In summary, changes in XML datasheets can automatically propagate to the entire PSE system assuming:

(1) The change requires only existing tags.

(2) The change obeys existing XML structure—XML schema.

## 4.8 Future Work and Conclusions

Shu et al. [91] described planned additions of experiment management (EM), collaboration support, and high performance computing, which are the most appealing missing features given the current state of WBCSim development. While WBCSim is being developed on several different fronts, the development track for incorporating XML technology intends to support automatic database changes, be more flexible and dynamic, and improve XML storage. Each of these future developments is discussed in turn next.

Currently, when a XML datasheet is changed, the behavior of the interface and wrappers accessing this XML datasheet will change accordingly. The database, however, has to be changed manually to store modified simulation runs. Perl scripts can be developed to backup the existing simulation runs, and use the XML to alter/create the new database tables automatically.

The existing set of the XML tags customized for WBCSim has yet to be tested for usability and flexibility. A wood scientist should be able to understand and change the XML datasheet easily. The goal is to make changing the XML datasheet a trivial task for a wood scientist who does not know much about XML.

Upon submission of an updated XML datasheet, a developer is still needed to verify the syntax of this XML datasheet, and monitor whether the change can be realized in the WBCSim interface and wrappers (for example, a change should not affect WBCSim interface usability). Perhaps this process can be further automated with editing, submitting, verifying, and testing a XML datasheet being done by a wood scientist himself using a Web interface.

The XML files are currently stored as plain text files under a certain directory in the file system. When the XML files are updated, the old ones are renamed with a date. Certainly

a better way of archiving/documenting these XML files is needed, especially when a set of XML files can be used to define a version of WBCSim.

WBCSim has evolved steadily from a prototype PSE intended as a tool for computer science PSE research and a Web-based interface for a few legacy computer programs, to a manufacture-oriented near commercial quality PSE that is seriously used by wood science researchers in industry and academia, which makes WBCSim models stable enough so that information about these models can be encoded in XML. Since interesting computational capabilities are still lacking [91], WBCSim will remain an object of computer science research for some time to come. Yet the program's interfaces, models, and output visualizations are now good enough to be used as production tools by wood scientists. The directions in which computer scientists would like to take WBCSim (collaboration, data mining, grid computing) are quite different from the directions that wood scientists would prefer for WBCSim (new models, refining existing models, more interface and visualization options). The present layered architecture of WBCSim supports these divergent development directions well.

With the addition of XML technology, WBCSim is now considerably easier to change and maintain. From the user perspective, using XML technology significantly reduces the development time so that users can enjoy new features more quickly. From the developer perspective, under the assumptions explained in Chapter 4.7.2, a wood scientist can edit the XML datasheet himself to make a change, instead of spending time explaining to a programmer how to make such a change. Furthermore, a developer doesn't have to hardcode such a change at various implementation layers, such as in the interface and wrappers.

The original stated goal [41] of WBCSim was to provide "an integrated set of facilities allowing wood scientists to concentrate on high-level problem solving rather than on low-level programming details and application scheduling/execution, allowing users to define, record, and modify problems, and visualize and optimize simulation results". Using XML technology to unify the WBCSim implementation layers has definitely brought this goal closer.

# Chapter 5: Execution Management via Computational Steering*

## 5.1 Introduction

Every day, computational scientists from various disciplines, such as wood science, physics, chemistry, and biology, using various computing environments, perform numerical experiments on their computers. Each of these numerical experiments (hereafter referred to as an experiment) goes though similar life cycles even though these sciences have very little in common. Ioannidis et al. [53] identified three stages in a typical experiment life cycle: 1) design of experiment, 2) data collection, and 3) data exploration. Besides the possible evolving nature of the design of experiment stage, scientists often focus their research on data collection and data exploration. Moreover, beyond just monitoring a running simulation and viewing the final results at the end, the scientists may wish to interactively control an experiment. This is known as "computational steering", which allows scientists to adjust parameters of the computation on-the-fly and explore "what if" analysis [38]. This often helps scientists deepen their understanding of underlying principles. As effects due to changes in parameters become more instantaneous, the cause-effect relationships could become more evident [76].

Mulder et al. [74] further classified the three main uses of computational steering as: model exploration, algorithm experimentation, and performance optimization. With model exploration, computational steering is used to explore parameter spaces and simulation behavior to gain additional insight into the simulation model. In algorithm experimentation, computational steering allows the user to adapt program algorithms during the run, for instance to experiment with different numerical methods. In performance optimization, steering is used to manually improve an application's performance, e.g., to perform load balancing interactively in parallel applications.

A problem solving environment (PSE) is a computational system that provides a complete and convenient set of high-level tools for solving problems from a specific domain [84]. A PSE commonly addresses many issues: Internet accessibility to legacy codes, visualization, experiment management (EM), multidisciplinary support, recommender systems, collaboration support, optimization, high performance computing, preservation of expert knowledge, design extensibility, and pedagogical uses [105]. SCIRun [76] is a framework for PSE construction from domain specific packages (such as the BioPSE package), which consist of predefined data types, algorithms, and modules. Such a framework enables the user to design scientific simulations and interactively control such simulations. Discover

* Computational steering in the problem solving environment WBCSim, J. Shu, L.T. Watson, N. Ramakrishnan, F.A. Kamke, S. Deshpande, *Software: Practice and Experience*, submitted.

[68] is a PSE focusing on a rule-based visualization system and optimization. Chapter 5.2 provides more details about SCIRun, Discover, and other computing environments with computational steering capability. These PSEs with computational steering capability generally focus on the *construction* of *new* simulations rather than making computational steering work for an *existing* PSE embedded with legacy simulation code, especially large scale simulation codes written decades ago without computational steering design in mind. This chapter describes a practical implementation of a minimal set of components to enable computational steering capability by using WBCSim as an example.

WBCSim is a PSE that increases the productivity of wood scientists conducting research on wood-based composite (WBC) materials and manufacturing processes. It integrates Fortran 90 simulation codes with a Web based graphical front end, an optimization tool, and various visualization tools. The WBCSim project was begun in 1997 with support from USDA, Department of Energy, and Virginia Polytechnic Institute & State University (VPI&SU). It has since been used by students in several wood science classes, by graduate students and faculty, and by researchers at several forest products companies. WBCSim also serves as a test bed for the design, construction, and evaluation of useful, production quality PSEs.

Goel et al. [41], [42] described an early version of WBCSim. Over the years, WBCSim has evolved in many ways. More advanced simulation models such as a mat formation model and a hot compression model are now incorporated into the system [91]. An experiment management tool [90] is also included now. This EM tool uses a database management system (DBMS) to store simulation input and results at the server end, provides helpful graphical interfaces to facilitate user access at the client side, and then uses various scripts to connect all of these components. Recently, XML (Extensible Markup Language) was used to unify the implementation layers of WBCSim [89]. A XML datasheet is tailored for each model (WBCSim has five models). The WBCSim interface layer, server scripts, and database management system all use this XML datasheet to improve the usability and maintainability of the three layers—client, server, developer—comprising the WBCSim system.

This chapter shows that with a reasonable number of changes, with a small set of components, a computational steering capability can be achieved for any PSE embedded with legacy simulation codes. This small set of changes and components includes 1) designing and adding a very simple steering module at the legacy simulation code level, 2) providing a way to monitor simulation execution (alert users when it is time to steer), 3) adding an interface to access and visualize simulation results, and even compare intermediate results across multiple steering attempts. This chapter describes in detail how such changes and components are implemented in WBCSim, and how computational steering can effectively

reduce computational time, make research more efficient, and open up new product design opportunities.

The chapter is organized as follows. Chapter 5.2 reviews some related work in PSEs and computational steering. Chapter 5.3 briefly describes WBCSim. Chapter 5.4 gives the motivation for adding computational steering capability. Chapter 5.5 elaborates on the WBCSim user interface, the model interface similarities, and also the unique aspects of the hot compression model, which is used to demonstrate the computational steering feature. Chapter 5.6 explains the architecture of WBCSim and how the computational steering capability is developed within this architecture. Chapter 5.7 demonstrates a typical scenario of computational steering for WBCSim. Chapter 5.8 generalizes the WBCSim computational steering work and summarizes the lessons learned. Chapter 5.9 outlines some future directions of computational steering for WBCSim and draws conclusions.

## 5.2 Related Work

There are many problem-specific PSEs developed for various application domains. Some of these PSEs have computational steering capabilities.

### 5.2.1 Problem Solving Environments

A PSE is a relatively new computing paradigm that was first introduced in problem domains such as partial differential equations (ELLPACK [41] and its descendents [8] for solving two- and three-dimensional elliptic partial differential equations) and linear algebra (Linear System Analyzer [14] for manipulating and solving large-scale sparse linear systems of equations).

Since then, many new PSEs have been introduced. Gismo [18], created at Washington University, is an object oriented Monte Carlo package for modeling all aspects of a satellite's design and performance. It has played a significant role in the design of the Gamma Ray Large Area Space Telescope, the successor to the Compton Gamma Ray Observatory that was launched into space in 1991 to explore the gamma ray portion of the electromagnetic spectrum for astrophysics research. VizCraft [39], developed at Virginia Polytechnic Institute and State University, provides a graphical user interface for a widely used suite of analysis and optimization codes that aid aircraft designers during the conceptual design of a high speed civil transport. VizCraft combines visualization and computation, encouraging the designer to think in terms of the overall problem solving task, not simply using the visualization to view the computational results. Also, a computing environment developed by Chen et al. [24], which combines particle systems, rigid body particle dynamics, computational fluid dynamics, rendering, and visualization techniques to simulate physically realistic, complex dust behavior, has been shown to be useful in interactive graphics applications for education, entertainment, or training. Expresso [92], a microarray experiment management PSE, is designed to assist biologists in planning, executing, and interpreting microarray experiments.

It serves as a unifying framework to study data-driven application composition systems, as envisaged under the NSF Next Generation Software (NGS) program. Physical and analytical stages of the microarray process are mirrored in Expresso with computational models from biophysics, molecular biology, biochemistry, robotics, image processing, statistics, and knowledge representation. These models are pushed deeper (earlier) into the design process to help avoid costly design errors and to provide, as needed, surrogate functions for the traditional stages of microarray experiments. L2W [31] is a PSE for land use change analysis. L2W organizes and unifies the diverse collection of software typically associated with ecosystem models (hydrological, economic and biological). It provides a Web based interface for potential watershed managers and other users to explore meaningful alternative land development and management scenarios and view their hydrological, ecological, and economic impacts. The JigCell model builder [3], [97], [2], [98], allows users to define chemical kinetic models as a set of reaction equations using a spreadsheet (an example of direct entry of equations) and outputs model definitions in the Systems Biology Markup Language. This spreadsheet paradigm demonstrates its effectiveness in reducing the number of errors made by systems biology modelers when compared to hand conversion of a metabolic pathway wiring diagram to differential equations. Site Specific System Simulator for Wireless System Design ($S^4W$) ([93], [72]) is a PSE that integrates visualization and computational tools with a high level graphical user interface. $S^4W$ improves the ability of wireless system engineers to design an indoor wireless system by encouraging them to think in terms of designing the system for optimal performance, while issues of computation management, data management, and location of computer resources are hidden from the user. The Land Information System (LIS) [57] is a multiscale hydrologic modeling and data assimilation framework that integrates the use of satellite and ground based observational data products with advanced land-surface modeling tools to aid several application areas, including water resources management, numerical weather prediction, agricultural management, air quality, and military mobility assessment.

Watson et al. [105] present a thorough summary of the key attributes of a PSE (described in Chapter 5.1), and also compare a PSE with other similar computing environments—a decision support system (DSS) and a geographic information system (GIS). Roughly, a DSS emphasizes the functionality (analysis, planning, and decision making), a GIS emphasizes the nature of data and information (spatial), while a PSE emphasizes the problem domain (such as wood-based composite materials).

### 5.2.2 Computational Steering

In 1987, a US National Science Foundation workshop on scientific visualization reported that scientists want to drive the scientific discovery process and interact with their data. Interactive visual computing is a process allowing scientists to communicate with the

data production process and manipulate data visualization during that process. Such a sophisticated process could allow scientists to steer, or dynamically modify computations while they are occurring [71]. The computational steering concept became established, and developed and flourished in the 90s and the turn of the century, when it has been applied in many scientific domains such as pollution analysis [15], molecular dynamics simulations [68], and medicine (cardiology and neuroscience) [76].

SCIRun [76] is one of the oldest PSEs that allows the interactive construction, debugging, and steering of large-scale scientific computations designed with predefined domain specific packages. The primary purpose of SCIRun is to enable the user to interactively control scientific simulations while the computation is in progress. This control allows the user, for example, to vary boundary conditions, model geometries, and/or various computational parameters during simulation. DISCOVER [68] is an interactive and collaborative PSE that enables geographically distributed scientists and engineers to collaboratively monitor and control parallel/distributed applications using Web based portals. The DISCOVER interaction servers build on servlet technology and enable clients to connect to, and collaboratively interact with, registered applications using a conventional browser. Geographically distributed servers are interconnected using CORBA. DISCOVER also has a concept of rule based visualization. These rules are decoupled from the system and can be externally injected to manage visualization behaviors at runtime, such as automatically selecting the appropriate visualization routines and methods, and adjusting the extraction threshold. CUMULVS [38] is a software infrastructure for developing collaborative environments. It supports interactive visualization and remote computational steering of distributed applications by multiple collaborators. In particular, CUMULVS provides a mechanism for constructing fault tolerant check points, and migrating applications in heterogeneous distributed computing environments. Computational Steering Environment (CSE) [65] provides an environment that allows scientists to easily define an interactive interface to an ongoing simulation. The architecture of CSE is a set of processes, called satellites, which implement standard visualization operations. The simulation itself is also packaged as a satellite. Satellites cooperate with each other by communicating data to a central data manager, which, in turn, notifies any interested satellites about data change. Falcon [32] is a set of tools that collectively support on-line program monitoring and steering of parallel and distributed applications. The four major conceptual components of Falcon include (1) a monitoring specification mechanism, which consists of a low level sensor specification language and a high level view specification language, (2) mechanisms for on-line information capture and analysis, (3) mechanisms for program steering, and (4) an associated system for the construction and use of graphical displays of program information.

In a survey of computational environments, Mulder et al. [74] discussed computational steering from various aspects such as scope, architecture, and user interface. They further

compared VASE [44], [55], SCIRun [76], Progress/Magellan [103], [104], CUMULVS [38], VIPER [81], and CSE [65]. In a grid (distributed computational grid) context, there are several other computational steering systems such as GViz [15], RealityGrid [17], and AutoPilot/Virtue [102]. These systems vary by their steering method, visualization interface, and scheme of incorporation within a grid environment.

## 5.3 WBCSim

WBCSim is intended to increase the productivity of wood scientists conducting research on wood-based composite materials by making legacy file based Fortran programs that solve scientific problems in the wood-based composites domain widely accessible and easy to use. WBCSim currently provides Internet access to command line driven simulations developed by the Wood-Based Composites Center at VPI&SU. WBCSim leverages the accessibility of the World Wide Web to make simulations with legacy code available to scientists and engineers away from their laboratories.

WBCSim currently supports five simulation models that help wood scientists studying wood-based composite material manufacturing.

(1) Rotary dryer simulation (RDS). The rotary dryer simulation model assists in the design and operation of the most common type of system used for the drying of wood particles [58], [59].

(2) Radio-frequency pressing (RFP). The radio-frequency pressing model [83] was developed to simulate the consolidation of wood veneer into a laminated composite using high frequency energy.

(3) Oriented strand board mat formation (OSB). The mat formation model [112] creates a three-dimensional spatial structure of a layered wood-based composite (e.g., oriented strand board and waferboard); it also calculates certain mat properties by superimposing a mesh on the mat structure.

(4) Hot compression (HC). The hot compression model ([113], [114]) simulates the mat consolidation and adhesive cure that occurs during industrial hot-pressing of wood-based panels.

(5) Composite material analysis (CMA). The composite material analysis model [91] was developed to assess the stress and strain behavior and strength properties of laminated materials (e.g., plywood and fiber-reinforced composites).

The current software architecture of WBCSim follows the three tier model described by Goel et al. [41], in which the tiers are (1) the client layer—user interface, (2) the server layer—Web server and a PHP module, and (3) the developer layer—legacy simulation codes and various optimization and visualization tools running on the server. Chapter 5.6 elaborates on the details of the three layers while explaining how the computational steering fits into this architecture.

WBCSim is equipped with an optimization algorithm DOT (Design Optimization Tool) [96] and various visualization tools: VRML [4], Mathematica [108], and the UNIX utility WhirlGif. The reader is referred to [91] for an in-depth treatment of these tools.

In 2004, Shu et al. [90] described a WBCSim experiment management component, which integrates a Web based graphical front end, server scripts, and a database management system to allow scientists to easily save, retrieve, and perform customized operations on experimental data.

In 2008, Shu et al. [89] described how XML is used to unify the implementation layers for WBCSim. A XML datasheet is tailored for each of the five models (mentioned above). The WBCSim interface layer, server scripts, and database management system all use this XML datasheet to improve the usability and maintainability of the client, server, and developer layers. XML is the acronym for Extensible Markup Language, which allows a developer to create customized tags, supporting the definition, transmission, validation, and interpretation of data within/between applications, and among organizations.

As WBCSim has evolved in various ways throughout many years, its original goals remain the same: (1) to increase the productivity of WBC research and manufacturing groups by improving their software environment, and (2) to continue serving as an example for the design, construction, and evaluation of small-scale problem solving environments.

## 5.4 Rationale for Computational Steering

The more advanced models in WBCSim, such as the hot compression model (a two-dimensional nonlinear partial differential equation), its Fortran 90 core program and its visualization/optimization tools, can take hours to run on a fast (Sun Blade 2000) workstation. As mentioned above, the simulation result could be undesirable after such a long wait. Some initial results could enable scientists to avoid this wait, and effectively direct the simulation toward other more promising outcomes. Moreover, scientists are very interested in some intermediate processes in the simulation that could alter the final results. For example: the hot press closure time lasts for 15 to 60 seconds, but it could be carefully examined to see how it affects the characteristics of the final panel; the build-up of gas pressure inside the mat during the hot compression process can be studied to see if it is possible to reduce the press time and yet still achieve adequate cure of the adhesive. By augmenting WBCSim with computational steering capability, many of these questions can be investigated and possibly answered. In short, computational steering in WBCSim is a tool for model exploration, which could reduce computational time, make research more efficient, and open various new opportunities in the following ways.

(1) Computational steering provides interactive control that could save countless hours of wasted computation time waiting for final failed or uninteresting simulation results whose nature might have been apparent from the first few iterations.

(2) By showing the intermediate results, cause and effect relationships could be evident. That could prompt the scientist to conduct various "what if" scenarios, which could lead to interesting discoveries.

(3) Scientists can concentrate on certain interesting parts/steps of the simulation and study them in a more controlled/interactive manner, and see how those parts affect the computation as a whole.

(4) Humans (scientists) are again being placed in the center of the computation. Thus, human expert knowledge can be effectively leveraged to steer a large computation to a goal.

(5) Computational steering also opens doors for automated optimization, if scientists can define the boundary conditions and goals properly.

## 5.5 User Interface

In 2008, Shu et al. [89] described how the WBCSim Java applet interface (browser plug-in technology) can be replaced by PHP (server scripting technology) code. PHP stands for Hypertext Preprocessor, which is an open source, server-side, HTML embedded scripting language used to create dynamic Web pages. Other similar server scripting technologies are JSP (Sun) and ASP (Microsoft). Upon any HTTP request, the server runs PHP scripts and sends plain HTML code back to the client, where no plug-in is needed.

A user launches the WBCSim Web page from a browser window, which contains a list of the simulation models that WBCSim supports. Under each model, a user can either choose to enter a simulation model by clicking "Use This Model as Guest" or read more information about the model by clicking "Detailed Description". Upon entering a particular simulation model, a user has the choice of creating a new simulation, or investigating the stored simulation runs. There are additional choices at the right top corner of the page to allow a user to go back to the home of the current model, switch models, refresh a page, enable frames, submit comments, and logout; there is also a question mark icon that leads to help. These additional choices are always at the top and the bottom (in case the Web page is too long) of the page to help a user. There are two kinds of stored simulation runs—incomplete ones and complete ones. For incomplete simulation runs, a user can resume and finish off the simulation. For complete simulation runs, a user can view the result of that simulation, change the input values of the simulation to run it again, and perform an "Advanced Option". In the "Advanced Option", a user is allowed to filter the simulation runs displayed. By selecting either "Title" or "Date" radio buttons, a user can narrow the filter process to either field. Then a user can input a regular expression (in UNIX grep style) in the text field, and press the "Filter" button to execute the filter process, which will reload this Web page with the simulation runs that meet the filter criteria. The "Advanced Option" also allows a user to apply comparison functions by selecting an item

from the "Compare Option" dropdown list. Each model of WBCSim has its own comparison functions. For example, the OSB model has three, which allow a user to compare "Void Fractions/Contact Area", "Density", and "Coefficient of Variation" among the simulation runs selected. If "Void Fractions/Contact Area" is selected from the "Compare Option" dropdown list, a new page is loaded with more options that allow comparisons among "space/mat", "lumen/mat", "void/mat", "lumen/flake", and "contact area". A user can also specify the interval of the Y-axis in order to narrow the comparison. To the right of these stored simulation runs, there is information about model specific alerts and announcements and a user profile.

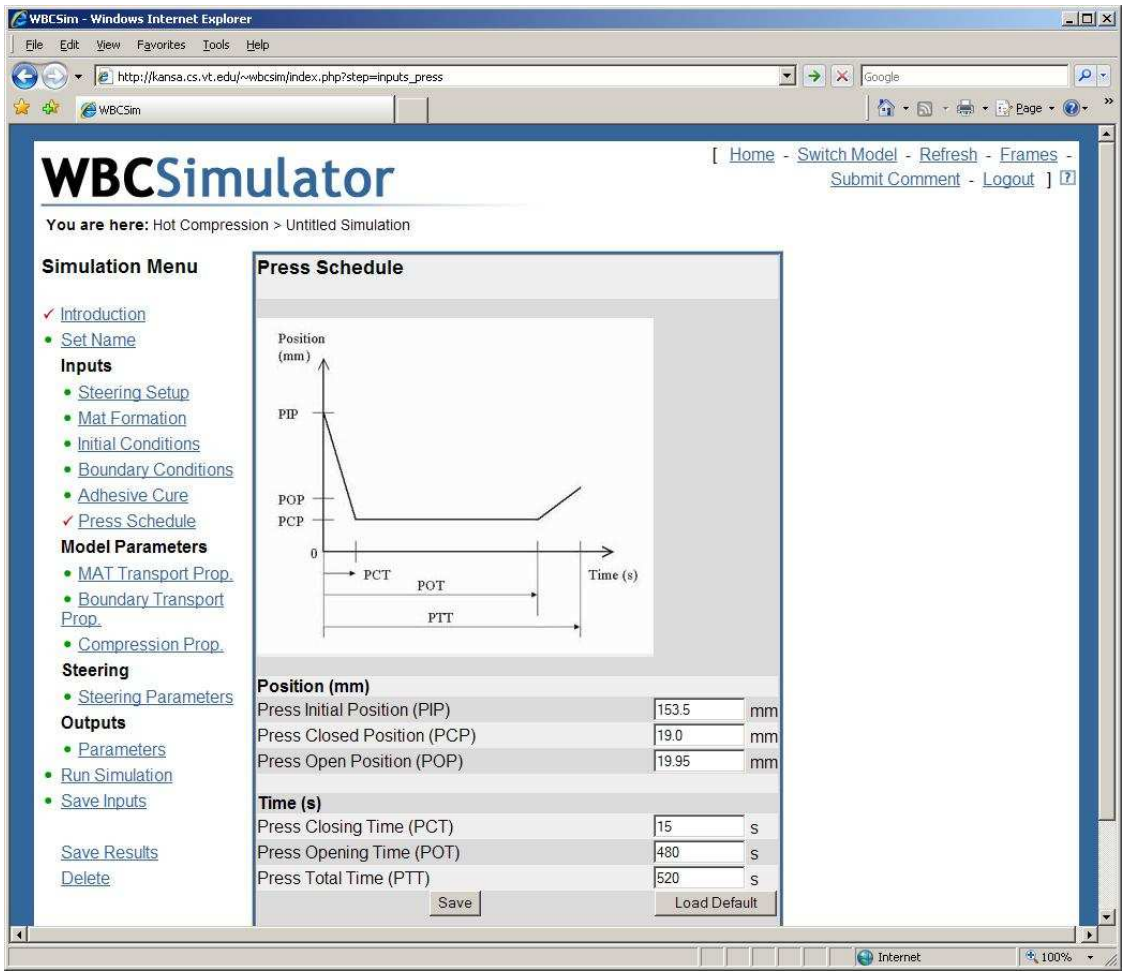


Fig. 5.1. The HC model user interface.

If a user decides to create a new simulation, he will actually see the specific model interface. There are some common features among all the model interfaces (see Fig. 5.1). Under the "WBCSimulator" title, there is a series of names and arrows to show the current navigation stage. In the left frame, there are many links corresponding the necessary steps

to run a particular simulation model. When these links are clicked, the corresponding Web pages will show up in the right frame. The first link is "Introduction" that explains the model behavior at an abstract level, and also gives general directions about how to use the model. The next link is "Set Name", which allows a user to describe the simulation. Following "Set Name", there are model-dependent links to allow a user to specify necessary simulation parameters in the right frame with any number of text boxes, dropdown lists, radio buttons, or plain buttons. At the bottom of left frame, there are links to "Run Simulation", "Save Inputs", "Save Results" and "Delete". The "Run Simulation" link will display a Web page to confirm and send the HTTP request to execute the proper PHP code with input values defined in the interface. This will cause a UNIX or Perl script to run compiled Fortran code, and any additional optimization or visualization codes. The "Save Inputs" link will only save the input values in the database as an incomplete simulation run, while the "Save Results" link will save both inputs and outputs in the database as a complete simulation run. The "Delete" link will delete the current simulation run from the database and any associated files. The "View Results" link will appear right under "Save Inputs" if a simulation is executed. The "View Results" will show the simulation results in both textual (normally tables of numbers) and graphical (VRML files, GIF files) forms.

The HC model (Fig. 5.1) interface differs from the other model interfaces by dividing the simulation parameters into four groups in the left frame menu: (1) model inputs, (2) model parameters (execution specification), (3) steering, and (4) model outputs. The first group contains links for specifying the parameters related to the steering setup, material, initial condition, boundary condition, adhesive cure, and press schedule. The second group defines the HC model execution by specifying mat transport properties, boundary transport properties, and compression properties. The third group provides intermediate results and options to steer the simulation during the run. The last group specifies the output, such as color or black and white, the time interval for generating frames for animation, and other parameters determining which data/image files are generated. This chapter focuses on demonstrating the computational steering feature via the HC model, as more steering specific features will be elaborated upon in Chapter 5.6 and 5.7. The reader can refer to [89] for more interface related details about the other models such as the RDS, RFP, OSB, and CMA models.

## 5.6 Architecture

The current software architecture of WBCSim follows the three-tier model described by Goel et al. [41], in which the tiers are (1) the client layer—user interface, (2) the server layer—Apache Web server and a PHP module, and (3) the developer layer—legacy

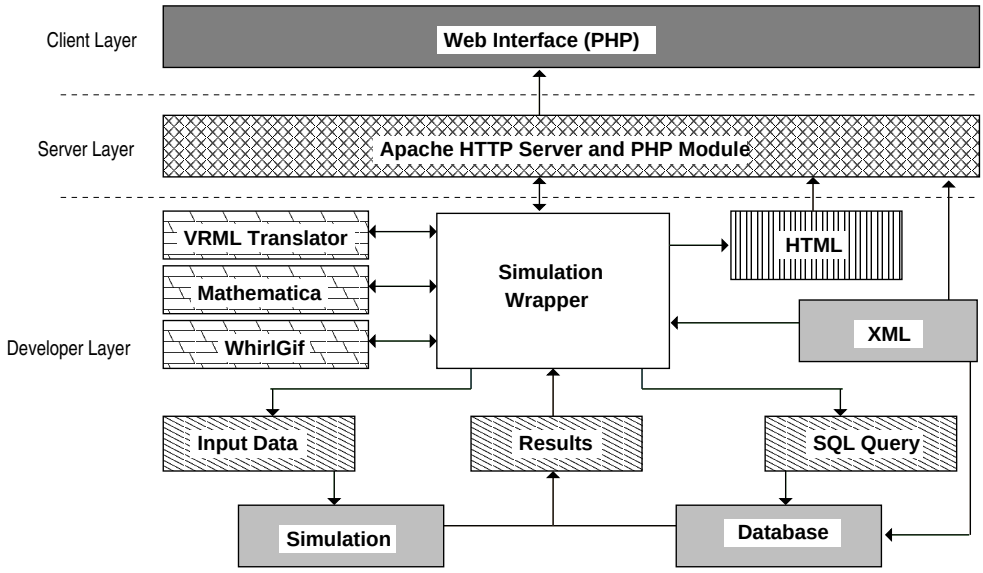


Fig. 5.2. WBCSim architecture overview.

simulation codes and various optimization and visualization tools running on the server. These layers are shown in Fig. 5.2.

### 5.6.1 Client Layer

The client layer is the only layer visible to end-users and typically the only layer running on the local machine. The main part of the client layer is the user interface, which is described in Chapter 5.5.

The user interface is generated from PHP and XML. As described earlier, each model has a XML datasheet, which uses a set of customized tags to describe the model. Readers can find some of these tags described in [89].

At the beginning of the PHP script, it reads the model specific XML datasheet and uses a XML parser to parse the information into an array structure. Then different parts of the PHP script use this array to construct interfaces, set parameter defaults, store user inputs, and assemble the inputs into a string (the corresponding simulation wrapper will feed this string to the Fortran simulations). Therefore, adding a parameter in the XML datasheet will simply add an element in the dynamic array. Then, the PHP code referring to this array will automatically update the interface when executed.

To support computational steering for the HC model, a parameter section called "Steering Setup" is added under the input section in the left frame menu (see Fig. 5.1). A user can specify the number of iterations for steering (how many times to steer), the maximum wait time per steering (when this time expires, the simulation will continue as normal), and the stepwise wait time (the simulation time intervals at which the simulation will check for steering inputs). There is another menu section called "Steering" added before

"Output Parameters". Through the "Steering" menu, a user can view the intermediate results (plots and data files) of a running simulation, and enter new values for the steerable parameters. Such interface changes are done quite easily by changing the corresponding HC XML datasheet. PHP code was altered to allow "Steering" to invoke a new Perl wrapper code running on the server, which generates plots for intermediate results and creates steering inputs to the HC Fortran program.

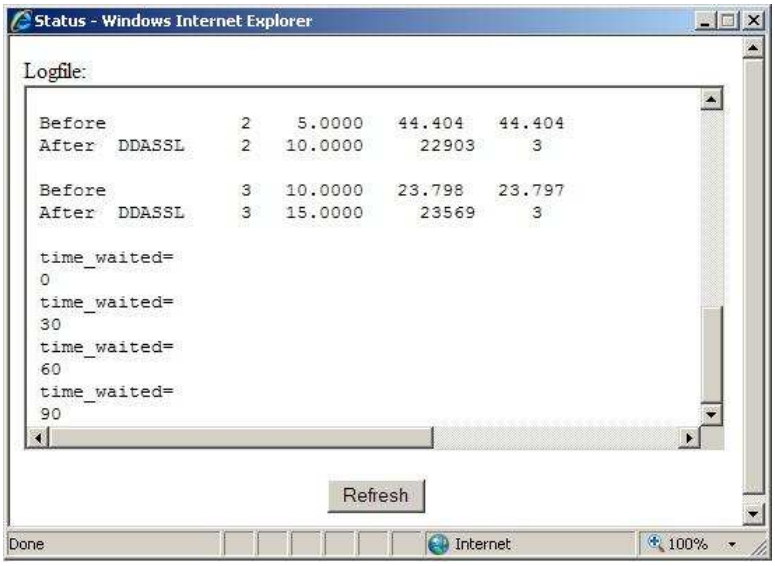


Fig. 5.3. The log window.

After starting a simulation in WBCSim, there is a log window where a user can monitor the progress of a simulation run (see Fig. 5.3). The Perl wrappers and the FORTRAN program all pipe their outputs to one log file per one simulation run. The log window simply takes such a log file to display in a browser window. Previously, a user had to click the "Refresh" button to refresh the contents of a log window manually. In order to better notify the user when it is time to steer, the log window is now programmed to refresh itself every five seconds. Moreover, for a long simulation run (such as 30 minutes), a user does not have to stare at the log window waiting for the right time to steer; instead, he can work on any other applications on his desktop. When the steering time arrives, this log window will pop out to the front of all the windows and take the focus. This alerts the user to go to the "Steering" menu to examine the intermediate results and steer.

The client layer also contains viewers for the visualization tools: the VRML translator in the HC model case. The reader can refer to [89] for more details. In summary, the client layer handles communication with the server layer by sending HTTP requests. When a user clicks a link, a URL appended with necessary action commands and parameters is sent to a Web sever in the server layer, where the corresponding PHP scripts, along with other

possible Perl scripts (including steering), visualization tools, and database operations, are executed.

### 5.6.2 Server Layer

By separating the legacy simulation codes from the user interface, the server layer functions as the key to how WBCSim can run a text-only application from a Web browser. The server layer consists of two components: an Apache HTTP server and a PHP module. Reference [89] describes why an old telnet connection method used in the original version of WBCSim [42] is untenable. The old telnet connection method creates a security hole by asking the client to open a special network port in order to communicate with the server. Currently, by requiring the standard Apache HTTP server with a PHP module, the server only needs the standard HTTP port 80, which is the standard and normally open/monitored port at the client computer for Web browsing.

The Apache HTTP Sever Project (http://httpd.apache.org/) is a collaborative software development effort aimed at creating a robust, commercial-grade, featureful, and freely-available source code implementation of an HTTP (Web) server. The project is jointly managed by a group of volunteers located around the world, using the Internet and the Web to communicate, plan, and develop the server and its related documentation. The January 2009 Netcraft Web Server Survey (http://news.netcraft.com/archives/web_server_survey.html) found that more than 52% of the Web sites on the Internet are using Apache, thus making it more widely used than all other web servers combined.

The PHP module serves as a plug-in to the Apache HTTP server. When the HTTP server receives a request to process a PHP file, the PHP module will parse and execute that PHP file and return HTML back to the client. This PHP module supports sessions. A session is a series of related interactions between a single client and the Web server, which can take place over an extended period of time. By using sessions, the PHP module can concurrently accept multiple requests from different clients and direct executions of multiple simulations.

The server layer is flexible enough to handle computational steering without any change.

### 5.6.3 Developer Layer

As its name suggests, the developer layer consists of legacy programs created by researchers to model WBC materials and manufacturing processes. These legacy programs are the heart of WBCSim. In general, WBCSim supports legacy programs written in any programming language as long as the program takes its input parameters from UNIX stdin. The input parameters can be a few numbers or a large data file. In particular, WBCSim supports five FORTRAN 77 and Fortran 90 simulation programs corresponding to the five models described in Chapter 5.3.

While each Fortran program has its own input format (stdin could be redirected to a file), the server layer communicates data with the developer layer via strings of parameters separated by white space (spaces, tabs, newlines). In order to cope with this string format, each legacy program is "wrapped" with a customized Perl script. The script receives this string of parameters from the PHP module at the server, and converts those parameters into an appropriate format for the Fortran program. Then the script calls the legacy program into action, feeding it the input, invoking any required optimization and visualization tools, packing all Fortran output in HTML files, and returning HTML files, first to the server layer, and then to the client layer. With this architecture, the developer layer is independent of the other layers, which makes the process of designing, and integrating new simulation codes relatively easy.

The changes to the developer layer required for computational steering are described now. The input parameters of the HC Fortran code are augmented with some initial steering setup parameters described in Chapter 5.6.1. Then, a generic steering module is inserted in the HC Fortran code. This steering module uses two nested loops. While the outer loop runs up to the number of steering interventions (called "iterations" in the steering input) that a user specified, the inner loop waits up to the maximum wait time per steering intervention (iteration). The inner loop checks for the steering input per stepwise wait time (such as every 30 seconds). If there is no steering input at all, the Fortran program will continue as usual after the maximum wait time expires. If the steering input shows no change from what the Fortran program currently has, the Fortran program will stop waiting and continue as usual. If the new steering inputs are different, the Fortran program will exit to start a new simulation with the new steering inputs (an abort/restart), or reset the steering parameters and continue the existing simulation run (normal steering). Since state variables may depend in a complicated way on the steering parameters, it can be highly nontrivial to modify a legacy scientific code to enable steering (i.e., to make the state consistent with the new steering parameters). Thus whether a legacy code is easily steerable depends on the physics and the code organization. A new Perl steer wrapper is also added to take the steering inputs from the Web interface, generate a steering input file for the Fortran program, and plot the intermediate results.

Shu et al. [90] described some additional operations in those wrappers to support the experiment management component. These other wrappers are associated with the save, retrieve, and compare functions from the user interface at the client layer. Like the simulation wrapper, these wrappers receive data from the server layer, and convert those input values into an appropriate format for the database.

Other than wrappers, the developer layer also includes optimization and visualization tools to maximize the simulation's value to the user [91].

## 5.7 Scenarios

Describing a typical usage scenario of computational steering in WBCSim is instructive. Consider research into the time of press closure in the hot compression process. The time of press closure is a mystery. It only lasts for 15 to 60 seconds, but much of the final panel characteristics are believed to be determined during this time. Heat and mass transfer are dependent on the press closing rate; the development of the density profile in the $z$-direction (along the smallest dimension of the laminate) begins during this time. How fast should the press be closed? Should it be closed at a constant rate or according to some kind of nonlinear closing schedule? Perhaps a stepwise closure (piecewise constant rate function) would be useful? There has been a lot of experimental work to determine the influence of press closure rate on panel properties. The problem is always to determine when the key events occur. Instead of physically testing a panel after it has completed the entire press cycle, a scientist can use the WBCSim computational steering feature to study this issue.

The scientist opens a browser window and launches the WBCSim Web page, which is dynamically generated from PHP and XML. The guest account is the default login option, which stores all the input and output data in a temporary directory that is purged periodically. The scientist selects the HC model from a list of all the available models by clicking "Use This Model as Guest". The scientist then launches the HC model interface (Fig. 5.1) by clicking "New Hot Compression".

Here the scientist needs to specify the various input parameters for steering setup, material, initial condition, boundary condition, adhesive cure, and press schedule. Fig. 5.1 shows the press schedule menu along with a diagram to illustrate various definitions of press position vs. press time. The initial press closure time (PCT) can be specified there. The scientist can also define various model parameters for mat transport properties, boundary transport properties, and compression properties. The steering parameters won't be available until the simulation is started. Finally, the scientist can specify the output styles, such as color or black and white, the time interval for generating frames for animation, and other parameters determining which data/image files are generated.

Next, upon clicking the link "Run Simulation", the input parameters (Boolean, numeric, or alphanumeric) are collected in the PHP code at the server as a long string (parameters are separated by white space). Then, the PHP module calls the HC model wrapper. Since the HC Fortran code has its own text-based user interface taking input from stdin, a temporary file is generated to contain all of the appropriate commands. Stdin is then redirected to this temporary data file for the simulation. The HC model wrapper then calls the HC Fortran code with this temporary file as stdin along with the properly formatted data file. When the simulation code is executing, the HC model wrapper monitors the simulation output stream for strings indicating execution milestones. The HC model wrapper uses the

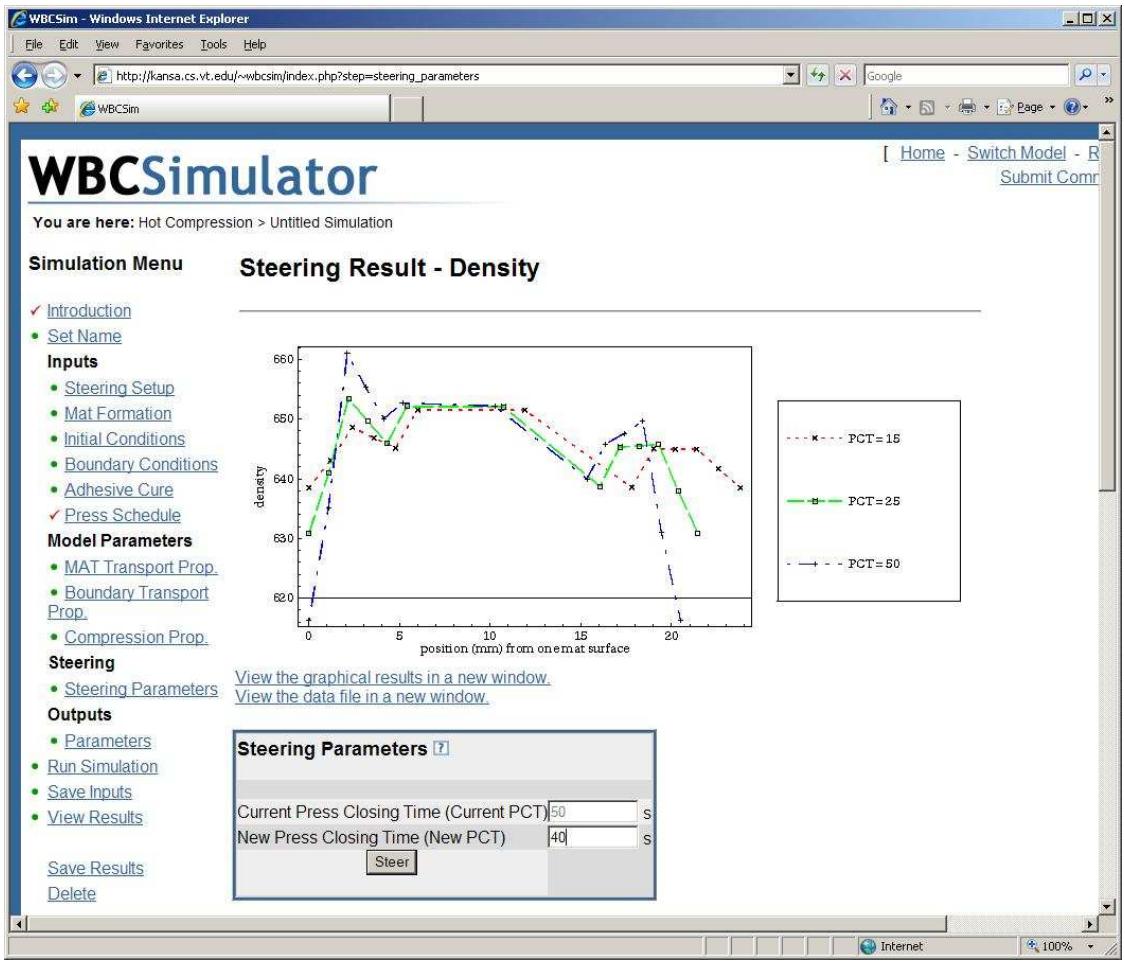


Fig. 5.4. The HC model steering interface.

standard error stream for storing these messages into a log file, which a user can access from the log window (Fig. 5.3).

When the simulation reaches the press closure time, the log window will come to the front and take the focus of the mouse. This prompts the scientist that it is time for steering. The scientist can then click the "Steering Parameters" link, which triggers a steering wrapper to take the intermediate result of the simulation and use Mathematica to plot density in the $z$-direction and adhesive effect in the $z$-direction (see Fig. 5.4). The scientist can enter a new press closure time and click the "Steer" button. This will terminate the existing simulation and start a new simulation with the new PCT. When the new simulation reaches this new PCT, the scientist can again enter the steering menu, which will plot the previous run along with the new simulation run to compare the effect of the two different PCTs. Fig. 5.4 shows a density plot of three iterations with PCT 15, 25, and 50. If the scientist is content with the current PCT, he can click the "Steer" button without entering anything; the steering wrapper will detect that there is no change, get out of the waiting loop and

continue the simulation as usual. This is just one possible steering scenario involving the press schedule.

When the simulation is complete and all the output data files are ready, the HC model wrapper calls Mathematica to read those data files and generate plots with various Mathematica commands such as ListContourPlot, ListPlot3D, MultipleListPlot, and Graphics3D. Mathematica also converts these internal graphics data structures into GIF format so that they can be viewed in a browser. Finally, the HC model wrapper embeds these GIF files in HTML files and returns these HTML files to the PHP module. These HTML files are passed back to the client, where a user can click "View Results" to access them. Fig. 5.5 shows the final simulation results, which are the 3D profiles of temperature, moisture content, adhesive cure index, and total pressure, in animations.

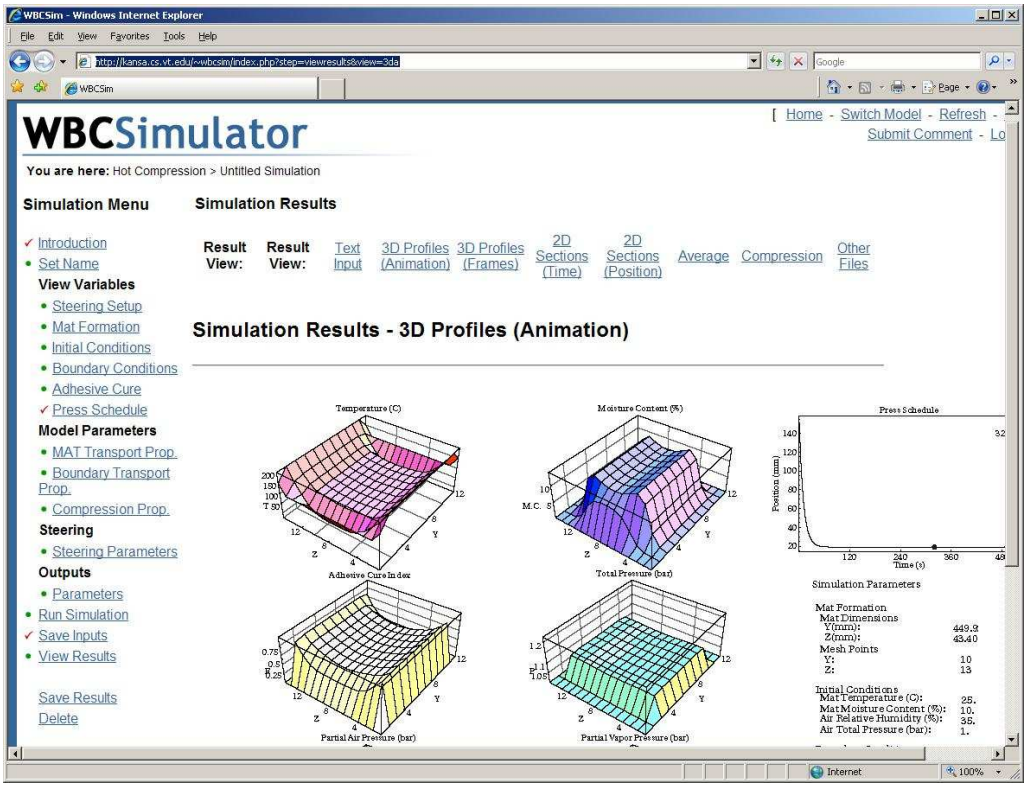


Fig. 5.5. The HC model final simulation results: 3D profiles of temperature, moisture content, adhesive cure index, and total pressure, in animations.

## 5.8 Computational Steering Instantiation in General

In summary, the set of changes and components required to provide a computational steering capability in WBCSim are: (1) design and add a very simple steering module at the legacy simulation code level, (2) provide a way to monitor simulation execution (alert users when it is time to steer), (3) add an interface to access and visualize simulation results, and perhaps to compare intermediate results across multiple steering attempts.

In theory, the steering module can be written in any programming language corresponding to the legacy simulation code (Fortran 90 for WBCSim). This steering module can take a list of the steering setup parameters as inputs (number of iterations, maximum wait time, step wait time). The two nested loop logic is simple in concept. While the outer loop runs up to the number of iterations, the inner loop waits up to the maximum wait time per iteration. The inner loop checks if the steering input is present per stepwise wait time. If there is no steering input after the maximum wait time expires, or the steering input shows no change, the module will exit the inner loop and proceed to the next iteration. If the new steering inputs are different, the module should perform the steering by either exiting to start a new simulation with the new steering inputs (an abort/restart), or reset the steering parameters and continue the existing simulation run (normal steering). The choice of these two execution paths is completely dependent on the physics (fundamental science) and the organization of the legacy simulation code. For example, the legacy simulation code could have many arrays declared throughout various parts of the code with dynamic dimensions. These dimensions could be calculated on the fly based on various conditions. If the steering parameters affect the dimensions of these arrays, it may not be easy to deallocate and reallocate these arrays, since certain parts of the code may need to be executed again to calculate these dimensions. Therefore, instead of making massive changes to the legacy code to support steering, a steering module can simply allow a simulation to exit and restart.

More importantly, adding steering requires significant collaboration between the PSE program developer and the domain scientists. That is because only the domain scientists fully understand the physics (fundamental science) and organization of the legacy simulation code. The domain scientists have to decide what parameters they would like to steer, what intermediate outputs and format they would like to see, and how to visualize such intermediate outputs. The domain scientists may need to pinpoint exactly where the steering module needs to be called in the simulation code and where the simulation needs to resume after the steering.

It is very critical to provide a way to monitor the simulation execution, especially a time consuming simulation. In addition, a user needs to be informed or alerted when the steering time arrives. Depending on the run time of the simulation, a user may not monitor the progress of the simulation the entire time. The user may work on other things on the computer or not even at his desk. For the WBCSim HC model, a typical simulation run lasts from 45 minutes to over an hour. Thus, the log window is sufficient as this log window would come to the front and take the focus when the steering time arrives. The mechanics of the log window is described in Chapter 5.6.1. The log window simply checks if there is any output from the steering module, and then pops to the front at the steering time. For longer simulations (say overnight simulations), other means should be considered such as email notification, or a predefined set of the steering inputs (different from a regular

simulation run, since the legacy code may not have supported intermediate inputs). The number of iterations, maximum wait time, and stepwise wait time need to be calibrated accordingly based on the actual scenarios.

A user needs an interface to access the intermediate results from the steering and steer the simulation if he chooses to. The intermediate results can be the raw data (data files) or visualizations (in 2D or 3D) or both. These results should provide meaningful clues for the user to determine the course of the simulation. It is very useful to be able to compare the intermediate results from several steering attempts (iterations). By examining these results, the user can apply his expert knowledge to steer the simulation to explore "what if" scenarios and potentially deepen his understanding of principles underlying the phenomenon. As effects due to changes in parameters become more instantaneous, the cause-effect relationships could become more evident. Chapter 5.7 describes the interface for the WBCSim HC model, which provides links to the actual data files for the intermediate results, plots comparing multiple steering iterations, and simple text fields to allow the user to specify new values for the steerable parameters. Depending on user needs and the computing environment, many creative steering interfaces can be designed and implemented.

Obviously, a certain amount of work needs to be done to enable computational steering and add additional steering points. Also, there is execution overhead, as the simulation will run longer depending on the number of steering iterations, and the wait time per iteration. However, with the ability to change direction in the middle of the simulation, computational steering can effectively reduce overall computational time by reducing the number of nonproductive simulation runs, thereby making research more efficient, and possibly lead to new product designs from increased understanding of the physical phenomena being modeled. In general, computational steering has the potential to revolutionize computer simulation experiments by allowing scientists to interactively steer a simulation in time and/or space, and concentrate more on the science than on operating the computer. Computational steering supersedes the traditional simulation mode of many long running experiments, which may not produce the desired results; instead, it allows the scientist to be placed at the center of a computation and respond to simulation results as they occur by interactively manipulating the input parameters.

## 5.9 Future Work and Conclusions

While WBCSim can be developed on several different fronts: experiment management (EM), collaboration support, high performance computing, and XML integration [91] [89], the development track for computational steering intends to streamline the process of adding a steering point, add support for multiple steering points, and add database support.

### 5.9.1 Streamline the Process of Adding a Steering Point

Currently, adding a steering point requires considerable work and interaction between a wood scientist and a program developer. The wood scientist needs to identify what parameters within a model simulation he would like to steer, when the steering would occur, what intermediate results he would like to see, in what format, and what the simulation behavior would be after steering (restart or continue with the new steered parameters). Then the program developer would insert such a steering module into the simulation code at the development level, and make the necessary interface changes at the interface level. A better approach would be standardizing the steering module so that a wood scientist could easily call such a steering module anywhere within the simulation code, and make the simulation code changes himself. Then, a wood scientist could simply alter the model specific XML sheet to change the interface to include a new steering point.

### 5.9.2 Add Multiple Steering Points

There could be multiple steering points for a given simulation. For example, for the hot compression model, a wood scientist could be interested not only in the press closure time, but also in the gas pressure. The build-up of gas pressure inside the mat during the hot compression process must be carefully monitored to balance the conflicting goals of minimum time in the press and adequate cure of the adhesive. Most of the gas pressure is due to steam. The steam is useful early in the press cycle because it assists with heat transfer to the core of the mat. However, the gas pressure must be reduced before the press is opened, otherwise the rapid expansion will cause a rupture of the panel. It would be informative to see how the press closure time affects the gas pressure by using two steering points. This requires the ability to allow two steering points to interact with each other such as moving the execution of a simulation from one early steering point to a later steering point, or restarting from an earlier steering point (rather than restarting the entire simulation).

### 5.9.3 Add Database Support for Computational Steering

The existing experiment management system can store either incomplete simulation inputs or a complete simulation run (both inputs and outputs). It would be useful to provide the ability to store intermediate results for computation steering. With this capability, a wood scientist could even compare one steering run (consisting of multiple simulation runs) to another steering run (consisting of other multiple simulation runs). Another user might also be interested in viewing or modifying an existing steering run.

### 5.9.4 Conclusions

WBCSim has evolved steadily from a prototype PSE intended as a tool for computer science PSE research and a Web-based interface for a few legacy computer programs, to a

manufacture-oriented near commercial quality PSE that is seriously used by wood science researchers in industry and academia. Since interesting computational capabilities are still lacking [91], WBCSim will remain an object of computer science research for some time to come.

Yet the program's interfaces, models, and output visualizations are now good enough to be used as production tools by wood scientists. The directions in which computer scientists would like to take WBCSim (collaboration, data mining, grid computing) are quite different from the directions that wood scientists would prefer for WBCSim (new models, refining existing models, more interface and visualization options). The present layered architecture of WBCSim supports these divergent development directions well.

With the addition of a computational steering capability, WBCSim allows a wood scientist to interactively explore a problem gaining insight into the physical mechanisms being modeled, effectively reduce computational time, make research more efficient, and better pursue new product design. It allows the scientist to be placed at the heart of a computation, and apply his expert knowledge to respond to simulation results as they occur by interactively manipulating the model parameters. Moreover, the experience with WBCSim suggests that computational steering can be achieved in any PSE or any other computing environment embedded with legacy simulation code.

The original stated goal [41] of WBCSim was to provide "an integrated set of facilities allowing wood scientists to concentrate on high-level problem solving rather than on low-level programming details and application scheduling/execution, allowing users to define, record, and modify problems, and visualize and optimize simulation results". This goal is definitely closer when wood scientists are provided with a computational steering capability.

# Chapter 6: Summary of Changes and Enhancements

This chapter summarizes the changes and enhancements throughout the years. It compares the early version of WBCSim (in 2000) with the current state of WBCSim (in 2009). The discussion is based on the three implementation layers of WBCSim.

## 6.1 Client Layer

The client layer is the only layer visible to end-users and typically the only layer running on the local machine.

In 2000, WBCSim had a Java applet interface. Java applets require a user to have a Java-enabled browser. Since different browsers (brands or versions) may have different versions of the Java plug-in, the developers stuck to a very early version of Java (Java 1.0) to ensure that the applets worked on all user computers (especially old computers at industrial manufacturing sites). This old version of Java not only has a very limited set of graphical user interface APIs, but also is very buggy. Moreover, this old Java applet interface has many pop-up windows in a very strict hierarchy (parent and child relationship). To run one simulation, a user may have five or six Java applet windows on his desktop overlapping each other. If a user accidentally closes the wrong window, the whole application could crash.

In 2003, the Java applet interface (Browser plug-in technology) was replaced by a PHP (server scripting technology) interface. PHP is the acronym for Hypertext Preprocessor, which is an open source, server-side, HTML embedded scripting language used to create dynamic Web pages. Other similar server scripting technologies are JSP (Sun) and ASP (Microsoft). Upon any HTTP request, the server runs PHP scripts and sends plain HTML code back to the client, where no plug-in is needed. While implementing this new PHP interface, almost all the old pop-up windows were incorporated into frame-based Web pages. A user can run a simulation within two browser windows (a main window, and an optional log window to monitor execution). Studies with student users prove the new interface significantly improves WBCSim's usability and is much more user friendly.

Around 2004, XML was used with PHP to generate the WBCSim Web interface. As described earlier, each model has a XML datasheet, which uses a set of customized tags to describe the model. The beginning of the PHP script reads the model specific XML datasheet and uses a XML parser to parse the information into an array structure. Then different parts of the PHP script use this array to construct interfaces, set parameter defaults, store user inputs, and assemble the inputs into a string (the corresponding simulation wrapper will feed this string to the Fortran simulations). Adding a parameter in the XML datasheet will simply add an element in the dynamic array. Then, the PHP code referring to this array will automatically update the interface when executed. Appendix E shows the complete XML datasheet for the hot compression model.

In 2008, there were some additional changes to XML and PHP scripts to support a computational steering interface. The addition was mainly done through changing the corresponding XML datasheet. The optional log window was enhanced to refresh itself and pop to the front when it is time to steer the simulation.

## 6.2 Server Layer

By separating the legacy simulation codes from the user interface, the server layer functions as the key to how WBCSim can run a text-only application from a Web browser.

In 2000, the server layer of WBCSim consisted of two components: a telnet server and a custom shell to facilitate server-client communication. The telnet server is not a replacement for a standard telnet server; it implements only enough of the telnet protocol to work with the WBCSim telnet client. The telnet server supports guest and regular logins and all of the operations provided by the previous Javamatic server, which could direct execution of multiple simulations and concurrently accept multiple requests from the client [41]. This telnet server manages the pool of connections and provides a means for executing remote commands on the server. Yet Another PSE Shell (YAPS) is a simple Perl script that the client invokes when it logs in via a telnet connection. The client talks to this shell instead of the UNIX login shell of the account. The common commands supported by YAPS were "store", "remove", "load", and "clean". All these commands were file system based and performed upon simulation related files. The details are described in Chapter 2.5.3. There was no command to run a simulation because the simulation wrapper can be invoked directly by YAPS based on the requests coming from the telnet client.

In 2003, an Apache HTTP server and a PHP module replaced the old telnet connection and the custom shell. The old telnet connection method creates a security hole by asking the client to open a special network port in order to communicate with the server. In today's world of highly tightened Internet security, it is problematic to ask a user to open/monitor a special port on his computer in order to use WBCSim. A standard Apache HTTP server with a PHP module only requires the standard HTTP port 80, which is the standard and normally open/monitored port at the client computer for Web browsing.

The PHP module serves as a plug-in to the Apache HTTP server. When the HTTP server receives a request to process a PHP file, the PHP module will parse and execute that PHP file and return HTML back to the client. This PHP module supports sessions. A session is a series of related interactions between a single client and the Web server, which can take place over an extended period of time. By using sessions, the PHP module can concurrently accept multiple requests from different clients and direct executions of multiple simulations.

The existing server layer with Apache and a PHP module has been proved to be very flexible and has not been changed since 2003 while XML technology and a computational

steering capability were added. The PHP scripts were changed to use XML datasheets in 2004.

## 6.3 Development Layer

The developer layer consists of legacy programs created by researchers to model WBC materials and manufacturing processes. These legacy programs are the heart of WBCSim. In general, WBCSim supports legacy programs written in any programming language as long as the program takes its input parameters from UNIX stdin. The input parameters can be a few numbers or a large data file. In particular, WBCSim supports five FORTRAN 77 and Fortran 90 simulation programs corresponding to the five models described in Chapter 2.3.

While each Fortran program has its own input format (stdin could be redirected to a file), the server layer communicates data with the developer layer via strings of parameters separated by white space. Therefore, in order to cope with this string format, each legacy program is "wrapped" with a customized Perl script. The script receives this string of parameters from the server, and converts those parameters into an appropriate format for that Fortran program. Then the script calls the legacy program into action feeding it the input, invokes any required optimization and visualization tools, and finally packs all the Fortran output in HTML files and passes their URLs to the server layer, and then to the client layer. With this architecture, the developer layer is independent of the other layers, which makes the process of designing, and integrating, new simulation codes relatively easy.

Other than simulation code and wrappers, the developer layer also includes optimization—DOT (Design Optimization Tool), and various visualization tools such as VRML and Mathematica to maximize the simulation's value to the user. In particular, since 2001, WhirlGif has been used to create animation for some WBCSim time series results (such as the CMA model's stress and strain distributions). WhirlGif is a small UNIX utility for generating a single animated GIF image file from a sequence of GIF image files. These are described in Chapters 2.7 and 2.8.

Since 2002, with the addition of the experiment management component, the wrapper performs one more operation before it calls the Fortran program to execute the simulation. The wrapper checks if a simulation with the current set of inputs has already been executed. If the simulation was executed before and the results were saved, the wrapper constructs the output data files from the database instead of running the Fortran program. The wrapper then feeds the output data files into any required optimization and visualization tools. This operation is transparent to the user; the response time is significantly reduced if the output is found in the database. Appendices A, B, and C illustrate these Perl wrapper codes.

Postgres is the database system being used. Postgres is a sophisticated object-relational database management system (DBMS) that supports almost all SQL constructs, including

subselects, transactions, and user-defined types and functions. Moreover, it is open source. In this architecture, the wrappers communicate with Postgres by Pgsql-perl5, which is an interface between Perl version 5 and Postgres. This interface uses the Perl version 5 application programming interface for C extensions, which calls the Postgres programmer's interface LIBPQ.

In early 2003, the Fortran programs were rigorously tested under several UNIX environments—the old DEC Alpha, SGI, IBM, and Sun—for portability. Later, some wrappers were converted to use the XML datasheet instead of using a hardcoded set of input parameters.

In 2008 and 2009, changes were made to the Fortran 90 code and wrappers to support computational steering. A description of the operation of computational steering follows. The input parameters of the HC Fortran code are augmented with some initial steering setup parameters described in Chapter 5.6.1. Then, a generic steering module is inserted in the HC Fortran code. This steering module uses two nested loops. While the outer loop runs up to the number of steering interventions (called "iterations" in the steering input) that a user specified, the inner loop waits up to the maximum wait time per steering intervention (iteration). The inner loop checks for the steering input per stepwise wait time (such as every 30 seconds). If there is no steering input at all, the Fortran program will continue as usual after the maximum wait time expires. If the steering input shows no change from what the Fortran program currently has, the Fortran program will stop waiting and continue as usual. If the new steering inputs are different, the Fortran program will exit to start a new simulation with the new steering inputs (an abort/restart), or reset the steering parameters and continue the existing simulation run (normal steering). A sample steering module is presented in Appendix F. A new Perl steer wrapper is also added to take the steering inputs from the Web interface, generate a steering input file for the Fortran program, and plot the intermediate results.

## 6.4 Cross Layers

Obviously these changes are not isolated and limited to one layer, but rather connected and applied to several layers. Such cross layer changes include adding more advanced simulation models, providing features to save and retrieve simulation runs from the database, allowing users to search for and compare simulation runs, using XML to unify the implementation layers, and enabling computational steering.

## 6.5 WBCSim Usage and Benefits to Various Users

WBCSim has been used by students in several wood science classes, by graduate students and faculty at VPI&SU, and by researchers at several forest products manufacturing companies. WBCSim has been demonstrated at several wood science conferences such as PETE ((Panel & Engineered Wood Technology Conference & Expo) and SEMI (Sustainable

Engineered Materials Institute) program reviews. It has generated interest from both academia and industry. WBCSim has the capability to support production accounts (separate from guest accounts). However, to date, all the users are still using guest accounts to access WBCSim.

The people using WBCSim can be divided into three groups: 1) regular users who know how to use WBCSim from the Web interface, but don't know anything about the underlying programming details, 2) advanced users/modelers who, in addition, know/own the core simulation programs, and visualization code such as the Mathematica scripts, and 3) programmers who know/own the rest of the code such as the PHP interface code, the Perl simulation wrappers, and the database; they also maintain the WBCSim Web server and database server.

Compared to WBCSim circa 2000, the present WBCSim is a functional production quality PSE. Regular users and advanced users/modelers today enjoy many new features that the old WBCSim did not have. There are now two new models, MAT and HC, that simulate the critical mat formation and hot compression processes, and which are much more advanced, complex, and interesting than the older models. The old CMA model has also been completely redesigned (both the core simulation and the interface). Compared to the old WBCSim file based saving and retrieving methods, the new DBMS component allows users to save, retrieve, search, and compare simulation runs. The new DBMS component not only saves the unique inputs and outputs into the database (making links for any duplicate runs), it may also skip the actual lengthy simulation run if it can find the simulation results in the database. This is transparent to users and significantly reduces the run time. For example, the mat formation simulation may run seven minutes when executing the Fortran simulation by default, but it may return the results in one minute if it can find the simulation results in the database.

Moreover, with the new computational steering capability, users can pause, examine, and compare the intermediate results from a simulation. Contrasted with the traditional way of running a lengthy simulation to see the result at the end, computational steering can leverage the user's expert knowledge on the fly (during the simulation run) and provide new insights and new product design opportunities. For example, it may take five minutes to reach a steering point, and steering may take three minutes (a user gets prompts to steer, views the intermediate results, and then steers). The full simulation run may take 45 minutes. Say a user wants to run three experiments, of which two turn out to be uninteresting (apparent at the first steering point). In the old WBCSim, it may take $45 * 3 = 135$ minutes to run all the simulations. In the new WBCSim, it may take $(5 + 3) * 3 + (45 - 5) = 64$ minutes. Moreover, a HCI usability study (conducted among undergrad student users at VPI&SU) showed the current left-right frame WBCSim interface is much easier to navigate and use (compared to the old WBCSim Java applet interface with many popup windows).

For programmers, the new WBCSim is much easier to change and maintain. Instead of maintaining an ancient Java applet API and telnet connection, programmers are now working with a very popular open source server scripting language PHP, and Apache. The new interface is designed using object-oriented concepts. There is a WBCSim object for the entire Web interface and a base object for all the simulation models. Then, each simulation has its individual object, which inherits various common methods from the base object. Since 2004, XML has been used to describe a simulation model and unify various implementation layers. Before the XML implementation, a simple change such as adding a new input parameter in an existing parameter section required a programmer to work with a modeler to pinpoint several places and make changes in WBCSim, such as the PHP interface code, the Perl simulation wrapper, a Mathematica script, and the database. Today, most of these layers interface with the same XML datasheet. Making a simple change to the XML sheet, adding this parameter in the corresponding parameter section, causes cascading changes at various implementation layers. For example, before the XML implementation, finding/changing PHP interface and Perl simulation wrappers could take hours. Today, with XML, a programmer or even a modeler can make such a change in the XML sheet within a few minutes.

Finally, the WBCSim code is much more organized: PHP, Perl, Fortran simulation, temporary data files are grouped into their corresponding folders. By converting the legacy Fortran code to ANSI standard Fortran 90 and adhering to stable technologies like PHP, Perl, and Apache, the WBCSim code has also become much more portable, having now been ported from OSF (DEC Alpha) to Solaris (SUN Sparc) to Linux (Intel x86).

# Chapter 7: Future Work and Conclusions

Recent changes and enhancements have been made to the simulation model codes and the experiment management component. The latter can be divided into three levels: 1) data management, 2) change management, and 3) execution management.

At the data management level, anything related to an experiment should be stored and documented. Chapter 3 described such a component where a DBMS is used to store the simulation runs. Then various high level interfaces are provided to allow users to save, retrieve, search, and compare these simulation runs.

At the change management level, a scientist should only focus on how to solve a problem in the experiment domain. Aside from running experiments, a scientist may only consider how to define a new model, how to modify an existing model, and how to interpret an experiment result. The current architecture of WBCSim makes adding a new model relatively easy and straightforward, as described in Chapter 2. By using XML described in Chapter 4, changing an existing model is intuitive and fast.

At the execution management level, how an experiment is executed is the main concern. By providing a computational steering capability, a scientist can pause, examine, and compare the intermediate results from a simulation. Contrasted with the traditional way of running a lengthy simulation to see the result at the end, computational steering can leverage the user's expert knowledge on the fly (during the simulation run) and provide new insights and new product design opportunities.

## 7.1 Future Work—Existing Research Topics

The existing research topics can be pursued deeper and/or generalized. Database support can be applied to all models. A more DBMS centric architecture (described in Chapter 3.7.2) can be explored. XML datasheets can be utilized to define the database table schema. Making a XML editing, verification, and submission interface can be very helpful for a wood scientist to alter WBCSim himself. Archiving/storing XML can effectively provide version control for WBCSim. The process of adding a steering point in a simulation can be streamlined. Adding multiple steering points may provide additional flexibility to simulation execution. Steering parameters and intermediate results can be saved into the database. Moreover, database support, XML, and computational steering can be generalized to support other PSEs or computational environments.

## 7.2 Future Work—Other Research Topics

There are also other interesting research topics such as collaboration support and high performance computing. Collaboration support would allow multiple geographically distributed users to jointly modify the same model interface, and concurrently view and annotate the same simulation output. Other features indirectly supporting collaboration

are user control over where their data is stored (locally on the client machine vs. remotely on the server), and support for exchanging data with other researchers. By utilizing either parallel or distributed computing, WBCSim can reduce the turnaround time to minutes, making interactive use feasible even for complex three-dimensional models. One approach can be using a cluster as a compute engine for WBCSim, transparent to the user, when a requested simulation can be estimated to require such resources.

## 7.3 Conclusions

WBCSim has evolved steadily from a prototype PSE, intended as a tool for computer science PSE research and a Web-based interface to a few legacy computer programs, to a manufacture-oriented near commercial quality PSE that is seriously used by wood science researchers in industry and academia. Since interesting computational capabilities are still lacking (Chapters 7.1–7.2), WBCSim will remain an object of computer science research for some time to come. Yet the program's interfaces, models, and output visualizations are now good enough to be used as production tools by wood scientists. The directions in which computer scientists would like to take WBCSim (collaboration, data mining, grid computing) are quite different from the directions that wood scientists would prefer for WBCSim (new models, refining existing models, more interface and visualization options). The present layered architecture of WBCSim supports this dichotomy well.

With the addition of the database support, WBCSim now has a much more efficient tool to manage its simulation execution and experiment data. With the addition of XML technology, WBCSim is now considerably easier to change and maintain. With the addition of a computational steering capability, WBCSim allows a wood scientist to interactively explore a problem gaining insight into the physical mechanisms being modeled, effectively reduce computational time, make research more efficient, and better pursue new product design.

The original stated goal [41] of WBCSim, to provide "an integrated set of facilities allowing wood scientists to concentrate on high-level problem solving rather than on low-level programming details and application scheduling/execution, allowing users to define, record, and modify problems, and visualize and optimize simulation results," now seems much closer to fruition.

## REFERENCES


[1] A. Ailamaki, Y.E. Ioannidis, M. Livny, "Scientific workflow management by database management", in *Proc. of 10th International Conference on Scientific and Statistical Database Management*, Capri, Italy, 1998.

[2] N.A. Allen, K.C Chen, C.A. Shaffer, J.J. Tyson, L.T. Watson, "Computer evaluation of network dynamics models with application to cell cycle control in budding yeast", *IEE Systems Biology*, 153 (2006) 13–21.

[3] N.A. Allen, C.A. Shaffer, M.T. Vass, N. Ramakrishnan, L.T. Watson, "Improving the development process for eukaryotic cell cycle models with a modeling support environment", *Simulation*, 79 (2003) 674–688.

[4] A.L. Ames, D.R. Nadeau, J.L. Moreland, *VRML 2.0 Sourcebook, 2nd ed*, John Wiley & Sons, Inc.: New York, NY, 1996.

[5] S. Balay, B. Gropp, L. Curfman McInnes, B. Smith, "A microkernel design for component-based parallel numerical software systems", Technical Report ANL/MCS-P727-0998, Argonne National Laboratory, 1998.

[6] L. Beca, "Building collaborative problem-solving environments as shared places", in *34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, Maui, Hawaii, 2001.

[7] L. Beca, G. Cheng, G.C. Fox, T. Jurga, K. Olszewski, M. Podgorny, P. Sokolowski, K. Walczak, "Java enabling collaborative education, health care and computing", *Concurrency Practice and Experience*, 9 (1997) 521–534.

[8] R.F. Boisvert, J.R. Rice, *Solving Elliptic Problems Using ELLPACK*, Springer-Verlag: New York, NY, 1985.

[9] A.J. Bolton, P.E. Humphrey, "The hot-pressing of dry-formed wood-based composites. Part I. A review of the literature, identifying the primary physical process and the nature of their interaction", *Holzforschung*, 42 (1988) 403–406.

[10] A.J. Bolton, P.E. Humphrey, P.K. Kavvouras, "The hot-pressing of dry-formed wood-based composites. Part III. Predicted vapour pressure and temperature variation with time, compared with experimental data for laboratory boards", *Holzforschung*, 43 (1989) 265–274.

[11] A.J. Bolton, P.E. Humphrey, P.K. Kavvouras, "The hot-pressing of dry-formed wood-based composites. Part IV. Predicted variation of mattress moisture content with time", *Holzforschung*, 43 (1989) 345–349.

[12] A.J. Bolton, P.E. Humphrey, P.K. Kavvouras, "The hot-pressing of dry-formed wood-based composites. Part VI. The importance of stresses in the pressed mattress and their relevance to the minimization of pressing time, and the variability of board properties", *Holzforschung*, 43 (1989) 406–410.

[13] M.E. Bowen, "Heat transfer in particleboard during hot pressing", Ph.D. Dissertation, Colorado State University, Fort Collins, CO, 1970.

[14] R. Bramley, D. Gannon, T. Stuckey, J. Villacis, E. Akman, J. Balasubramanian, F. Breg, S. Diwan, M. Govindaraju, "The linear system analyzer", Technical Report TR-511, Dept. of Computer Science, Indiana University, Bloomington, IN, 1998.

[15] KW. Brodlie, S. Mason, M. Thompson, M. Walkley, J.D. Wood, "Reacting to a crisis: benefits of collaborative visualization and computational steering in a grid environment", in *Proceedings of UK e-Science All Hands Conference*, Sheffield, UK, 2002.

[16] J.M. Brooke, P.V. Coveney, J. Harting, S. Jha, S.M. Pickles, R.L. Pinning, A.R. Porter, "Computational steering in RealityGrid", in *Proceedings of the 22nd International VLDB Conference*, (Glasgow, England, 2003) 179–183.

[17] J. Brooke, T. Eickermann, U. Woessner, "Application steering in a collaborative environment", in *Proceedings of Supercomputing*, San Francisco, California, USA, 2003.

[18] T. Burnett, C. Chaput, H. Arrighi, J. Norris, D.J. Suson, "Simulating the Glast satellite with Gismo", *IEEE Computing in Science and Engineering*, 2 (2000) 9–18.

[19] M. Cannataro, C. Comito, F. Schiavo, P. Veltri, "PROTEUS: a grid based problem solving environment for Bioinformatics", in *Workshop on DataMining Ontology for Grid Programming (KGGI 03)*, Halifax, Canada, 2003.

[20] L.M.H. Carvalho, C.A.V. Costa, "Modeling and simulation of the hot-pressing process in the production of medium density fiberboard (MDF)", *Chem. Eng. Comm.*, 170 (1998) 1–21.

[21] H. Casanova, J. Dongarra, "NetSolve: a network server for solving computational science problems", *International Journal of Supercomputer Applications and High Performance Computing*, 11 (1997) 212–223.

[22] A.C. Catlin, C. Chui, C. Crabill, E.N. Houstis, S. Markus, J.R. Rice, S. Weerawarana, "PDELab: an object-oriented framework for building problem solving environments for PDE based applications", in *A. Vermeulen, ed., 2nd Object-Oriented Numerics Conference*, (Rogue Wave Software, Corvallis, OR, 1994) 79–92.

[23] A. Chabert, E. Grossman, L. Jackson, S. Pietrovicz, "NCSA Habanero: synchronous collaborative framework and environment", in *Conference Supplement of the Fifth European Conference on Computer-Supported Cooperative Work (ECSCW'97)*, (nc Lancaster L, UK, 1997) 7–8.

[24] J.X. Chen, X. Fu, "Integrating physics-based computing and visualization: modeling dust behavior", *IEEE Computing in Science and Engineering*, 1 (1999) 12–16.

[25] J. Czyzyk, J.H. Owen, S.J. Wright, "NEOS: optimization on the Internet", Technical Report OTC-97/04, Argonne National Laboratory, 1997.

[26] D. Dabdub, R. Manohar, "Performance and portability of an air quality model", *Parallel Computing*, 23 (1997) 2187–2200.

[27] C. Dai, P.R. Steiner, "Spatial structure of wood composites in relation to processing and performance characteristics. Part III. Modeling the formation of multi-layered random flake mats", *Wood Science and Technology*, 28 (1994) 229–239.

[28] T.L. Disz, R. Evard, M.W. Henderson, W. Nickless, R. Olson, M.E. Papka, R. Stevens, "Designing the future of collaborative science: Argonne's Futures Laboratory", *IEEE Parallel & Distributed Technology*, 3 (1995) 14–21.

[29] S. Dong, J.A. Insley, N.T. Karonis, M.E. Papka, J. Binns, G.E. Karniadakis, "Simulating and visualizing the human arterial system on the TeraGrid", *Future Generation Computing Systems*, 22 (2006) 1011–1017.

[30] W.R. Dyksen, C.J. Ribbens, "Interactive ELLPACK: an interactive problem solving environment for elliptic partial differential equations", *ACM Transactions on Mathematical Software*, 13 (1987) 113–132.

[31] R. Dymond V. Lohani, D. Kibler, D. Bosch, E.J. Rubin, R. Dietz, J. Chanat, C. Speir, C.A. Shaffer, N. Ramakrishnan, L.T. Watson, "From landscapes to waterscapes: a PSE for landuse change analysis", *Engineering with Computers*, 19 (2003) 9–25.

[32] G. Eisenhauer, W. Gu, K. Schwan, N. Mallavarupu, "Falcon—toward interactive parallel programs: the on-line steering of a molecular dynamics application", in *Proceedings of The Third International Symposium on High-Performance Distributed Computing*, San Francisco, California, USA, 1994.

[33] B. Ford, R.M.J. Iles, "The what and why of problem solving environments for scientific computing", in *B. Ford and F. Chatelin, ed., Problem Solving Environments for Scientific Computing*, (Elsevier Science Publishers, North-Holland, 1985) 3–18.

[34] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, J. Von Reich, *The Open Grid Services Architecture Version 1.5*, Global Grid Forum (GGF) Document GFD.80, 2006.

[35] I. Foster, J. Voeckler, M. Wilde, Y. Zhao, "Chimera: A virtual data system for representing, querying, and automating data derivation", in *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, (Edinburgh, Scotland, 2002) 37–46.

[36] J.C. French, A.K. Jones, J.L. Pfaltz, "Summary of the final report of the NSF workshop on scientific database management", *ACM-SIGMOD record*, 19(4) (1990) 32–40.

[37] D. Gannon, R. Bramley, T. Stuckey, J. Villacis, J. Balasubramanian, E. Akman, F. Breg, S. Diwan, M. Govindaraju, "Component architectures for distributed scientific problem solving", *IEEE Computational Science and Engineering*, 5 (1998) 50–63.

[38] G.A. Geist, J.A. Kohl, P.M. Papadopoulos, "CUMULVS: providing fault-tolerance, visualization and steering of parallel applications", *International Journal of High Performance Computing Applications*, 11 (1997) 224–236.

[39] A. Goel, C.A. Baker, C.A. Shaffer, B. Grossman, W.H. Mason, L.T. Watson, R.T. Haftka, "VizCraft: a problem solving environment for aircraft configuration design", *IEEE Computing in Science and Engineering*, 3 (2001) 56–66.

[40] E. Gallopoulos, E. Houstis E, JR. Rice, "Computer as thinker/doer: problem-solving environments for computational science", *IEEE Computing in Science and Engineering*, 1 (1994) 11-23.

[41] A. Goel, C. Phanouriou, F.A. Kamke, C.J. Ribbens, C.A. Shaffer, L.T. Watson, "WBCSim: a prototype problem solving environment for wood-based composites simulations", *Engineering with Computers*, 15 (1999) 198–210.

[42] A. Goel, C. Phanouriou, F.A. Kamke, C.J. Ribbens, C.A. Shaffer, L.T. Watson, "WBCSim: a prototype problem solving environment for wood-based composites simulations", Technical Report TR-98-25, Dept. of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1998.

[43] D.S. Goldin, S.L. Venneri, A.K. Noor, "Beyond incremental change", *IEEE Computer*, 31 (1998) 31–39.

[44] R. Haber, B. Bliss, D. Jablonowski, C. Jog, "A distributed environment for runtime visualization and application steering in computational mechanics", *Computing Systems in Engineering*, 3 (1992) 501–515.

[45] T.E.G. Harless, F.G. Wagner, P.H. Short, R.D. Seale, P.H. Mitchell, D.S. Ladd, "A model to predict the density profile of particleboard", *Wood and Fiber Science*, 19 (1987) 81–92.

[46] C.R. Haselein, "Numerical simulation of pressing wood-fiber composites", Ph.D. Dissertation, Forest Products Dept., Oregon State University, Corvallis, OR, 1998.

[47] E.N. Houstis, A.C. Catlin, J.R. Rice, V.S. Verykios, N. Ramakrishnan, C.E. Houstis, "PYTHIA-II: a knowledge/database system for managing performance data and recommending scientific software", *ACM Transactions on Mathematical Software*, 26 (2000) 227–253.

[48] E. Houstis, E. Gallopoulos, R. Bramley, J.R. Rice, "Problem-solving environments for computational science", *IEEE Computational Science and Engineering*, 4 (1997) 18–21.

[49] E.N. Houstis, J.R. Rice, S. Weerawarana, A.C. Catlin, P. Papachiou, K.Y. Wang, M. Gaitatzes, "PELLPACK: a problem solving environment for PDE-based applications on multicomputer platforms", *ACM Transactions on Mathematical Software*, 24 (1998) 30–73.

[50] P.E. Humphrey, "Physical aspects of wood particleboard manufacture", Ph.D. Dissertation, University of Wales, Bangor, UK, 1982.

[51] P.E. Humphrey, "The hot-pressing of dry-formed wood-based composites. Part II. A simulation model for heat and moisture transfer, and typical results", *Holzforschung*, 43 (1989) 199–206.

[52] Y. Ioannidis, M. Livny, "Conceptual schemas: Multi-faceted tools for desktop scientific experiment management", *Journal of Intelligent and Cooperative Information Systems*, 1(3) (1992) 451–474.

[53] Y. Ioannidis, M. Livny, S. Gupta, N. Ponnekanti, "ZOO: A desktop experiment management environment", in *Proceedings of the UK e-Science All Hands Meeting*, (Bombay, India, 1996) 274–285.

[54] P.L. Isenhour, J. Begole, W.S. Heagy, C.A. Shaffer, "Sieve: A Java-based collaborative visualization environment", in *Proc. of Late Breaking Hot Topics, IEEE Visualization '97*, (Phoenix, AZ, USA) 13–16.

[55] D.J. Jablonowski, J.D. Bruner, B. Bliss, R.B. Haber, "VASE: the visualization and application steering environment", in *Proceedings of Supercomputing 1993*, Tokyo, Japan, 1993.

[56] A. Joshi, S. Weerawarana, N. Ramakrishnan, E.N. Houstis, J.R. Rice, "Neuro-fuzzy support for problem-solving environments: a step toward automated solution of PDEs", *Special Joint Issue of IEEE Computer and IEEE Computational Science and Engineering*, 3 (1996) 44–56.

[57] S. Kumar, C. Peters-Lidard, Y. Tian, R. Reichle, J. Geiger, C. Alonge, J. Eylander, P. Houser, "An integrated hydrologic modeling and data assimilation framework", *IEEE Computer*, 41 (2008) 52–59.

[58] F.A. Kamke, J.B. Wilson, "Computer simulation of a rotary dryer: retention time", *American Institute of Chemical Engineers Journal*, 32 (1985) 263–268.

[59] F.A. Kamke, J.B. Wilson, "Computer simulation of a rotary dryer: heat and mass transfer", *American Institute of Chemical Engineers Journal*, 32 (1985) 269–275.

[60] F. Kayihan, A. Johnson, "Heat and moisture movement in wood composite materials during the pressing operation - a simplified model", in *R.W. Lewis, K. Morgan, B.A. Schrefler ed., Numerical Methods in Heat Transfer, Vol. II.*, (John Wiley & Sons Ltd., New York, 1983) 511–531.

[61] R.G. Knox, V.L. Kalb, E.R. Levine, D.J. Kendig, "A problem-solving workbench for interactive simulation of ecosystems", *IEEE Computational Science and Engineering*, 4 (1997) 52–60.

[62] M.E. Lang, M.P. Wolcott, "Modeling the consolidation of wood-strand mat", *Mechanics of Cellulosic Materials*, ASME AMD 209/MD 60 (1995) 153–176.

[63] M.E. Lang, M.P. Wolcott, "A model for viscoelastic consolidation of wood-strand mats. Part I. Structural characterization of the mat via Monte Carlo Simulation", *Wood and Fiber Science*, 28 (1996) 100–109.

[64] M.E. Lang, M.P. Wolcott, "A model for viscoelastic consolidation of wood-strand mats. Part II. Static stress-strain behavior of the mat", *Wood and Fiber Science*, 28 (1996) 369-379.

[65] R.V. Liere, J.V. Wijk, "CSE: a modular architecture for computational steering", in *Proceedings of Virtual Environments and Scientific Visualization 1996*, (Vienna, Austria, 1996) 257–266.

[66] K. Long; Van B. Straalen, "PDESolve: an object-oriented PDE analysis environment", in *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing*, (Philadelphia, PA, 1998).

[67] C. Lu, "Organization of wood elements in partially oriented flakeboard mats", Ph.D. Dissertation. Dept. of Forestry, University of British Columbia, Vancouver, BC, 1999.

[68] V. Mann, V. Matossian, R. Muralidhar, M. Parashar, "DISCOVER: an environment for Web-based interaction and steering of high-performance scientific applications", *Concurrency and Computation: Practice and Experience*, 13 (2001) 737–754.

[69] S. Markus, S. Weerawarana, E.N. Houstis, J.R. Rice, "Scientific computing via the Web: the Net Pellpack PSE server", *IEEE Computational Science and Engineering*, 4 (1997) 43–51.

[70] J. Marti, "Viewing IGES files through VRML", in *Proceedings of Visualization '97*, (IEEE Computer Society Press, Los Alamitos, CA, 1997) 471–472.

[71] BH. McCormick, F.A. DeFanti, M.D. Brown, "Visualization in scientific computing", *Computer Graphics 1987*, 21 (1987) 61–70.

[72] D. Mishra, C.A. Shaffer, N. Ramakrishnan, L.T. Watson, K.K. Bae, J. He, A. Verstak, W.H. Tranter, "$S^4W$: a problem solving environment for wireless system design", *Software: Practice and Experience*, 37 (2007) 1539–1558.

[73] S.M. Mniszewski, P.H. Beckman, P.K. Fasel, W.F. Humphrey, "Efficient coupling of parallel applications using PAWS", in *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, (Chicago, IL, 1998).

[74] J. Mulder, J.V. Wijk, R.V. Liere, "A survey of computational steering environments", *Future Generation Computer Systems*, 15 (1999) 119–129.

[75] T.H. Naylor, J.L. Balintfy, D.S. Burdick, K. Chu, *Computer Simulation Techniques*, (John Wiley and Sons, New York, 1968).

[76] S.G. Parker, M. Miller M, C.D. Hansen, C.R. Johnson, P.P. Sloan, "An integrated problem solving environment: the SCIRun computational steering system", in *Proceedings of 31st Hawaii International Conference on System Sciences*, (HICSS-31)-Vol. VII, Edited by H. El-Rewini, IEEE Computer Society, January 1998.

[77] S.G. Parker, D.M. Weinstein, C.R. Johnson, "The SCIRun computational steering software system", *Modern Software Tools in Scientific Computing, E. Arge, A.M. Bruaset, Langtangen HP (eds.), Birkhauser Press*, (1997) 1–40.

[78] C. Phanouriou, M. Abrams, "Transforming command-line driven systems to Web applications", in *Sixth International World Wide Web Conference*, (Santa Clara, CA, 1997) 1–3.

[79] N. Ramakrishnan, L.T. Watson, D.G. Kafura, C.J. Ribbens, C.A. Shaffer, "Programming environments for multidisciplinary grid communities", *Concurrency and Computation: Practice and Experience*, 14 (2002) 1241–1273.

[80] O.F. Rana, M.Z. Li, D.W. Walker, M.S. Shields, "An XML based component model for generating scientific applications and performing large scale simulations in a meta-computing environment", *GCSE*, (1999) 210–224.

[81] S. Rathmayer, M. Lenke, "A tool for online visualization and interactive steering of parallel HPC applications", in *Proceedings of the 11th International Parallel Processing Symposium*, Geneva, Switzerland, 1997.

[82] W.C. Regli, "Internet-enabled computer-aided design", *IEEE Internet Computing*, 1 (1997) 39–50.

[83] J. Resnik, F.A. Kamke, "Modeling the cure of adhesive-wood bonds using high frequency energy", in *Final Report, U.S.-Slovene Joint Board on Scientific and Technological Cooperation, Project 95-AES10*, Ljubljana, Slovenia: University of Ljubljana, 1998.

[84] J.R. Rice, R.F. Boisvert, "From scientific software libraries to problem-solving environments", *Computing in Science and Engineering*, 3 (1996) 44–53.

[85] W.E. Schiesser, *Computational Mathematics in Engineering and Applied Science: ODEs, DAEs, and PDEs*, (CRC Press, Boca Raton, 1994).

[86] C.A. Shaffer, L.T. Watson, D.G. Kafura, N. Ramakrishnan, "Features of problem solving environments for computational science", in *Proc. High Performance Computing Symp., A. Tentner (ed.), Soc. for Computer Simulation Int. 2002*, (San Diego, CA, USA) 242–247.

[87] A. Shah, "Symphony: A Java-based composition and manipulation framework for distributed legacy resources", M.S. Thesis, Dept. of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1998.

[88] J. Shu, L.T. Watson, N. Ramakrishnan, F.A. Kamke, S. Deshpande, "Computational steering in the problem solving environment WBCSim", Technical Report TR-09-08, Dept. of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, 2009.

[89] J. Shu, L.T. Watson, N. Ramakrishnan, F.A. Kamke, C. North, "Unification of problem solving environment implementation Layers with XML", *Advances in Engineering Software*, 39 (2008) 189–201.

[90] J. Shu, L.T. Watson, N. Ramakrishnan, F.A. Kamke, B.G. Zombori, "An experiment management component for the WBCSim problem solving environment", *Advances in Engineering Software*, 35 (2004) 115–123.

[91] J. Shu, L.T. Watson, B.G. Zombori, F.A. Kamke, "WBCSim: an environment for modeling wood-based composites manufacture", *Engineering with Computers*, 21 (2006) 259–271.

[92] A. Sioson, J.I. Watkinson, C. Vasquez-Robinet, M. Ellis, M. Shukla, N. Kumar Ramakrishnan, L.S. Heath, R. Grene, B.I Chevone, K. Kafadar, L.T. Watson, "Expresso and chips: creating a next generation microarray experiment management system", in *Proceedings of the Next Generation Software Workshop, 17th Internat. Parallel & Distributed Processing Symp.*, Los Alamitos, California, USA, 2003.

[93] R. Skidmore, A. Verstak, N. Ramakrishnan, T.S. Rappaport, L.T. Watson, J. He, S. Varadarajan, C.A. Shaffer, J. Chen, K.K. Bae, J. Jiang, W.H. Tranter, "Towards integrated PSEs

for wireless communications: experiences with the $S^4W$ and *SitePlanner* projects", *ACM SIGMOBILE Mobile Computing and Communications Review*, 8 (2004) 20–34.

[94] H. Thöemen, "Modeling the physical processes in natural fiber composites during batch and continuous pressing", Ph.D. dissertation. Forest Products Dept., Oregon State University, Corvallis, OR, 2000.

[95] Y. Umetani, M. Tsuji, K. Iwasawa, H. Hirayama, "DEQSOL: A numerical simulation language for vector/parallel processors", in *B. Ford, F. Chatelin ed., Problem Solving Environments for Scientific Computing*, (Elsevier Science Publishers, North-Holland, 1987) 147–162.

[96] Vanderplaats Research and Development, Inc, *DOT Users Manual, Version 4.20*, Colorado Springs: CO, 1985.

[97] M. Vass, N. Allen, C.A. Shaffer, N. Ramakrishnan, L.T. Watson, J.J. Tyson, "The JigCell model builder and run manager", *Bioinformatics*, 20 (2004) 3680–3681.

[98] M. Vass, C.A. Shaffer, N. Ramakrishnan, L.T. Watson, J.J. Tyson, "The JigCell model builder: a spreadsheet interface for creating biochemical reaction network models", *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3 (2006) 155–164.

[99] A. Verstak, "Data and computation modeling for scientific PSEs", Blacksburg, VA: M.S. Thesis, Virginia Polytechnic Institute and State University, 2002.

[100] A. Verstak, N. Ramakrishnan, L.T. Watson, J. He, C.A. Shaffer, K.K. Bae, J. Jiang, W.H. Tranter, T.S. Rappaport, "BSML: a binding schema markup language for data interchange in PSEs", *Scientific Programming*, 11 (2003) 199–224.

[101] A. Verstak, M. Vass, N. Ramakrishnan, C.A. Shaffer, L.T. Watson, K.K. Bae, J. Jiang, W.H. Tranter, T.S. Rappaport, "Lightweight data management for compositional modeling in problem solving environments", in *Proc. High Performance Computing Symp., A. Tentner (ed.), Soc. for Computer Simulation Int. 2001*, (San Diego, CA, USA) 148–153.

[102] J.S. Vetter, D.A. Reed, "Real-time performance monitoring, adaptive control, and interactive steering of computational grids", *International Journal of High Performance Computing Applications*, 14 (2000) 357–366.

[103] J. Vetter, K. Schwan, "Progress: a toolkit for interactive program steering", in *Proceedings of the 1995 International Conference on Parallel Processing*, Urbana-Champain, Illinois, USA, 1995.

[104] J. Vetter, K. Schwan, "High performance computational steering of physical simulations", in *Proceedings of the 11th International Parallel Processing Symposium*, Geneva, Switzerland, 1997.

[105] L.T. Watson, V.K. Lohani, D.F. Kibler, R.L. Dymond, N. Ramakrishnan, C.A. Shaffer, "Integrated computing environments for watershed management", *Journal of Computing in Civil Engineering*, 16 (2002) 259–268.

[106] S. Weerawarana, E.N. Houstis, J.R. Rice, A. Joshi, "PYTHIA: a knowledge based system to select scientific algorithms", *ACM Transactions on Mathematical Software*, 22 (1996) 447–468.

[107] C.F. Weggel, "Versatile 2-D analysis", *IEEE Spectrum*, 34 (1997) 92–93.

[108] S. Wolfram, *The Mathematica Book, 3rd ed.*, Wolfram Media/Cambridge University Press: Champaign, IL, 1996.

[109] J.D. Wood, K. Brodlie, J. Walton, "GViz—visualization and steering on the grid", in *Proceedings of UK e-Science All Hands Conference*, Nottingham, UK, 2003.

[110] C. Youn, M.E. Pierce, G.C. Fox, "Building problem solving environments with application Web service toolkits", in *International Conference on Computational Science (ICCS03)*, (Melbourne, Australia, 2003) 403–412.

[111] B.G. Zombori, "Modeling the transient effects during the hot-pressing of wood-based composites", Blacksburg, VA: Ph.D. Dissertation, Virginia Polytechnic Institute and State University, 2001.

[112] B.G. Zombori, F.A. Kamke, L.T. Watson, "Simulation of the mat formation process", *Wood and Fiber Science*, 33 (2001) 564–579.

[113] B.G. Zombori, F.A. Kamke, L.T. Watson, "Simulation of internal conditions during the hot-pressing process", *Wood and Fiber Science*, 35 (2003) 2–23.

[114] B.G. Zombori, F.A. Kamke, L.T. Watson, "Sensitivity analysis of internal mat environment during hot-pressing", *Wood and Fiber Science*, 36 (2004) 195–209.

## Appendix A: Perl Subroutine `gencontourplot`

The following subroutine `gencontourplot` is one of the subroutines in the Perl script `OSBsub.pl` called by the OSB Wrapper `OSB.pl`. This subroutine illustrates that the current WBCSim approach keeps the Mathematica code as a blueprint external to the wrapper. Then in every simulation run, the wrapper makes a copy of that blueprint code, uses *sed* to insert the dynamic graphics output directory, and then feeds it directly to the Mathematica kernel, which is discussed in Chapter 2.

```
# This subroutine generates contour plot GIF files.
sub gencontourplot
{
   # Output status message to stdin.
   print "STEP Use Mathematica to Generate Graphics\n";
   # Define contour plot's blueprint of Mathematica script.
   $MATH_SCRIPT = "/d/pse/WWW/bin/osb_math1_contour.script";
   # Define a working copy of the blueprint.
   $MATH_SCRIPT_COPY = "$INPUT_DIR/$$/OSB_$$.redirect";
   # Define complete working directory in a format required by Mathematica.
   $IN_DIR = "\\/d\\/pse\\/WWW\\/data\\/tmp\\/$$\\/";

   # Substitute "inputdirectory" (in the blueprint) by the working directory.
   print("PSE: Change Directory Location In Math Script.\n"), return
        if system "sed 's/inputdirectory/$IN_DIR/' $MATH_SCRIPT > $MATH_SCRIPT_COPY";

   # Output status message to stdin.
   print "STEP Generate GIF Files - Contour Plot via Mathematica\n";
   # Runs Mathematica.
   print("PSE:WARNING Could Not Execute Mathematica.\n"), return
        if system "$MATHEMATICA < $MATH_SCRIPT_COPY";

   # Embed the GIF files in a HTML file.
   $RESULT_CONTOUR_PLOT_FILE = "$OUTPUT_DIR/$$/$RESULT_ID.contourplot.html";
   open(FILE_OUT, ">$RESULT_CONTOUR_PLOT_FILE")
       or print("PSE:WARN Could Not Write Contour Plot HTML.\n"), return;
   print FILE_OUT "<html><head>\n";
   print FILE_OUT "<title>Mat Formation - Contour Plot</title></head>\n";
   print FILE_OUT "<body bgcolor=\"#ffffff\" text=\"#102030\" link=\"#B00909\"
vlink=\"#502020\">\n";
   print FILE_OUT "<center><h1>Mat Formation</h1>\n";
   print FILE_OUT "<h2>Simulation Results - Contour Plot</h2></center><hr>\n";
   print FILE_OUT "<a href=\"\#Density\">Density Distribution</a><br> ";
   print FILE_OUT "<a href=\"\#Thickness\">Thickness Distribution</a><br>";
   print FILE_OUT "<a href=\"\#Number\">Number Distribution</a><br>";
   print FILE_OUT "\n";
   print FILE_OUT "\n";
   print FILE_OUT "<hr><h3><A NAME=\"Density\"></A>Density Distribution</h3>\n";
   print FILE_OUT "<table border=0 cellpadding=0 cellspacing=5><tr><th><img
src=\"density.gif\"></th></tr></table>\n";
   print FILE_OUT "<hr><h3><A NAME=\"Thickness\"></A>Thickness Distribution</h3>\n";
   print FILE_OUT "<table border=0 cellpadding=0 cellspacing=5><tr><th><img
src=\"thickness.gif\"></th></tr></table>\n";
   print FILE_OUT "<hr><h3><A NAME=\"Number\"></A>Number Distribution</h3>\n";
```

```
   print FILE_OUT "<table border=0 cellpadding=0 cellspacing=5><tr><th><img
src=\"number.gif\"></th></tr></table>\n";
   print FILE_OUT "<hr></body></html>\n";
   close(FILE_OUT)
        or print("PSE:WARN Could Not Write Graph HTML.\n"), return;

   # Output result HTML filename and related info.
   &PrintURL("$RESULT_CONTOUR_PLOT_FILE", "text/html", "Contour Plot");
}
```

## Appendix B: Perl Code Example of WhirlGif from `gen3Dplot`

The following segment of code is from the subroutine `gen3Dplot`, which is one of the subroutines in the Perl script `HOTsub.pl` called by the HOT wrapper `HOT.pl`.

The code illustrates how WBCSim uses the the UNIX utility WhirlGif to generate a single animated GIF image file from a sequence of GIF image files, as described in Chapter 2.

```
# Because the number of frames may change from one simulation
# run to another, the code reads the number of frames from a
# data file generated by a Fortran program. Then the wrapper
# constructs a command like "WhirlGif -loop 1000 frame1.gif
# frame2.gif frame3.gif frame4.gif > animation.gif",
# where frame*.gif are the frame images generated by
# Mathematica.

# $COUNT stores the number of frames, which was readin earlier.
if ($COUNT != 1)
{
   # Define WhirlGif.
   $WHIRLGIF = "$PSE_HOME/bin/whirlgif";

   # Construct the command.
   # -loop defines the number of animation loop.
   # -time defines the animation speed.
   $MOVIE = "$WHIRLGIF -loop 1000 -time 25";

   # Continue to construct the command depending on the $COUNT
   for $k (1..$COUNT)
   {
      $MOVIE = "$MOVIE $INPUT_DIR/$$/Movie$k.gif "; $k = $k + 1;
   }

   # Continue to construct the command.
   $MOVIE = "$MOVIE > $INPUT_DIR/$$/Movie.gif";

   # Output status message to stdin.
   print "STEP Generate GIF Files - 3D Movie via WhirlGif\n";
   # Runs WhirlGif.
   print("PSE:WARN Could Not Execute WhirlGif.\n"), return
        if system "$MOVIE";

   # Embed the GIF files in a HTML file.
   $RESULT_PLOT_FILE = "$OUTPUT_DIR/$$/$RESULT_ID.plot3D.html";
   open(FILE_OUT, ">$RESULT_PLOT_FILE")
       or print("PSE:WARN Could Not Write Graph HTML.\n"), return;
   print FILE_OUT "<html><head>\n";
   print FILE_OUT "<title>Hot Compression</title></head>\n";
   print FILE_OUT "<body bgcolor=\"#ffffff\" text=\"#102030\" link=\"#B00909\"
vlink=\"#502020\">\n";
   print FILE_OUT "<center><h1>Hot Compression</h1>\n";
   print FILE_OUT "<h2>Simulation Results - 3D Profiles (Animation)</h2></center><hr>\n";
   print FILE_OUT "<h3><A NAME=\"movie\"></A></h3>\n";
   print FILE_OUT "<table border=0 cellpadding=0 cellspacing=5><tr><th><img
src=\"Movie.gif\"></th></tr></table>\n";
   print FILE_OUT "<hr></body></html>\n";
```

```
    close(FILE_OUT)
         or print("PSE:WARN Could Not Write Graph HTML.\n"), return;

    # Output result HTML filename and related info.
    &PrintURL("$RESULT_PLOT_FILE", "text/html", "3D Profiles (Animation)");
} # End of if
```

## Appendix C: Perl OSB Save Wrapper `osbsave.pl`

The following Perl script is the OSB save wrapper `osbsave.pl`, which illustrates how WBCSim interacts with the database to save inputs and outputs, as discussed in Chapter 3.

```
#!/usr/local/bin/perl -w
#
# Author: Jiang Shu
# Last Modification: 11/11/2002
#

# config.pl defines system specific variables such as
# $MATHEMATICA = "/usr/local/bin/math";
require "config.pl";

# dbutil.pl defines some common used database operations
# such as subroutine CheckDB, which takes
# 1) target value
# 2) table name
# 3) condition
# 4) database name
# and check if with this condition the target value is in
# the table of that database.
require "dbutil.pl";

# Pg(Pgsql_perl5) is an interface between Larry Wall's language
# Perl Version 5 and the database PostgreSQL (previously
# Postgres95). This has been done by using the Perl5 application
# programming interface for C extensions which calls the
# Postgres programmer's interface LIBPQ.
use Pg;

# This subroutine stores OSB inputs and outputs to database.
sub OSBStoreDB
{
   # Prepare inputs.

   # Name, ID
   # This is the case that the simulate has been run.
   # User clicks "Store Results" from output window.
   # System saves both inputs and outputs.
   # Run ID is already generated, and is passed in here.
   if ($#ARGV==1 && $ARGV[1] =  /PSE/)
   {
      # Set run ID.
      $run_id = $ARGV[1];
      # Use the run ID as input ID.
      $input_id = $run_id;
      # Output ID is the same as the input ID.
      $output_id = $input_id;

      # Output status message to stdin.
      print "PSE:ID $input_id $PROBLEM_NAME\n";

      # Get the actual inputs from the $input_id.input file.
```

```
    open(INPUT, "$OUTPUT_DIR/$input_id.input")
        or print("PSE:ERROR Cannot Open input_id.input!\n"), return;

    # Skip the first entry = Problem Description.
    # Then get the inputs.
    $_ = <INPUT>; $_ = <INPUT>;

    # Close $input_id.input file.
    close(INPUT)
         or die "PSE:ERROR Closes $input_id.input!";

    # Separate the inputs (a string) into an array of elements.
    @IN = split;

    # Call OSBConstructInputs to construct two types of inputs.
    ($inputs, $inputs2) = &OSBConstructInputs(@IN);
}
# Name, parameters
# This is the case that the simulate has NOT been run.
# User clicks the "Store Problem" from input window.
# System only saves the inputs.
# Run ID is not generated so far.
else
{
    # Output input parameters to stdin.
    print @ARGV, "\n";
    # Shift out the first entry = Problem Description.
    shift(@ARGV);

    # Generate a Run ID.
    $run_id = &GenerateFile();
    # Use the run ID as input ID.
    $input_id = $run_id;
    # Output ID is null (there is no output).
    $output_id = "null";

    # Call OSBConstructInputs to construct two types of inputs.
    ($inputs, $inputs2) = &OSBConstructInputs(@ARGV);
}

# Start actual logic/operation.

# Initialize the parameters.
# UPPER CASE VARIABLES are constants and defined in STORE.pl,
# which calls this subroutine.
$dbmain = $DATABASE;
$null = "null";
$date = &GetDate2();

# Check database for inputs.
@temp = &CheckDB("input_id", $SIM_INPUTS, "$inputs2");
$inputfound = $temp[0];

# Connect to database.
$conn = Pg::connectdb("dbname=$dbmain");
die $conn->errorMessage unless PGRES_CONNECTION_OK eq $conn->status;
```

```
    print "Connected to $dbmain for Saving.\n";

    # Inputs do not exist.
    if ($inputfound eq $null)
    {
        # Output status message to stdin.
        print "Save - Input Not Found\n";

        # Save inputs in TABLE - osb_inputs.
        # Output status message to stdin.
        print "Save - Save the Input\n";

        $inputs = "('$input_id', $inputs)";
        $result = $conn->exec("INSERT INTO $SIM_INPUTS VALUES $inputs");
        die $conn->errorMessage unless PGRES_COMMAND_OK eq $result->resultStatus;
        print "Save Inputs, status = ", $result->cmdStatus, "\n";

        # Save outputs (if there is any) in TABLE - osb_outputs.
        if ($output_id ne $null)
        {
            # Output status message to stdin.
            print "Save - Save the Output\n";

            # Construct outputs.
            $outputs = &OSBConstructOutputs();

            $outputs = "('$output_id', $outputs)";
            $result = $conn->exec("INSERT INTO $SIM_OUTPUTS VALUES $outputs");
            die $conn->errorMessage unless PGRES_COMMAND_OK eq $result->resultStatus;
            print "Save Outputs, status = ", $result->cmdStatus, "\n";

            # Some files are stored, outside of database.
            &OSBStoreFiles($input_id);
        }
        else
        {
            # Output status message to stdin.
            print "Save - Output is NOT found, so NO outputs saved\n";
        }

        # Store the run.
        # Output status message to stdin.
        print "Save - Save Run\n";
        $result = $conn->exec("INSERT INTO $SIM_RUNS VALUES ('$run_id', '$PROBLEM_NAME',
'$USERNAME', '$date', '$input_id', '$output_id')");
        die $conn->errorMessage unless PGRES_COMMAND_OK eq $result->resultStatus;
        print "Save Runs - run_id = $run_id, input_id = $input_id, output_id = $output_id, status
= ", $result->cmdStatus, "\n";
    }
    # Inputs do exist.
    else
    {
        # Output status message to stdin.
        print "Input Found - $inputfound\n";

        # NOT save inputs.
```

```
       # Output status message to stdin.
       print "Save - Not Save the Inputs\n";

       # Check if the outputs are in the database.
       @temp = &CheckDB("output_id", $SIM_OUTPUTS, "output_id=\'$inputfound\'");
       $outputfound = $temp[0];

       # No output is found in the database.
       if ($outputfound eq $null)
       {
          # Output was generated.
          if ($output_id ne $null)
          {
             # Output status message to stdin.
             print "Save - Output Not Found, But Output is Available -> Save the Outputs\n";

             # Construct outputs.
             $outputs = &OSBConstructOutputs();
             $outputs = "('$inputfound', $outputs)";

             # Save the outputs in the database.
             $result = $conn->exec("INSERT INTO $SIM_OUTPUTS VALUES $outputs");
             die $conn->errorMessage unless PGRES_COMMAND_OK eq $result->resultStatus;
             print "Save Outputs, status = ", $result->cmdStatus, "\n";
             $outputfound = $inputfound;

             # Some files are stored, outside of database.
             &OSBStoreFiles($inputfound);

             # Smart Update.
             # Update the old runs to point to the newly saved outputs.
             $result = $conn->exec("UPDATE $SIM_RUNS SET output_id=\'$outputfound\' WHERE
input_id=\'$inputfound\'");
             die $conn->errorMessage unless PGRES_COMMAND_OK eq $result->resultStatus;
             print "Smart Update, status = ", $result->cmdStatus, "\n";
          }
          # Output was NOT generated.
          else
          {
             # Output status message to stdin.
             print "Save - Output is NOT found, NOT Save the Outputs\n";
          }
       }
       # Outputs are found in the database.
       else
       {
          # Output status message to stdin.
          print "Save - Output Found - $outputfound, NOT Save the Outputs\n";
       }

       # Store the run.
       print "Save - Save Run\n";
       $result = $conn->exec("INSERT INTO $SIM_RUNS VALUES ('$run_id', '$PROBLEM_NAME',
'$USERNAME', '$date', '$inputfound', '$outputfound')");
       die $conn->errorMessage unless PGRES_COMMAND_OK eq $result->resultStatus;
```

```
       print "Save Runs - run_id = $run_id, input_id = $inputfound, output_id = $outputfound,
status = ", $result->cmdStatus, "\n";
   }
} # End of OSBStoreDBsub

# This subroutine constructs two types of inputs.
# 1) $inputs = just the parameters
# 2) $inputs2 = inputs with        column names for the query
sub OSBConstructInputs
{
   # Initialize the parameters.
   $i = 0;
   $inputs = "";
   $inputs2 = "";

   # First few column names in the table.
   @p = ("dimension", "layers", "seed", "mat_d1", "mat_d2", "nmp1", "nmp2", "cut1", "cut2",
"noi", "pn");

   # Get @p data.
   for $i (0..10)
   {
      $a = shift(@_);
      $inputs = "$inputs$a, ";
      $inputs2 = "$inputs2$p[$i]=$a and ";

      # Get dimension (2D or 3D).
      if ($i == 0)
      {
         $dimension = $a;
      }

      # Get number of layers.
      if ($i == 1)
      {
         $number_of_layers = $a;
      }

      $i++;
   }

   # Get number of press.
   $pn = 2*$a;
   $pndata = "{";
   $i = 1;
   for $i (1..$pn)
   {
      $a = shift(@_);
      $pndata = "$pndata\"$a\", ";
      $i++;
   }
   chop($pndata); chop($pndata);
   $pndata = "$pndata}";
   $inputs = "$inputs\'$pndata\', ";
   $x = "pressdata";
   $inputs2 = "$inputs2$x=\'$pndata\' and ";
```

```
    # Get layers data.
    $layers = "{";
    $i = 1;
    for $i (1..$number_of_layers)
    {
        $one_layer = "\"";
        $j = 0;
        for $j (1..7)
        {
            $a = shift(@_);
            $one_layer = "$one_layer$a ";
            $j++;
        }

        $j = 0;
        for $j (1..4)
        {
            $a = shift(@_);
            $one_layer = "$one_layer$a ";
            $type = $a;
            $a = shift(@_);
            $one_layer = "$one_layer$a ";
            $number_of_parameters = $a;
            if ($type == 3)
            {
                $number_of_parameters = $number_of_parameters * 2 + 1;
            }

            $k = 1;
            for $k (1..$number_of_parameters)
            {
                 $a = shift(@_);
                 $one_layer = "$one_layer$a ";
                 $k++;
            }

            $j++;
        }
        chop($one_layer);
        $one_layer = "$one_layer\"";
        $layers = "$layers$one_layer, ";
        $i++;
    }
    chop($layers); chop($layers);
    $layers= "$layers}";

    $inputs = "$inputs\'$layers\'";
    $x = "layerdata";
    $inputs2 = "$inputs2$x=\'$layers\'";

    return ($inputs, $inputs2);
} # End of ConstructInput

# This subroutine constructs outputs.
sub OSBConstructOutputs
```

```
{
    # Initialize the parameters.
    $all_outputs = "";

    # The data files will be saved.
    @datafiles = ("osb_Azoned.dat", "osb_Azoned.dat", "osb_Azoned.dat", "osb_Azoned.dat",
"osb_voidcont.dat", "osb_stat.dat");

    foreach (@datafiles)
    {
        # Process one data file at a time.
        $one_data_file = &OSBConstructOutputsSub1($_);
        $all_outputs = "$all_outputs\'$one_data_file\', ";
    }

    # One additional data file for Hot Compression, which has
    # different format from the above ones.
    $one_data_file = &OSBConstructOutputsSub2("osb_matform.dat");
    $all_outputs = "$all_outputs\'$one_data_file\'";

    return $all_outputs;
} # End of OSBConstructOutputs

# This subroutine constructs outputs for @datafiles defined in
# OSBConstructOutputs.
sub OSBConstructOutputsSub1
{
    # Receive the data file name as argument.
    $datafile = $_[0];

    # Parse the folder name.
    ($_, $_, $_, $_, $_, $folder) = split /-/, $run_id;

    open(OUTPUT, "$OUTPUT_DIR/$folder/$datafile")
        or print("PSE:ERROR Cannot Open $datafile!\n"), return;

    # Construct outputs.
    $matrix = "";

    foreach (<OUTPUT>)
    {
       @one_row = split;
       $a_row = "{";
       foreach (@one_row)
       {
          $a_row = "$a_row$_, ";
       }
       chop($a_row); chop($a_row);
       $a_row = "$a_row}, ";
       $matrix = "$matrix$a_row";
    }
    chop($matrix); chop($matrix);

    close(OUTPUT)
         or die "PSE:ERROR Cannot Close $datafile!";
```

```
    return "{$matrix}";
} # End of OSBConstructOutputsSub1

# This subroutine constructs outputs for osb_matform.dat,
# which has different format from the other files.
sub OSBConstructOutputsSub2
{
    # Receive the data file name as argument.
    $datafile = $_[0];

    # Parse the folder name.
    ($_, $_, $_, $_, $_, $folder) = split /-/, $run_id;

    open(OUTPUT, "$OUTPUT_DIR/$folder/$datafile")
        or print("PSE:ERROR Cannot Open $datafile!\n"), return;

    # Construct outputs.
    $one_line = "";

    <OUTPUT>;

    foreach (<OUTPUT>)
    {
       @one_row = split;
       if ($one_row[0] ne "/")
       {
          $one_line = "$one_line$one_row[1], ";
       }
    }
    chop($one_line); chop($one_line);

    close(OUTPUT)
         or die "PSE:ERROR Cannot Close $datafile!";

    return "{$one_line}";
} # End of OSBConstructOutputsSub2

# Some files are stored outside of the database such as:
# 1) The descriptive input file, which only helps a user to
#    understand the input file.
# 2) 2D and 3D layer images, which are only for visual effect.
#    datafiles (containing coordinates of flakes)
#    are not saved in the database.
sub OSBStoreFiles
{
    # Receive ID as argument.
    $id = $_[0];

    # Parse the folder name.
    ($_, $_, $_, $_, $_, $folder) = split /-/, $output_id;

    # Save the descriptive input file.
    print("PSE:ERROR Copying 2D pictures.\n"), exit 2
         if system "$CP $OUTPUT_DIR/$folder/osb_input.dat $SIMDIR/$id.osb_input.dat";

    # Save 2D Gif Files.
```

```
    for $i (1..$number_of_layers)
    {
           print("PSE:ERROR Copying 2D pictures.\n"), exit 2
                if system "$CP $OUTPUT_DIR/$folder/layer$i.gif $SIMDIR/$id.layer$i.gif";
    }
    print("PSE:ERROR Copying 2D pictures.\n"), exit 2
         if system "$CP $OUTPUT_DIR/$folder/layer_all.gif $SIMDIR/$id.layer_all.gif";

    # Save 3D Gif File.
    if ($dimension == 2)
    {
       print("PSE:ERROR Copying 3D pictures.\n"), exit 2
            if system "$CP $OUTPUT_DIR/$folder/3D.gif $SIMDIR/$id.3D.gif";
    }
} # End of OSBStoreFiles

1; # It will fail without this line
# End of file
```

## Appendix D: Perl OSB Load Wrapper `osbload.pl`

The following Perl script is the OSB load wrapper `osbload.pl`, which illustrates how WBCSim interacts with the database to load saved simulation runs, as discussed in Chapter 3.

```
#!/usr/local/bin/perl -w
#
# Author: Jiang Shu
# Last Modification: 11/11/2002
#

# config.pl defines system specific variables such as
# $MATHEMATICA = "/usr/local/bin/math";
require "config.pl";

# dbutil.pl defines some common used database operations
# such as subroutine CheckDB, which takes
# 1) target value
# 2) table name
# 3) condition
# 4) database name
# and check if with this condition the target value is in
# the table of that database.
require "dbutil.pl";

# Pg(Pgsql_perl5) is an interface between Larry Wall's language
# Perl Version 5 and the database PostgreSQL (previously
# Postgres95). This has been done by using the Perl5 application
# programming interface for C extensions which calls the
# Postgres programmer's interface LIBPQ.
use Pg;

sub OSBLoadDB
{
    # UPPER CASE VARIABLES are constants and defined in LOAD.pl,
    # which calls this subroutine.
    $dbmain = $DATABASE;

    # Connect to database.
    $conn = Pg::connectdb("dbname=$dbmain");
    die $conn->errorMessage unless PGRES_CONNECTION_OK eq $conn->status;
    print "Connected to $dbmain for Loading\n";

    # Retrieve all the runs from OSB Runs table.
    $result2 = $conn->exec("SELECT * FROM $SIM_RUNS");
    die $conn->errorMessage unless PGRES_TUPLES_OK eq $result2->resultStatus;
    print "Retrieve all the runs from - $SIM_RUNS, status = ", $result2->cmdStatus, "\n";

    while (@row = $result2->fetchrow)
    {
          $ID = $row[0];

          # Output status message to stdin.
          print "PSE:ID $ID***$row[1]***$row[3]\n";
```

```
         # Print archive URL.
         print "PSE:URL http://wbc.forprod.vt.edu/ pse/data/tmp/$ID.input x-wbc/x-archive
Archive File\n";

         # Print parameters, have to split long lines.
         # Prepare target string.
         $target = "";
         @p = ("dimension", "layers", "seed", "mat_d1", "mat_d2", "nmp1", "nmp2", "cut1",
"cut2", "noi", "pn", "pressdata", "layerdata");
         foreach (@p)
         {
            $target = "$target$_, ";
         }
         chop($target); chop($target);

         # Get the inputs.
         $para = $conn->exec("SELECT $target FROM $SIM_INPUTS WHERE input_id=\'$row[4]\'");
         @temp = $para->fetchrow;

         print "PSE:PARAM ";

         $param = "";

         # Print out the single values.
         $i = 1;
         for $i (1..11)
         {
            $_ = shift(@temp);
            $param = "$param$_ ";
            if ($i == 1)
            {
               $dimension = $_;
            }
            $i++;
         }

         # Print out press data.
         $_ = shift(@temp);
         @x = split /{|}|,|"/, $_;
         $space = $x[0];
         foreach (@x)
         {
            if ($_ ne $space)
            {
               $param = "$param$_ ";
            }
         }

         # Print out layer data.
         $_ = shift(@temp);
         @x = split /{|}|,|"/, $_;
         $space = $x[0];
         foreach (@x)
         {
            if ($_ ne $space)
            {
```

```
            $param = "$param$_ ";
         }
      }
      chop($param);

      while (length($param) > 66)
      {
            print substr($param, 0, 66), "\\\n";
            $param = substr($param, 66);
      }
      print $param, "\n";

      # Print output URLs.
      $null = "null";
      @temp = &CheckDB("output_id", $SIM_OUTPUTS, "output_id=\'$row[4]\'");
      $outputfound = $temp[0];
      if ($outputfound ne $null)
      {
          ($_, $_, $_, $_, $_, $folder) = split /-/, $outputfound;
          print "PSE:URL http://wbc.forprod.vt.edu/ pse/data/tmp/$folder/$ID.input.txt
text/plain Namelist Input Data\n";
        print "PSE:URL http://wbc.forprod.vt.edu/ pse/data/tmp/$folder/$ID.contourplot.html
text/html Contour Plot\n";
       print "PSE:URL http://wbc.forprod.vt.edu/ pse/data/tmp/$folder/$ID.surfaceplot.htmll
text/html Surface Plot\n";
          print "PSE:URL
http://wbc.forprod.vt.edu/ pse/data/tmp/$folder/$ID.compressionplot.html text/html
Compression Plot\n";
          print "PSE:URL http://wbc.forprod.vt.edu/ pse/data/tmp/$folder/$ID.layerplot.html
text/html Layer Plot\n";
          if ($dimension == 2)
          {
             print "PSE:URL http://wbc.forprod.vt.edu/ pse/data/tmp/$folder/$ID.3D.html
text/html 3D Plot\n";
          }
        print "PSE:URL http://wbc.forprod.vt.edu/ pse/data/tmp/$folder/$ID.outputfiles.html
text/html Text Output Files\n";
      }
   }
} # End of OSBLoadDB

1; # It will fail without this line
# End of file
```

## Appendix E: XML Datasheet for Hot Compression Model `hot.xml`

The following is the complete XML datasheet for the hot compression model, as discussed in Chapters 4 and 5.

```
<?xml version="1.0"?>
<simulation>
  <simulation_name>Hot Pressing</simulation_name>

  <simulation_name_abbreviation>HOT</simulation_name_abbreviation>

  <simulation_image>../img/hot_compression_front.jpg</simulation_image>

  <short_description>
    This model simulates hot pressing process of flake mat created by mat
    formation model in a batch press using two dimenional heat and mass transfer
    theory. It calculates the internal environment conditions such as the
    temperature, moisture, and pressure changes and adhesive cure during the
    consolidation process.
  </short_description>

  <long_description>
    This model simulates hot pressing process of flake mat created by mat
    formation model in a batch press using two dimenional heat and mass transfer
    theory. It calculates the internal environment conditions such as the
    temperature, moisture, and pressure changes and adhesive cure during the
    consolidation process.
    For more information, please see the following dissertation on the HOT
    model:

http://scholar.lib.vt.edu/theses/available/etd-04262001-172039/unrestricted/5Compression.pdf
  </long_description>

  <announcement>
  </announcement>

  <parameter_section>
    <title>Menu Inputs</title>

    <parameter>
      <name>Steering Setup</name>
      <short_name>inputs_steering_setup</short_name>
      <value>Steering Setup</value>
    </parameter>

    <parameter>
      <name>Mat Formation</name>
      <short_name>inputs_mat</short_name>
      <value>Mat Formation (One Layer Mat)</value>
    </parameter>

    <parameter>
      <name> Initial Conditions</name>
      <short_name>inputs_inits</short_name>
      <value>Initial Conditions</value>
```

```
  </parameter>

  <parameter>
    <name> Boundary Conditions</name>
    <short_name>inputs_boundary</short_name>
    <value>Boundary Conditions</value>
  </parameter>

  <parameter>
    <name> Adhesive Cure</name>
    <short_name>inputs_adhesive</short_name>
    <value>Adhesive Cure</value>
  </parameter>

  <parameter>
    <name> Press Schedule</name>
    <short_name>inputs_press</short_name>
    <value>Press Schedule</value>
  </parameter>

  <help_text>None</help_text>
</parameter_section>

<parameter_section>
  <title>Menu Model Parameters</title>

  <parameter>
    <name>MAT Transport Prop.</name>
    <short_name>model_mat</short_name>
    <value>Mat Transport Properties</value>
  </parameter>

  <parameter>
    <name> Boundary Transport Prop.</name>
    <short_name>model_boundary</short_name>
    <value>Boundary Transport Properties</value>
  </parameter>

  <parameter>
    <name>Compression Prop.</name>
    <short_name>model_compression</short_name>
    <value>Compression Properties</value>
  </parameter>

  <help_text>None</help_text>
</parameter_section>

<parameter_section>
  <title>Menu Steering</title>

  <parameter>
    <name>Steering Parameters</name>
    <short_name>steering_parameters</short_name>
    <value>Steering Parameters</value>
  </parameter>
```

```
  <help_text>None</help_text>
</parameter_section>

<parameter_section>
  <title>Menu Outputs</title>

  <parameter>
    <name>Parameters</name>
    <short_name>output_parameters</short_name>
    <value>Output Style</value>
  </parameter>

  <help_text>None</help_text>
</parameter_section>

<parameter_section>
  <title>Steering Setup</title>

  <parameter>
    <name>Iteration</name>
    <short_name>iteration</short_name>
    <field_type>textbox</field_type>
    <default>1</default>
  </parameter>

  <parameter>
    <name>Max Wait Time (secs)</name>
    <short_name>maxwaittime</short_name>
    <field_type>textbox</field_type>
    <default>300</default>
  </parameter>

  <parameter>
    <name>Wait Time (secs)</name>
    <short_name>waittime</short_name>
    <field_type>textbox</field_type>
    <default>30</default>
  </parameter>

  <help_text>None</help_text>
</parameter_section>

<parameter_section>
  <title>Mat Formation (One Layer Mat)</title>

  <parameter>
    <name>Mat Formation Data (can only use one layer data):</name>
    <short_name>matformation</short_name>
    <field_type>textbox</field_type>
    <default>PSE2004-06-15-15-06-28011</default>
  </parameter>

  <help_text>None</help_text>
</parameter_section>

<parameter_section>
```

```
<title>Initial Conditions</title>

<parameter>
  <name>Mat Initial Temperature (To)</name>
  <short_name>To</short_name>
  <unit>C</unit>
  <field_type>textbox</field_type>
  <min>10</min>
  <max>50</max>
  <tooltip_text>Mat Initial Temperature (To)</tooltip_tex>
 <default>
    25</default>
</parameter>

<parameter>
  <name>Mat Moisture Content (M.C.o)</name>
  <short_name>MCo</short_name>
  <unit>%</unit>
  <field_type>textbox</field_type>
  <min>0</min>
  <max>20</max>
  <tooltip_text>Mat Moisture Content (M.C.o)</tooltip_tex>
  <default>10</default>
</parameter>

<parameter>
  <name>Difference in MC at Face from core (M.C.f)</name>
  <short_name>MCf</short_name>
  <unit>%</unit>
  <field_type>textbox</field_type>
  <min>0</min>
  <max>20</max>
  <tooltip_text>Difference in MC at Face from core (M.C.f)</tooltip_tex>
  <default>0</default>
</parameter>

<parameter>
  <name>Ambient Air Relative Humidity (R.H.o)</name>
  <short_name>RHo</short_name>
  <unit>%</unit>
  <field_type>textbox</field_type>
  <min>0</min>
  <max>100</max>
  <tooltip_text>Ambient Air Relative Humidity (R.H.o)</tooltip_tex>
  <default>35</default>
</parameter>

<parameter>
  <name>Ambient Air Total Pressure (Patm)</name>
  <short_name>Patm</short_name>
  <unit>(Pa=10&lt;sup&gt;-6&lt;/sup&gt; bar)</unit>
  <field_type>textbox</field_type>
  <min>101325</min>
  <max>101325</max>
  <tooltip_text>Ambient Air Total Pressure (Patm)</tooltip_tex>
  <default>101325</default>
```

```
  </parameter>

  <help_text>Initial Conditions -&lt;br/&gt;&lt;br/&gt;Mat MC has initial gradient [MCf
    (Diff. between face and core): 4 when you want the MC at face 12%, and MCo
    (core): low (8 %)].&lt;br/&gt;&lt;br/&gt;Theses MCs are adjustable, so you can
    set MCo(core MC), and the difference from core to face layer, MCf (4 % as
    default).  Smooth curve of MC in the thickness direction is given by built-in
    function (MMF) with these two values.&lt;br/&gt;&lt;br/&gt;&lt;center&gt;&lt;img
    src=../img/hot_ic_help.jpg alt="Initial Conditions Help"
    border=0&gt;&lt;/center&gt;</help_text>
</parameter_section>

<parameter_section>
  <title>Boundary Conditions</title>

  <parameter>
    <name>Top Platen</name>
    <short_name>Tt</short_name>
    <unit>C</unit>
    <field_type>textbox</field_type>
    <min>10</min>
    <max>200</max>
    <tooltip_text>Temperature at Top Platen</tooltip_tex>
    <default>100</default>
  </parameter>

  <parameter>
    <name>Left Edge</name>
    <short_name>Tl</short_name>
    <unit>C</unit>
    <field_type>textbox</field_type>
    <min>10</min>
    <max>200</max>
    <tooltip_text>Temperature at Left Edge</tooltip_tex>
    <default>100</default>
  </parameter>

  <parameter>
    <name>Right Edge</name>
    <short_name>Tr</short_name>
    <unit>C</unit>
    <field_type>textbox</field_type>
    <min>10</min>
    <max>200</max>
    <tooltip_text>Temperature at Right Edge</tooltip_tex>
    <default>100</default>
  </parameter>

  <parameter>
    <name>Bottom Platen</name>
    <short_name>Tb</short_name>
    <unit>C</unit>
    <field_type>textbox</field_type>
    <min>10</min>
    <max>200</max>
    <tooltip_text>Temperature at Bottom Platen</tooltip_tex>
```

```
    <default>200</default>
  </parameter>

  <help_text>None</help_text>
</parameter_section>

<parameter_section>
  <title>Adhesive Cure</title>

  <parameter>
    <name>Adhesive Type</name>
    <short_name>adhesivetype</short_name>
    <field_type>dropdown</field_type>
    <dropdown_item>Phenol-formaldehyde</dropdown_item>
    <dropdown_item>Urea-formaldehyde</dropdown_item>
    <default>Phenol-formaldehyde</default>
  </parameter>

  <help_text>None</help_text>
</parameter_section>

<parameter_section>
  <title>Phenol-formaldehyde</title>

  <parameter>
    <name>Reaction Constant (A)</name>
    <short_name>A</short_name>
    <unit>1/s</unit>
    <field_type>textbox</field_type>
    <min>0.1</min>
    <max>10</max>
    <tooltip_text>Reaction Constant (A)</tooltip_tex>
    <default>0.253</default>
  </parameter>

  <parameter>
    <name>Activation Energy (E)</name>
    <short_name>E</short_name>
    <unit>J/mol</unit>
    <field_type>textbox</field_type>
    <min>1*10^3</min>
    <max>1*10^5</max>
    <tooltip_text>Activation Energy (E)</tooltip_tex>
    <default>12423</default>
  </parameter>

  <parameter>
    <name>Order of Reaction (n)</name>
    <short_name>n</short_name>
    <unit>unit>
    <field_type>textbox</field_type>
    <min>0.1</min>
    <max>2</max>
    <tooltip_text>Order of Reaction (n)</tooltip_tex>
    <default>0.587</default>
  </parameter>
```

```
  <parameter>
    <name>Resin Content (rc)</name>
    <short_name>rc</short_name>
    <unit>%</unit>
    <field_type>textbox</field_type>
    <min>0</min>
    <max>20</max>
    <tooltip_text>Resin Content (rc)</tooltip_tex>
    <default>6.4</default>
  </parameter>

  <parameter>
    <name>Resin Enthalph (hg)</name>
    <short_name>hg</short_name>
    <unit>J/kg</unit>
    <field_type>textbox</field_type>
    <min>1*10^5</min>
    <max>5*10^5</max>
    <tooltip_text>Resin Enthalph (hg)</tooltip_tex>
    <default>3*10^5</default>
  </parameter>

  <help_text>None</help_text>
</parameter_section>

<parameter_section>
  <title>Urea-formaldehyde</title>

  <parameter>
    <name>Reaction Constant (A)</name>
    <short_name>A</short_name>
    <unit>1/s</unit>
    <field_type>textbox</field_type>
    <min>0.1</min>
    <max>10</max>
    <tooltip_text>Reaction Constant (A)</tooltip_tex>
    <default>0.253</default>
  </parameter>

  <parameter>
    <name>Activation Energy (E)</name>
    <short_name>E</short_name>
    <unit>J/mol</unit>
    <field_type>textbox</field_type>
    <min>1*10^3</min>
    <max>1*10^5</max>
    <tooltip_text>Activation Energy (E)</tooltip_tex>
    <default>1563</default>
  </parameter>

  <parameter>
    <name>Order of Reaction (n)</name>
    <short_name>n</short_name>
    <unit><unit>
    <field_type>textbox</field_type>
```

```
    <min>0.1</min>
    <max>2</max>
    <tooltip_text>Order of Reaction (n)</tooltip_tex>
    <default>0.838</default>
  </parameter>

  <parameter>
    <name>Resin Content (rc)</name>
    <short_name>rc</short_name>
    <unit>%</unit>
    <field_type>textbox</field_type>
    <min>0</min>
    <max>20</max>
    <tooltip_text>Resin Content (rc)</tooltip_tex>
    <default>6.4</default>
  </parameter>

  <parameter>
    <name>Resin Enthalph (hg)</name>
    <short_name>hg</short_name>
    <unit>J/kg</unit>
    <field_type>textbox</field_type>
    <min>1*10^5</min>
    <max>5*10^5</max>
    <tooltip_text>Resin Enthalph (hg)</tooltip_tex>
    <default>3*10^5</default>
  </parameter>

  <help_text>None</help_text>
</parameter_section>

<parameter_section>
  <title>Press Schedule</title>

  <parameter>
    <name>Image</name>
    <default>"img/pressnew.jpg" height=313 width=370</default>
  </parameter>

  <parameter>
    <name>Position (mm)</name>
  </parameter>

  <parameter>
    <name>Number of Variables</name>
    <default>3</default>
  </parameter>

  <parameter>
    <name>Press Initial Position (PIP)</name>
    <short_name>PIP</short_name>
    <unit>mm</unit>
    <field_type>textbox</field_type>
    <min>50</min>
    <max>200</max>
    <tooltip_text>Press Initial Position (PIP)</tooltip_tex>
```

```
  <default>153.5</default>
</parameter>

<parameter>
  <name>Press Closed Position (PCP)</name>
  <short_name>PCP</short_name>
  <unit>mm</unit>
  <field_type>textbox</field_type>
  <min>0.375</min>
  <max>50.8</max>
  <tooltip_text>Press Closed Position (PCP)</tooltip_tex>
  <default>19.0</default>
</parameter>

<parameter>
  <name>Press Open Position (POP)</name>
  <short_name>POP</short_name>
  <unit>mm</unit>
  <field_type>textbox</field_type>
  <min>1.375</min>
  <max>52.8</max>
  <tooltip_text>Press Open Position (POP)</tooltip_tex>
  <default>19.95</default>
</parameter>

<parameter>
  <name>Time (s)</name>
</parameter>

<parameter>
  <name>Number of Variables</name>
  <default>3</default>
</parameter>

<parameter>
  <name>Press Closing Time (PCT)</name>
  <short_name>PCT</short_name>
  <unit>s</unit>
  <field_type>textbox</field_type>
  <min>10</min>
  <max>80</max>
  <tooltip_text>Press Closing Time (PCT)</tooltip_tex>
  <default>50</default>
</parameter>

<parameter>
  <name>Press Opening Time (POT)</name>
  <short_name>POT</short_name>
  <unit>s</unit>
  <field_type>textbox</field_type>
  <min>230</min>
  <max>520</max>
  <tooltip_text>Press Opening Time (POT)</tooltip_tex>
  <default>480</default>
</parameter>
```

```
  <parameter>
    <name>Press Total Time (PTT)</name>
    <short_name>PTT</short_name>
    <unit>s</unit>
    <field_type>textbox</field_type>
    <min>250</min>
    <max>600</max>
    <tooltip_text>Press Total Time (PTT)</tooltip_tex>
    <default>520</default>
  </parameter>

  <help_text>None</help_text>
</parameter_section>

<parameter_section>
  <title>Mat Transport Properties</title>

  <parameter>
    <name>Heat Transfer Properties</name>
  </parameter>

  <parameter>
    <name>Number of Variables</name>
    <default>3</default>
  </parameter>

  <parameter>
    <name>Cell Wall Thermal Conductivity (kcw)</name>
    <short_name>kcw</short_name>
    <unit>J/m/s/k</unit>
    <field_type>textbox</field_type>
    <min>0.217</min>
    <max>0.217</max>
    <tooltip_text>Cell Wall Thermal Conductivity (kcw)</tooltip_tex>
    <default>0.217</default>
  </parameter>

  <parameter>
    <name>Air Thermal Conductivity (ka)</name>
    <short_name>ka</short_name>
    <unit>J/m/s/k</unit>
    <field_type>textbox</field_type>
    <min>0.024</min>
    <max>0.024</max>
    <tooltip_text>Air Thermal Conductivity (ka)</tooltip_tex>
    <default>0.024</default>
  </parameter>

  <parameter>
    <name>Moisture Thermal (kw)</name>
    <short_name>kw</short_name>
    <unit>J/m/s/k</unit>
    <field_type>textbox</field_type>
    <min>0.4</min>
    <max>0.4</max>
    <tooltip_text>Moisture Thermal (kw)</tooltip_tex>
```

```
  <default>0.4</default>
</parameter>

<parameter>
  <name>Mass Transfer Properties</name>
</parameter>

<parameter>
  <name>Number of Variables</name>
  <default>4</default>
</parameter>

<parameter>
  <name>Mat Permeability (Kg)</name>
  <short_name>Kg</short_name>
  <unit>m&lt;sup&gt;3&lt;/sup&gt;/m</unit>
  <field_type>textbox</field_type>
  <min>9*10^-13</min>
  <max>9*10^-13</max>
  <tooltip_text>Mat Permeability (Kg)</tooltip_tex>
  <default>9.*10^-13</default>
</parameter>

<parameter>
  <name>Correction Factor (beta)</name>
  <short_name>beta</short_name>
  <unit><unit>
  <field_type>textbox</field_type>
  <min>0.07</min>
  <max>0.07</max>
  <tooltip_text>Correction Factor (beta)</tooltip_tex>
  <default>0.07</default>
</parameter>

<parameter>
  <name>Diffusion Attenuation Factor (alpha)</name>
  <short_name>alpha</short_name>
  <unit><unit>
  <field_type>textbox</field_type>
  <min>0.5</min>
  <max>0.5</max>
  <tooltip_text>Diffusion Attenuation Factor (alpha)</tooltip_tex>
  <default>0.5</default>
</parameter>

<parameter>
  <name>Bound Water Diffusivity (Db)</name>
  <short_name>Db</short_name>
  <unit>kg/m&lt;sup&gt;3&lt;/sup&gt;</unit>
  <field_type>textbox</field_type>
  <min>3*10^-13</min>
  <max>3*10^-13</max>
  <tooltip_text>Bound Water Diffusivity (Db)</tooltip_tex>
  <default>3*10^-13</default>
</parameter>
```

```
  <help_text>Mass Transfer Properties -&lt;br/&gt;&lt;br/&gt;Km = ko * beta, here Km is
    modified permeability, ko is original permeability, and beta is correction
    factor.</help_text>
</parameter_section>

<parameter_section>
  <title>Boundary Transport Properties</title>

  <parameter>
    <name>Top Boundary</name>
  </parameter>

  <parameter>
    <name>Number of Variables</name>
    <default>3</default>
  </parameter>

  <parameter>
    <name>Heat Transfer Coefficient (Ht)</name>
    <short_name>Ht</short_name>
    <unit>J/m&lt;sup&gt;2&lt;/sup&gt;/s/k</unit>
    <field_type>textbox</field_type>
    <min>0</min>
    <max>1*10^3</max>
    <tooltip_text>Heat Transfer Coefficient (Ht) at Top Boundary</tooltip_tex>
    <default>0.25</default>
  </parameter>

  <parameter>
    <name>Bulk Flow Coefficient (Kt)</name>
    <short_name>Kt</short_name>
    <unit>m</unit>
    <field_type>textbox</field_type>
    <min>1*10^-6</min>
    <max>1*10^3</max>
    <tooltip_text>Bulk Flow Coefficient (Kt) at Top Boundary</tooltip_tex>
    <default>1*10^-6</default>
  </parameter>

  <parameter>
    <name>Diffusion Coefficient (Dt)</name>
    <short_name>Dt</short_name>
    <unit>m/s</unit>
    <field_type>textbox</field_type>
    <min>0</min>
    <max>1*10^3</max>
    <tooltip_text>Diffusion Coefficient (Dt) at Top Boundary</tooltip_tex>
    <default>1.5</default>
  </parameter>

  <parameter>
    <name>Left Boundary</name>
  </parameter>

  <parameter>
    <name>Number of Variables</name>
```

```
  <default>3</default>
</parameter>

<parameter>
  <name>Heat Transfer Coefficient (Hl)</name>
  <short_name>Hl</short_name>
  <unit>J/m&lt;sup&gt;2&lt;/sup&gt;/s/k</unit>
  <field_type>textbox</field_type>
  <min>0</min>
  <max>1*10^3</max>
  <tooltip_text>Heat Transfer Coefficient (Hl) at Left Boundary</tooltip_tex>
  <default>0.25</default>
</parameter>

<parameter>
  <name>Bulk Flow Coefficient (Kl)</name>
  <short_name>Kl</short_name>
  <unit>m</unit>
  <field_type>textbox</field_type>
  <min>1*10^-6</min>
  <max>1*10^3</max>
  <tooltip_text>Bulk Flow Coefficient (Kl) at Left Boundary</tooltip_tex>
  <default>1*10^-6</default>
</parameter>

<parameter>
  <name>Diffusion Coefficient (Dl)</name>
  <short_name>Dl</short_name>
  <unit>m/s</unit>
  <field_type>textbox</field_type>
  <min>0</min>
  <max>1*10^3</max>
  <tooltip_text>Diffusion Coefficient (Dl) at Left Boundary</tooltip_tex>
  <default>1.5</default>
</parameter>

<parameter>
  <name>Right Boundary</name>
</parameter>

<parameter>
  <name>Number of Variables</name>
  <default>3</default>
</parameter>

<parameter>
  <name>Heat Transfer Coefficient (Hr)</name>
  <short_name>Hr</short_name>
  <unit>J/m&lt;sup&gt;2&lt;/sup&gt;/s/k</unit>
  <field_type>textbox</field_type>
  <min>0</min>
  <max>1*10^3</max>
  <tooltip_text>Heat Transfer Coefficient (Hr) at Right Boundary</tooltip_tex>
  <default>0.25</default>
</parameter>
```

```
<parameter>
  <name>Bulk Flow Coefficient (Kr)</name>
  <short_name>Kr</short_name>
  <unit>m</unit>
  <field_type>textbox</field_type>
  <min>1*10^-6</min>
  <max>1*10^3</max>
  <tooltip_text>Bulk Flow Coefficient (Kr) at Right Boundary</tooltip_tex>
  <default>1*10^-6</default>
</parameter>

<parameter>
  <name>Diffusion Coefficient (Dr)</name>
  <short_name>Dr</short_name>
  <unit>m/s</unit>
  <field_type>textbox</field_type>
  <min>0</min>
  <max>1*10^3</max>
  <tooltip_text>Diffusion Coefficient (Dr) at Right Boundary</tooltip_tex>
  <default>1.5</default>
</parameter>

<parameter>
  <name>Bottom Boundary</name>
</parameter>

<parameter>
  <name>Number of Variables</name>
  <default>3</default>
</parameter>

<parameter>
  <name>Heat Transfer Coefficient (Hb)</name>
  <short_name>Hb</short_name>
  <unit>J/m&lt;sup&gt;2&lt;/sup&gt;/s/k</unit>
  <field_type>textbox</field_type>
  <min>0</min>
  <max>1*10^3</max>
  <tooltip_text>Heat Transfer Coefficient (Hb) at Bottom Boundary</tooltip_tex>
  <default>35</default>
</parameter>

<parameter>
  <name>Bulk Flow Coefficient (Kb)</name>
  <short_name>Kb</short_name>
  <unit>m</unit>
  <field_type>textbox</field_type>
  <min>3.3*10^-13</min>
  <max>1*10^3</max>
  <tooltip_text>Bulk Flow Coefficient (Kb) at Bottom Boundary</tooltip_tex>
  <default>3.3*10^-13</default>
</parameter>

<parameter>
  <name>Diffusion Coefficient (Db)</name>
  <short_name>Db</short_name>
```

```
    <unit>m/s</unit>
    <field_type>textbox</field_type>
    <min>0.5*10^-6</min>
    <max>1*10^3</max>
    <tooltip_text>Diffusion Coefficient (Db) at Bottom Boundary</tooltip_tex>
    <default>0.5*10^-6</default>
  </parameter>

  <help_text>None</help_text>
</parameter_section>

<parameter_section>
  <title>Compression Properties</title>

  <parameter>
    <name>Yield Strain (Epsy)</name>
    <short_name>Epsy</short_name>
    <field_type>textbox</field_type>
    <min>0.015</min>
    <max>0.015</max>
    <tooltip_text>Yield Strain (Epsy)</tooltip_tex>
    <default>0.015</default>
  </parameter>

  <parameter>
    <name>Expansion Ratio (mu)</name>
    <short_name>mu</short_name>
    <field_type>textbox</field_type>
    <min>0.05</min>
    <max>0.05</max>
    <tooltip_text>Expansion Ratio (mu)</tooltip_tex>
    <default>0.05</default>
  </parameter>

  <help_text>Compression Properties -&lt;br/&gt;&lt;br/&gt;None</help_text>
</parameter_section>

<parameter_section>
  <title>Steering Parameters</title>

  <parameter>
    <name>Current Press Closing Time (Current PCT)</name>
    <short_name>CURRENTPCT</short_name>
    <unit>s</unit>
    <field_type>textbox</field_type>
    <min>10</min>
    <max>80</max>
    <tooltip_text>Current Press Closing Time (Current PCT)</tooltip_tex>
    <default>50</default>
  </parameter>

  <parameter>
    <name>New Press Closing Time (New PCT)</name>
    <short_name>NEWPCT</short_name>
    <unit>s</unit>
    <field_type>textbox</field_type>
```

```
    <min>10</min>
    <max>80</max>
    <tooltip_text>New Press Closing Time (New PCT)</tooltip_tex>
    <default>50</default>
  </parameter>

  <help_text>Look at the diagram and change the NEW PCT</help_text>
</parameter_section>

<parameter_section>
  <title>Output Style</title>

  <parameter>
    <name>Output Style</name>
    <short_name>outputstyle</short_name>
    <field_type>radio</field_type>
    <value>Screen (Color)</value>
  </parameter>

  <parameter>
    <name>Output Style</name>
    <short_name>outputstyle</short_name>
    <field_type>radio</field_type>
    <value>Print (B&amp;amp;W)</value>
  </parameter>

  <parameter>
    <name>Output Style</name>
    <short_name>outputstyle</short_name>
    <default>Screen (Color)</default>
  </parameter>

  <help_text>Output Style -&lt;br/&gt;&lt;br/&gt;The output plots are either depicted
    in color which are optimized for display on the monitor or in black and white
    for printing.</help_text>
</parameter_section>

<parameter_section>
  <title>3D Profiles</title>

  <parameter>
    <name>Time Intervals (s)</name>
    <short_name>ticheck</short_name>
    <field_type>checkbox</field_type>
    <default>checked</default>
  </parameter>

  <parameter>
    <name>Time Intervals (s)</name>
    <short_name>tidropdown</short_name>
    <field_type>dropdown</field_type>
    <dropdown_item>5</dropdown_item>
    <dropdown_item>10</dropdown_item>
    <dropdown_item>15</dropdown_item>
    <dropdown_item>20</dropdown_item>
    <dropdown_item>30</dropdown_item>
```

```
    <dropdown_item>60</dropdown_item>
    <dropdown_item>120</dropdown_item>
    <dropdown_item>180</dropdown_item>
    <dropdown_item>240</dropdown_item>
    <default>20</default>
  </parameter>

  <help_text>3D Profiles -&lt;br/&gt;&lt;br/&gt;The 3D profiles are powerful graphical
    tools to have a better understanding of the relative magnitude and interaction
    of the different physical processes during the hot-pressing. A set of profiles
    from selected model variables are generated and animated to provide a better
    understanding of the dynamic processes present in the vertical midplane of the
    mat. The time interval between the output frames are defined by the user. The
    shorter the time interval the more frames are generated which can increase the
    simulation time.</help_text>
</parameter_section>

<parameter_section>
  <title>2D Sections and Functions</title>

  <parameter>
    <name>2D Sections of:</name>
  </parameter>

  <parameter>
    <name>Number of Variables</name>
    <default>7</default>
  </parameter>

  <parameter>
    <name>Temperature</name>
    <short_name>temperature</short_name>
    <field_type>checkbox</field_type>
    <default>checked</default>
  </parameter>

  <parameter>
    <name>Moisture Content</name>
    <short_name>moisturecontent</short_name>
    <field_type>checkbox</field_type>
    <default>checked</default>
  </parameter>

  <parameter>
    <name>Adhesive Cure Index</name>
    <short_name>adhesivecureindex</short_name>
    <field_type>checkbox</field_type>
    <default>checked</default>
  </parameter>

  <parameter>
    <name>Total Pressure</name>
    <short_name>totalpressure</short_name>
    <field_type>checkbox</field_type>
    <default>checked</default>
  </parameter>
```

```
<parameter>
  <name>Partial Air Pressure</name>
  <short_name>partialairpressure</short_name>
  <field_type>checkbox</field_type>
  <default><default>
</parameter>

<parameter>
  <name>Partial Vapor Pressure</name>
  <short_name>partialvaporpressure</short_name>
  <field_type>checkbox</field_type>
  <default><default>
</parameter>

<parameter>
  <name>Relative Humidity</name>
  <short_name>relativehumidity</short_name>
  <field_type>checkbox</field_type>
  <default><default>
</parameter>

<parameter>
  <name>as Function of Time at:</name>
</parameter>

<parameter>
  <name>Number of Variables</name>
  <default>4</default>
</parameter>

<parameter>
  <name>Mat Core and Surface</name>
  <short_name>matcoreandsurface</short_name>
  <field_type>checkbox</field_type>
  <default>checked</default>
</parameter>

<parameter>
  <name>Probe Positions</name>
  <short_name>probepositions</short_name>
  <field_type>checkbox</field_type>
  <default><default>
</parameter>

<parameter>
  <name>Vertical Direction</name>
  <short_name>verticaldirectionT</short_name>
  <field_type>checkbox</field_type>
  <default>checked</default>
</parameter>

<parameter>
  <name>Horizontal Direction</name>
  <short_name>horizontaldirectionT</short_name>
  <field_type>checkbox</field_type>
```

```
    <default><default>
  </parameter>

  <parameter>
    <name>as Function of Postion at:</name>
  </parameter>

  <parameter>
    <name>Number of Variables</name>
    <default>2</default>
  </parameter>

  <parameter>
    <name>Vertical Direction</name>
    <short_name>verticaldirectionP</short_name>
    <field_type>checkbox</field_type>
    <default>checked</default>
  </parameter>

  <parameter>
    <name>Horizontal Direction</name>
    <short_name>horizontaldirectionP</short_name>
    <field_type>checkbox</field_type>
    <default><default>
  </parameter>

  <help_text>2D Sections -&lt;br/&gt;&lt;br/&gt;The output profiles of the variables
    can be cut with an imaginary plane in the middle, either vertically or
    horizontally, creating projections of the variables. The information can be
    presented in two different ways, either displaying the variables as function of
    time using the location as the parameter, or as a function of location using
    time as the parameter. The two approaches although representing the same data,
    can emphasize completely different aspects of he mechanisms
    involved.&lt;br/&gt;&lt;br/&gt;The user selects the variables of interest in the
    first column and the projection method in the second column. The variables can
    be depicted as function of time by defining some or all of the provided four
    locations. The most widely used and recommended location is the surface (top)
    and core of the mat. The second method allows to position six probes by defining
    the locations of the probes as a fraction of the mat dimensions. Additionally,
    the variables can be projected in the vertical direction (through the thickness
    of the mat) or the horizontal direction (through the width of the mat) at the
    mesh point locations. The selected variables can be depicted as function of
    location in the mat at every minute at either the vertical or horizontal
    direction by defining the last two available projection methods.</help_text>
</parameter_section>

<parameter_section>
  <title>Averages and Compression</title>

  <parameter>
    <name>Averages</name>
  </parameter>

  <parameter>
    <name>Number of Variables</name>
    <default>3</default>
```

```
</parameter>

<parameter>
  <name>Mat Temperature</name>
  <short_name>mattemperatureA</short_name>
  <field_type>checkbox</field_type>
  <default>checked</default>
</parameter>

<parameter>
  <name>Mat Moisture Content</name>
  <short_name>matmoisturecontentA</short_name>
  <field_type>checkbox</field_type>
  <default>checked</default>
</parameter>

<parameter>
  <name>Adhesive Cure Index</name>
  <short_name>adhesivecureindexA</short_name>
  <field_type>checkbox</field_type>
  <default>checked</default>
</parameter>

<parameter>
  <name>Compression</name>
</parameter>

<parameter>
  <name>Number of Variables</name>
  <default>3</default>
</parameter>

<parameter>
  <name>Vertical</name>
  <short_name>verticalC</short_name>
  <field_type>checkbox</field_type>
  <default>checked</default>
</parameter>

<parameter>
  <name>Horizontal</name>
  <short_name>horizontalC</short_name>
  <field_type>checkbox</field_type>
  <default>checked</default>
</parameter>

<parameter>
  <name>Other Outputs</name>
  <short_name>otheroutputsC</short_name>
  <field_type>checkbox</field_type>
  <default>checked</default>
</parameter>

<help_text>Averages -&lt;br/&gt;&lt;br/&gt;The average temperature, moisture content
  and adhesive cure index can be plotted as function of time during the press
  schedule. The rate of the heating up of the mat the rate of the depletion of
```

```
    moisture from the mat and the extent of the cure of the thermosetting adhesive
    can be followed by the help of these graphs. The average plots are valuable to
    compare different pressing methods.&lt;br/&gt;&lt;br/&gt;&lt;br/&gt;Compression
    - &lt;br/&gt;&lt;br/&gt;The model predicts the final vertical density profile
    and horizontal density distribution of the
    mat.&lt;br/&gt;&lt;br/&gt;&lt;br/&gt;Other Outputs - &lt;br/&gt;&lt;br/&gt;The
    ram pressure during the press schedule and the springback of the mat during the
    press opening are generated.</help_text>
</parameter_section>

<parameter_section>
  <title>Probe Positions</title>

  <parameter>
    <name>Probe#</name>
  </parameter>

  <parameter>
    <name>Number of Variables</name>
    <default>6</default>
  </parameter>

  <parameter>
    <name>Field 0</name>
    <default>0</default>
  </parameter>

  <parameter>
    <name>Field 1</name>
    <default>1</default>
  </parameter>

  <parameter>
    <name>Field 2</name>
    <default>2</default>
  </parameter>

  <parameter>
    <name>Field 3</name>
    <default>3</default>
  </parameter>

  <parameter>
    <name>Field 4</name>
    <default>4</default>
  </parameter>

  <parameter>
    <name>Field 5</name>
    <default>5</default>
  </parameter>

  <parameter>
    <name>y()</name>
  </parameter>
```

```
<parameter>
  <name>Number of Variables</name>
  <default>6</default>
</parameter>

<parameter>
  <name>Field 0</name>
  <short_name>y0</short_name>
  <field_type>textbox</field_type>
  <default>3</default>
</parameter>

<parameter>
  <name>Field 1</name>
  <short_name>y1</short_name>
  <field_type>textbox</field_type>
  <default>1</default>
</parameter>

<parameter>
  <name>Field 2</name>
  <short_name>y2</short_name>
  <field_type>textbox</field_type>
  <default>2</default>
</parameter>

<parameter>
  <name>Field 3</name>
  <short_name>y3</short_name>
  <field_type>textbox</field_type>
  <default>3</default>
</parameter>

<parameter>
  <name>Field 4</name>
  <short_name>y4</short_name>
  <field_type>textbox</field_type>
  <default>5</default>
</parameter>

<parameter>
  <name>Field 5</name>
  <short_name>y5</short_name>
  <field_type>textbox</field_type>
  <default>4</default>
</parameter>

<parameter>
  <name>z()</name>
</parameter>

<parameter>
  <name>Number of Variables</name>
  <default>6</default>
</parameter>
```

```
    <parameter>
      <name>Field 0</name>
      <short_name>z0</short_name>
      <field_type>textbox</field_type>
      <default>5</default>
    </parameter>

    <parameter>
      <name>Field 1</name>
      <short_name>z1</short_name>
      <field_type>textbox</field_type>
      <default>4</default>
    </parameter>

    <parameter>
      <name>Field 2</name>
      <short_name>z2</short_name>
      <field_type>textbox</field_type>
      <default>6</default>
    </parameter>

    <parameter>
      <name>Field 3</name>
      <short_name>z3</short_name>
      <field_type>textbox</field_type>
      <default>3</default>
    </parameter>

    <parameter>
      <name>Field 4</name>
      <short_name>z4</short_name>
      <field_type>textbox</field_type>
      <default>2</default>
    </parameter>

    <parameter>
      <name>Field 5</name>
      <short_name>z5</short_name>
      <field_type>textbox</field_type>
      <default>2</default>
    </parameter>

    <help_text>Probe Positions -&lt;br/&gt;&lt;br/&gt;It is specified in the table to the
      right.  If Probe Positions checkbox is not checked, it will always use the
      default values instead.  Another word, the proper way to use this feature is to
      check the checkbox first, and then input the fields in the Proble Positions
      table and click "Save".</help_text>
  </parameter_section>

</simulation>
```

## Appendix F: Fortran 90 Sample Steering Module `modsteering.f`

The following Fortran 90 code is a sample steering module, which illustrates how to use a two-level loop (tries and max wait time) to control the steering process, as discussed in Chapter 5.

```
      Module ModSteering
      Use REAL_PRECISION, Only: R8
C
C
C*********************************************************************
C
C
C*********************************************************************
C
      Contains
C
      Subroutine Steering_Sample
C
      Use ModGlobal, Only:iteration, max_wait_time, wait_time
C
      Implicit None
      Integer tries
      Integer time_waited
      Integer received_input
      Logical file_exist
      Real(KIND=R8) steering_paramter_1, steering_paramter_2, steering_paramter_3
      Real(KIND=R8) steering_output_1, steering_output_2, steering_output_3
      Real(KIND=R8) steering_input_1, steering_input_2, steering_input_3
C
      NameList/Steering_Input/
     $     steering_input_1,
     $     steering_input_2,
     $     steering_input_3
C
      tries = 0

      Do While(tries.lt.iteration)

         steering_output_1 = steering_paramter_1
         steering_output_2 = steering_paramter_2
         steering_output_3 = steering_paramter_3

         Open(Unit=810,Status='Replace',File='steering_output.dat')
         Write(810,*)'steering_output_1',steering_output_1
         Write(810,*)'steering_output_2',steering_output_2
         Write(810,*)'steering_output_3',steering_output_3
         Close(Unit=810)

         time_waited = 0
         received_input = 0

         Do While ((time_waited.lt.max_wait_time).and.
         $ (received_input.ne.1))
```

```
          inquire(file='steering_input.dat',exist=file_exist)

          If (file_exist) Then

            Open(Unit=820,File='steering_input.dat',Status='Old')
            Read(Unit=820, NML=Steering_Input)
            Close(Unit=820)

            steering_paramter_1 = steering_input_1
            steering_paramter_2 = steering_input_2
            steering_paramter_3 = steering_input_3

            received_input = 1

          Else

            Write(*,*)'time_waited='
            Write(*,*)time_waited

            call sleep(wait_time)

            time_waited = time_waited + wait_time

          EndIf

        EndDo

        tries = tries + 1

      EndDo
C
      Return
      End Subroutine Steering_Sample
C
C*********************************************************************
C
      End Module ModSteering
```