HIERARCHICAL STRATEGY FOR RAPID FINITE ELEMENT ANALYSIS

A Thesis

by

JULIAN VARGHESE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2003

Major Subject: Aerospace Engineering

HIERARCHICAL STRATEGY

FOR

RAPID FINITE ELEMENT ANALYSIS

A Thesis

by

JULIAN VARGHESE

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

John D. Whitcomb (Chair of Committee)	Junuthula N. Reddy (Member)
Dimitris C. Lagoudas (Member)	Walter E. Haisler (Interim Head of Department)

December 2003

Major Subject: Aerospace Engineering

ABSTRACT

Hierarchical Strategy for Rapid Finite Element Analysis. (December 2003)

Julian Varghese, B.Tech, University of Kerala, Kerala, India

Chair of Advisory Committee: Dr. John D. Whitcomb

A new methodology is introduced where the natural hierarchical character of model descriptions and simulation results are exploited to expedite analysis of problems. The philosophy and the different concepts involved are illustrated by implementing the strategy to solve some practical problems. The end result was a mix of mechanics, well-designed data structures and software interfaces that forms a rapid analysis environment. This can be very advantageous for cases where a sequence of analyses is required because of safety concerns or cost.

When designing a structure, it is common to make frequent modifications to the model during the process. In such cases, the ability to use data from different models within the same analysis environment becomes a major advantage. The proposed system's forte is its hierarchical framework that allows models to communicate with each other and share information with one another. This makes it ideal for global local analyses where solutions from a global model are used to derive the boundary conditions for the local model.

The system was also used to conduct a micro mechanical analysis on unidirectional composites that have a non-uniform spatial distribution of the fibers. The hierarchical strategy is not tied to any specific methodology and can be adapted to solve problem using different technologies. This allows the strategy to be used across multiple length scales and governing equations.

To my parents, for their love, patience and sacrifices

ACKNOWLEDGEMENTS

I wish to express my deep sense of gratitude to my advisor, Dr. John D. Whitcomb, for supporting me financially, morally and academically. Without his patience and constant guidance, this work would have never been complete. I really appreciate the time he has taken out of his personal life to help me out. The care and genuine regard for the well being of his students is something that can be found in very few people. I also wish to thank him and his lovely family for inviting me to thanksgiving dinner and for making sure we have a great time every year.

I am thankful from the bottom of my heart to Dr. Xiaodong Tang for being my mentor and teammate in this research work and above all for being such a nice friend. I was very fortunate to get a chance to work with him. He played a very important role in this project and his contribution provides the backbone for my research work. I will always be indebted to him for his tremendous help throughout this research. I also acknowledge his help allowing me to use several in-house computer codes.

I would like to express my gratitude to Dr. J.N. Reddy and Dr. Dimitris Lagoudas for serving on my thesis committee and for providing the valuable time from their busy schedules. I also want to thank Dr. Vikram Kinra for readily offering to provide photographs that were used in this thesis. I also appreciate the help provided by the University Writing Center in writing this thesis.

My sincere thanks also go to Aerospace Engineering Department staff, especially Ms. Karen Knabe, Pam McConal and Ms. Donna Holick, who are among the sweetest people I have met in my life, for their kind help during my graduate studies here at Texas A&M.

A whole bunch of thanks goes to my teammate, classmate and a real good buddy-Deepak Goyal. He has been of immense help all throughout my graduate studies as well as outside work. I want to thank my office-mate and teammate, Jong-Il Lim for being such a caring and understanding friend. I wish to express my gratitude to Dr. Jae Noh

who is probably the first person I met in our research team. His experience and knowledge has been of great help. I want to thank the rest of my research team members: Seung Don, Woosick and the latest members, Ji-woong, Bryan and Bhavya, for providing a conducive environment for this research work. It was great being part of such a lively and friendly group of people. I am also thankful to all of my friends, especially Sandeep, Jiney, Nishanth, Pillai, Pratheesh, Elango, Bharat, Barani, Atul and Sudharsan for their support. I am thankful to all the people who have directly or indirectly helped me accomplish whatever I have.

Finally, I wish to express my sincere appreciation to my father, mother and sister for their never-ending support, love, prayers and sacrifices. Without them, I would not have been able to pursue graduate studies here at Texas A&M University.

This work is based on research supported by NASA through grant # NAG-1-01080 for “Hierarchical Strategy for Rapid Analysis Environment”. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Aeronautics and Space Administration. I also acknowledge the support of the Texas Institute for Intelligent Bio-Nano Materials and Structures for Aerospace Vehicles, funded by NASA Cooperative Agreement No. NCC-1-02038.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Page

Page

1. INTRODUCTION

Overview

Depending on its complexity, analysis of a scientific problem can be very cumbersome and time-consuming. Even if the governing equations are determined, it is usually not possible to arrive at a closed form solution of the problem. In such cases, one of the work-around solutions is to use numerical techniques to solve the equations involved. This might not give you an exact solution, but depending on how you solve the equations, it is possible to get a practical solution for the problem.

The drawback of numerical solving of problems is that it is a tedious process and becomes hard to manage with increase of complexity. For this reason, it is not practical to solve large problems numerically by hand. With the advent of computers, the time required to analyze large problems has been considerably reduced. In addition to the advantage of a much higher processing speed, the computer takes care of the 'book-keeping'. The finite element method, which is basically a numerical technique to solve partial differential equations, gained popularity with the development of fast computers. Although certain key features can be found in earlier works, the presentation of the finite element method is attributed to Argyris and Kelsey (1960) and Turner, Clough, Martin and Topp (1956). With this method, it became easy to analyze structural components with complex shapes. Although, it was initially used to analyze structural problems, the finite element method is currently used in many fields of science as well as business and finance engineering [1].

Gone are the times when you needed a supercomputer to analyze a problem. With the explosive growth of the computer industry, it is now possible to run a fairly

---

This thesis follows the style and format of *Journal of Composite Materials.*

large model on a desktop PC. Although, an increase in computer processing speed reduces the time taken for solving a set of equations, there are other bottlenecks that hold up the time required for an analysis. Ironically, with the availability of very fast computers, some of the major bottlenecks involved in analyzing a problem from start to finish are those processes that require human involvement.

There are many factors that govern how quickly one can perform analysis of a structural configuration. When using finite elements, the steps typically consist of defining a solid model, converting the solid model into a finite element mesh, preparing the non-geometric data input (such as material properties and boundary conditions), using a finite element solver to solve the equations and provide the solution in terms of displacements, stresses, etc., and most importantly interpret the results. Each of these steps can be quite time consuming in terms of computing power and human involvement. The aim of this work is to explore the hierarchical aspects involved in analyzing a problem and exploiting them to reduce the analysis time.

Literature Review

The term "hierarchical" is used to refer to different types of ideas in the literature. It is important to know what we mean by this in order to understand the uniqueness of this philosophy and the differences when compared to other proposed methodologies. This section talks about the relevance of hierarchy, the finite element method in general and some of its various versions, especially those that deal with some sort of hierarchy. It mentions the bottlenecks faced when conducting an analysis and other common problems facing the finite element method.

There is an aspect of hierarchy/inheritance in both science and almost every thing we see in day-to-day life. The whole theory of genetics and evolution of life itself is based upon inheritance. In the same way, the human race can be considered a huge hierarchy with every human being related to one another in some way albeit along a rather long path. Inheritance is most evident in reproduction where properties of both parents are combined to obtain a new offspring that has inherited properties as well as

some new properties of its own. Certain characteristics such as the blood group have to follow a strict inheritance rule. For example, a combination of an A type and B type from the parents can only produce a child with an AB blood type. An important application of genetics that is gaining a lot of recognition now is DNA fingerprinting. Using this system, it is possible to determine the identity of a person by using just a small sample of his/her DNA and comparing it with a sample from a relative on the mother's side. This is based on the fact that some DNA in a person is inherited from the maternal parent. It is also common knowledge that apart from the genetics, the environment and culture in which a human being grows up affects the behavior of the person. The person assumes or acquires some of the properties of his/her surrounding environment. For example, a growing child's behavior is easily molded by the actions of the people that he or she is most in contact with.

The idea of re-use or inheritance is also seen in the way we think or work. When we try to solve a problem, our mind automatically starts to try to relate the problem to another similar problem that is already solved. We seldom start with a blank slate. We try to make use of some previous experience and make modifications to the previous solution to help solve the current problem. The mind tries to seek out an analogy with an existing solution and go from there. When teaching new material to students in school, students love example problems. That way the students get a feel for how to solve this type of problem. Once they get this experience, it is easier for them to solve a similar kind of a problem. When a child sees a new animal for the first time, its response would be to first recognize it as a live creature that moves. Then it recognizes that it has similar but at the same time weird-looking features like eyes, nose, ears etc. If it is a tiger, the child might draw an analogy to its pet dog. Then it sees that it is much bigger in size and has different patterns on its body. This way, the child gradually learns that it is a tiger by making more additions to its assessment of the creature in front of it.

Apart from biological systems, we also see hierarchy in materials in general, inanimate or not. We can say the smallest building blocks of materials are atoms, or in a

stricter sense, the number of atomic sub-components which seem to growing day by day with new discoveries. But the arrangement of these particles in a particular way gives new materials with special properties. As we go from the atomic scale to macro scale, we see a wide variety of materials with different properties. It is common now to determine macroscopic properties somewhat by micro or nano or even atomistic scale properties. But some factors do not appear at the atomistic scale. For example, defects and grain boundaries in materials cannot be explained from an atomistic level. Diamond and soot are both made of carbon and so some properties are inherited, but its macroscopic properties are totally different because "other factors" come into play. Carbon nanotubes are an up and coming material that is being researched quite extensively due to its high strength and stiffness. But it is hard to model because other phenomena like quantum mechanics come into play in the nano-scale and conventional continuum mechanics is not valid at such scales. So, research on materials with carbon nanotubes span over different scales and different governing equations.

The most common programs that we use these days are built using object–oriented programming, which is based on the idea of inheritance. Fields such as pattern recognition, artificial intelligence and computer networks make use of hierarchy extensively [2]. Hierarchical networks are also implemented in a class of control systems that can be used for space navigation [3].

The finite element method itself can be considered a 2-level hierarchy in the sense that the problem field is subdivided into smaller elements. Oden et al. [4] introduced the concept of hierarchical modeling as an approach to overcome the difficulties of multiscale modeling. In this methodology, a hierarchy of descriptions of the physics of the problem is first set up, ranging from the coarsest possible description to the most detailed description contained in the class of models. Thus in this context, it deals with scale hierarchy.

In this work, the term hierarchical strategy is used to convey the idea that analysis models can be organized and managed hierarchically in order to rapidly set up a

new analysis model. New models are derived from the base model whose information is either inherited or overridden by the new model. Thus, we are dealing with a hierarchy of models.

The global/local analysis method is inherently hierarchical in nature if you consider the local model is actually a more refined part of the global model. Thus, it makes sense to manage data in a hierarchical form to tap into the full potential of the global/local analysis method.

Much of the analyses carried out by industry and academia uses global/local analysis. The global/local technique in some sense has been around since before the finite element method was developed. This technique comes in very handy when designing large complex structures such as aircraft and automobiles where large finite element models of these structures are utilized. These models are useful in obtaining a more accurate response to load conditions or for optimizing different characteristics. But there is a practical limit to the amount of refinement that these model can hold simply due to the fact that the computational cost for such an endeavor would be too much. In such cases, separate analysis is done on localized regions of the structure where the refinement is high enough to obtain a reliable design. Global/local analysis is also used in fracture mechanics to calculate the stress intensity factor for cracks [5]. Iterative global/local finite element analysis is found to be less taxing on computer memory requirements [6]. The global local method is also used in the failure analysis of textile composites [7].

A number of strategies have been developed to enhance finite element solutions in the regions of high gradients. In the h-method, the finite element mesh is refined by keeping the elements of the same order and subdividing them. In the p-method [8], the same mesh is retained but the order of the interpolation function (approximation) is increased. The third method (h-p) is a combination of the first two strategies. The h-p version uses a simultaneous increase of the polynomial degree and mesh refinement. The

rates of convergence for the h-, p- and h-p versions in terms of the number of degrees of freedom have been identified and quantified by Babuska and Szabo [9].

The hierarchical finite element method (or HFEM) [10,11] belongs to the p-version of the FEM. In HFEM, the order of approximation (interpolation) functions are hierarchically increased – the new order of approximation is based on the lower order that is previously constructed.

A version of the finite element called the s-version, where s stands for superpositioning, was introduced by Fish[12]. The basic idea of this method is that a portion of the finite element mesh in which steep gradients are indicated by the solution is overlaid by a patch of higher-order hierarchical elements. Fish and Guttal [13] developed the s-version of the finite element method for laminated plates and shells. In their technique, the global domain is idealized using a 2-D Equivalent Single Layer (ESL) model and the location of the critical regions where a Discrete Layer (DL) model is needed is identified using Dimensional Reduction Error (DRE) indicators. These regions are superimposed by a stack of 3D elements (DL model) and thus both the local and global effects are predicted. Fish [14] also used the s-version to hierarchically model discontinuous fields such as for crack propagation. This is achieved by overlaying portions of the finite element mesh where discontinuities need to be embedded with a finite element mesh that is discontinuous across the crack. This also proves to be computationally efficient due to the hierarchical nature of the method where the base mesh can be fixed and only the super-imposed mesh needs to be modified.

Another strategy to enrich finite element solutions is by adding special shape functions that are known to approximately model the behavior of the exact solution. This idea was introduced by Mote[15] who developed a global-local Finite Element where a combined global and local dependant variable representation couples the conventional and finite element Ritz methods.

Voletia et al [16] address the use of global/local FEM to analyze large-scale periodic structures made up of multi-material composite systems. They explain two different techniques – the specified boundary method and the multi-point constraint method. The global/local FEM techniques prove to be faster especially as the size of the problem increases.

Sun and Mao [17] proposed a refined global local finite element analysis method which involves 3 steps to improve the efficiency of the analysis. The global analysis of a coarse mesh provides a displacement solution, which is then used in the local analysis for computing detailed stresses using refined meshes. Finally, a refined global analysis is conducted to improve the accuracy of both displacements and stresses.

Whitcomb [18] described the process of iterative global/local finite element analysis where the accuracy is retained by using an iterative procedure to enforce equilibrium between the global and local regions. Babuska et al [19], Reddy and Robbins [20] have done some work on the reliability, convergence and accuracy aspect of global/local techniques. N.F. Knight, Jr. et al [21] present a global/local analysis methodology for obtaining the detailed stress state of structural components. In this methodology, a global model that is a finite element model of an entire structure or a subcomponent of a structure is first analyzed. A region requiring a more detailed interrogation is then identified and a local model is generated for that region. The result from the global model is then used to provide the boundary conditions for the local model with an interpolation procedure. Noor et al [22] describes two predictor-corrector procedures for the accurate determination of the global as well as detailed response characteristics of plates and shells.

Ghosh et al [23] proposed a hierarchical multi-scale computational model for damage in composite materials. Analysis is done at the structural and micro-structural scales. The microscopic analysis is conducted using a voronoi cell finite element model (VCFEM) while a conventional displacement based FEM code executes the macroscopic

analysis. A simple plate with a hole problem was analyzed hierarchically using three different refinements levels.

Noor et al [24] used hierarchical sensitivity analysis to identify the parameters that have the most effect on the non-linear response of composite structures. Their modeling approach used for multilayered panels can be divided into different categories that cover a wide range of length scales from local to global structural response: detailed micromechanical three-dimensional continuum models, quasi-three-dimensional models, and two-dimensional plate and shell models. The nonlinear response of the structure is dependent on a hierarchy of interrelated geometric and material parameters at these different categories. The sensitivity of the response to variations in these parameters at each level provides insight into the importance of the parameters and helps in the development of materials to meet certain performance requirements. Ransom and Knight [25] discuss a methodology for the global/local stress analysis of composite panels. Noor et al [26] discusses different global/local methodologies and their application to non-linear analysis.

J. Fish et al [27] developed a hierarchical version of the composite grid method (denoted as HFAC), which exploits the solution of the shell model in studying local effects via a 3D solid model. It is hierarchical in the sense that information from the analysis of an equivalent single layer (ESL) model is exploited in the resolution of local effects using a discrete layer (DL) model. The multigrid and composite grid methods [28,29,30,31,32,33] are a widely used hierarchical global/local strategy. Some works on methods based on hierarchical decomposition of the approximation space are described in [34,35,36,37,38].

Woo and Whitcomb [39] developed macro-elements where the regions in an element could have heterogeneous material properties. Thus, it permits microstructure within a single element. This is useful for analyzing some heterogeneous material where it is not practical to model the microstructure directly using traditional finite elements and it might not be accurate to use homogenized properties.

Different kinds of problems can be analyzed by using the proposed system. The hierarchical strategy does not restrict the user to use a specific type of methodology. In addition to using the hierarchical system to analyze structural problems using a global/local methodology, it has also been adapted to analyze unidirectional composites with non-uniform distribution of fibers. When analyzing composite materials, it is common to assume a periodic arrangement of the fibers in the matrix. With this assumption it is sufficient to analyze just the unit cell or even a fraction of the unit cell for problems such as determining effective properties. In reality, composite materials usually do not exhibit such an ideal case of periodicity. Figure 1 shows the photograph of a typical microstructure with non-uniform distribution of fibers in the matrix. In addition to the non-uniformity of the distribution of the fibers, there is also a variation in the size of the fibers.

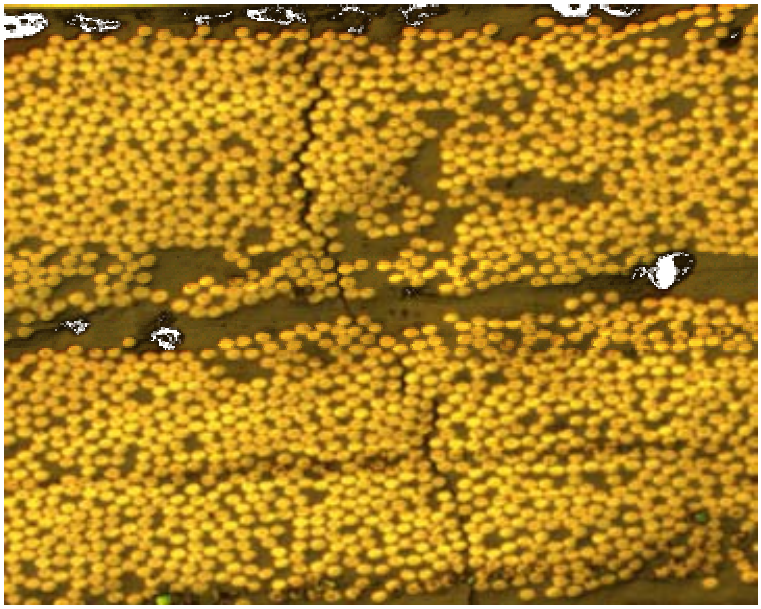

Figure 1: A typical microstructure showing the non-uniform distribution of fibers. (Courtesy Dr.Kinra, Texas A&M University)

The spatial arrangement of the fibers in the matrix affects the properties and the performance of the composite. The non-uniform distribution of the fibers in the matrix is

usually linked to the manufacturing and processing techniques used for making the composites. Whited, Gokhale and Deshpande [40] performed thermal cycling experiments on a metal matrix composite having continuous aligned alumina fibers distributed in the matrix of an Al-Li alloy. They measured the center-to-center spatial distribution of the fiber that have microcracks between them and analyzed this distribution as a function of the number of thermal cycles. It was found that the formation and growth of microcracks strongly depends on the distance between the fibers. The microcracks were more likely to form between the fibers that were closely spaced.

Brockenbrough et al [41] studied the effects of fiber distribution and fiber cross-sectional geometry on the deformation of a metal matrix composite reinforced with continuous fibers using discrete finite element meshes with a random array of fibers. it was found that if the composite had non-periodic distribution of fibers, and is subjected to transverse loading where the applied stresses are transmitted to the fibers via the matrix, analytical and numerical models based on perfectly periodic distributions of the fibers fail to provide accurate estimates of the actually deformation response in the plastic regime. For a high volume fraction of fibers, even the elastic moduli varied markedly for tensile and shear deformation under transverse loading. This proves that analyses based on periodic arrangement of fibers are not capable of capturing the response of transverse deformation of real composites with non-periodic arrangements of fibers.

It has also been shown that fracture of composites [42] depend on the spatial distribution of the fibers in the matrix. Sorensen and Talreja [43] found that the local stresses due to cool-down during processing depend significantly on the spatial distribution of fibers. Yang, Tewari and Gokhale [44] used a digital image analysis technique to quantify the non-uniform spatial arrangement of Nicalon fibers in a ceramic matrix composite (CMC). This quantitative data was then used to generate a computer simulated microstructure model that is statistically equivalent to the non-uniform

microstructure of the CMC. Gokhale, Yang and Shan [45] then used this computer simulated microstructure model for parametric studies on the micro-mechanical response of the composite. They showed that their model accounts for the non-uniform spatial arrangement of fibers having a range of sizes and perform better than models that assume periodic arrangement.

Scope of Research

Even though there are various versions of the finite element method, there are several basic issues that can cause bottlenecks during an analysis that does not deal specifically with a kind of methodology but with the generic finite element method. As the complexity of the problem increases, analysis models increase in size and the amount of data that needs to be handled becomes overwhelming. When designing a structure, it is common to make frequent modifications to the model during the process. A number of analyses must be conducted before adequate information can be obtained to make a good decision regarding a final design. In such cases, the ability to use data from different models within the same analysis environment becomes a major advantage. This calls for interaction between models and at different detail levels. In many cases, the data belonging to a model is shared with other models. For example, data such as mesh information and computational results from one model can be used in another and thus there is a 'flow' of data. This is the case in global/local analyses where a global analysis is required to determine an overall response and this result is used in identifying and conducting analyses on areas that require detailed examination. Thus, data management and control is a big issue that needs to be dealt with. Also, time is an important factor and it is always advantageous to be able to set up analysis model quickly and efficiently. This is especially true for cases where a sequence of analyses is required because of safety concerns or cost. Global/local analyses are a good example of such a case. Parametric studies mandated by the statistical character of manufacturing and service environments also involve a number of analyses. One way to achieve this is by using the computer to automate as many steps as possible that are involved in generating an

analysis model. Some of these functions are boundary detecting and matching. By letting the computer do this work instead of the user manually entering the information, a lot of time can be saved and analyses can be conducted efficiently. It is these and other problems that are addressed in this work. The goal is to develop an environment for rapid analysis using hierarchical description of models and efficient and robust data control mechanisms to solve problems quicker as well as reliably.

The prototype environment is used to approach two different problems. One is to analyze structural problems using a global/local methodology. A simple yet relevant case of rivets holes around a plane fuselage window is considered and its results are discussed.

The environment is also used to conduct a micro mechanical analysis of unidirectional composites with non-uniform distribution of fibers in the matrix. This is interesting because the spatial arrangement of fibers is usually not uniform and this is known to affect the properties and performance of unidirectional fiber reinforced composites. The non-uniform distribution of the fibers is usually attributed to the processing techniques used for making the composites. Therefore, such models could be useful in optimizing the manufacturing processes.

2. HIERARCHICAL SYSTEM

Introduction

This section begins by discussing the basic philosophy of the hierarchical strategy. The different concepts involved in implementing the hierarchical framework will be dealt with in detail. It also talks about the unique features of this system and makes comparisons with similar programs. A simple structural configuration will be used to illustrate the basic concepts involved.

Hierarchical Strategy

A new philosophy is developed wherein the hierarchical definition of data is made use of in creating a better environment to conduct analyses of practical problems. The preliminary work included evaluating current global/local strategies in terms of data requirements, interfaces, extensibility and robustness. Hierarchical data structures and a framework for conducting analyses were designed that would use these new features that exploit the hierarchical nature of models.

The word hierarchy indicates a classification of a group of entities based on some common properties or attributes. When we talk about hierarchy, invariably the term inheritance also comes in to the picture. Inheritance is the acquisition of a property or an attribute from an entity. Thus, there is a hierarchy formed because of some commonality between the two entities based on the property that was inherited. In such a case, the entity that inherits the property is called the ‘child’ and the entity that the property was inherited from is called the ‘parent’. It is also common to use the terms ‘derived’ and ‘base’ respectively to describe the two entities.

In our case, the entities that we deal with are analysis models. New models can be derived from existing models by inheriting certain data, but at the same time maintain its individuality by having its own properties that are different from the existing models. The easiest way to describe the structure of a hierarchy is using the help of a tree diagram. The inheritance tree is used to describe how different analysis models can be

organized and managed hierarchically to rapidly create a new analysis model. This sort of representation will be used through out this work to describe the hierarchy of models.

New models are derived from the base model whose data is inherited, overridden or expanded by the new model. By inheriting data, the derived model establishes the link to the parent model but what defines the model's own unique identity is the data that is overridden or expanded. Data in this context could mean anything from geometric mesh information and load conditions or boundary conditions to even solutions of analysis models. This way the derived model can have new characteristics that are different from its base models. This differentiating data is defined to be a 'component'. A robust as well as efficient mechanism was designed for implementing inheritance, which is the essence of the hierarchy. Figure 2 shows two views of the inheritance tree for a curved panel. This is a simple tree that has no branching. The component view shows the different components (i.e. the differentiating data) involved while the model view shows the complete model at each level.

Figure 3 shows a more complex inheritance tree with multiple branches. The model at each 'node' in the tree is described in terms of all the components along its model path. The model path is the shortest route which links the top-most model in the inheritance tree to the current model. Figure 3 illustrates the meaning of model path of a model. In this case, the model path for the model FRCd is the route traced by the models - LC, H2, H1, H1C, H1RC and finally FRCd. Any other route would require retracing through a model that had already been covered by the path. Thus, the derived model FRCd is a combination of the component at FRCd and the components of all the models in its model path just mentioned. The figure shows two different views of the inheritance tree. The component view shows the corresponding components at the nodes in the inheritance tree. In the model view, each node in the tree is associated with a complete model.

One of the most important aspects of this philosophy is the sharing of data by models in the hierarchy. This feature makes it ideal for cases like global/local analysis where results from a global analysis are used to provide the boundary conditions for the

local model. Efficient data flow mechanisms are required and complex recursive functions were developed that traverse the hierarchical tree to implement this. This kind of recursive strategy can be used to access data belonging to any model in the hierarchy. The recursive strategy is designed such that it can be used to perform tasks on particular models or a collection of models in the hierarchy without making modifications to the mechanism. This mechanism gives a model in the hierarchy the ability to 'interrogate' another model for information.

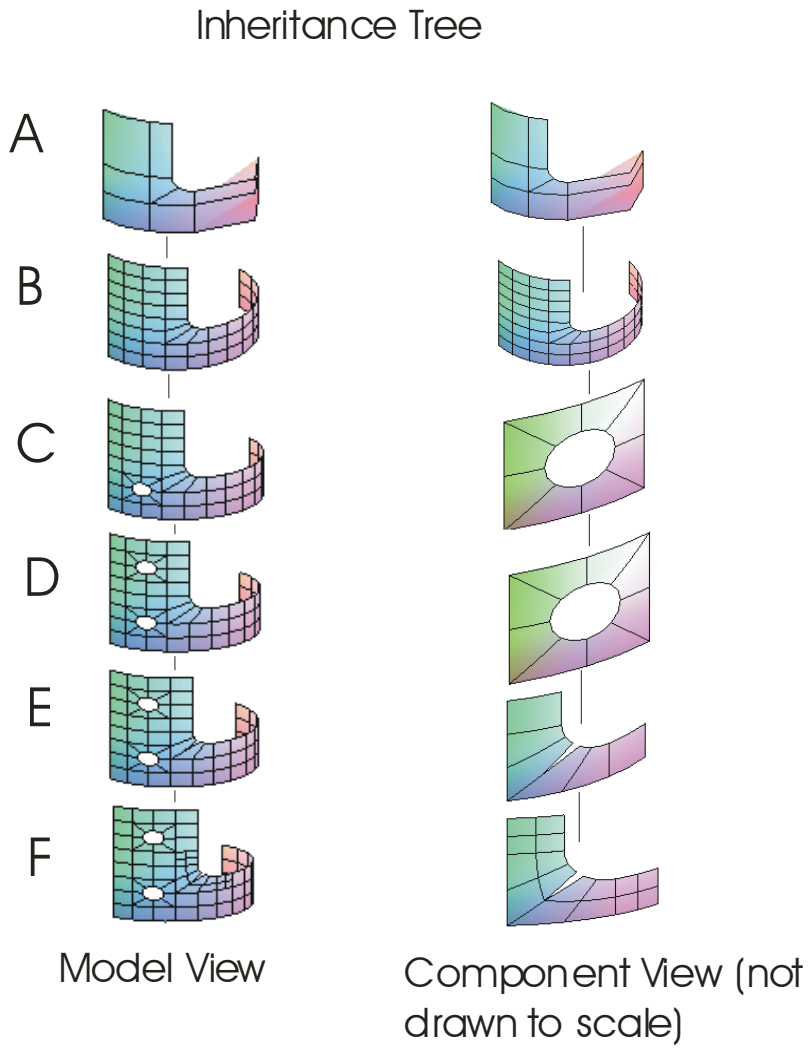

Figure 2: Inheritance tree with no branching (Model view and component view)

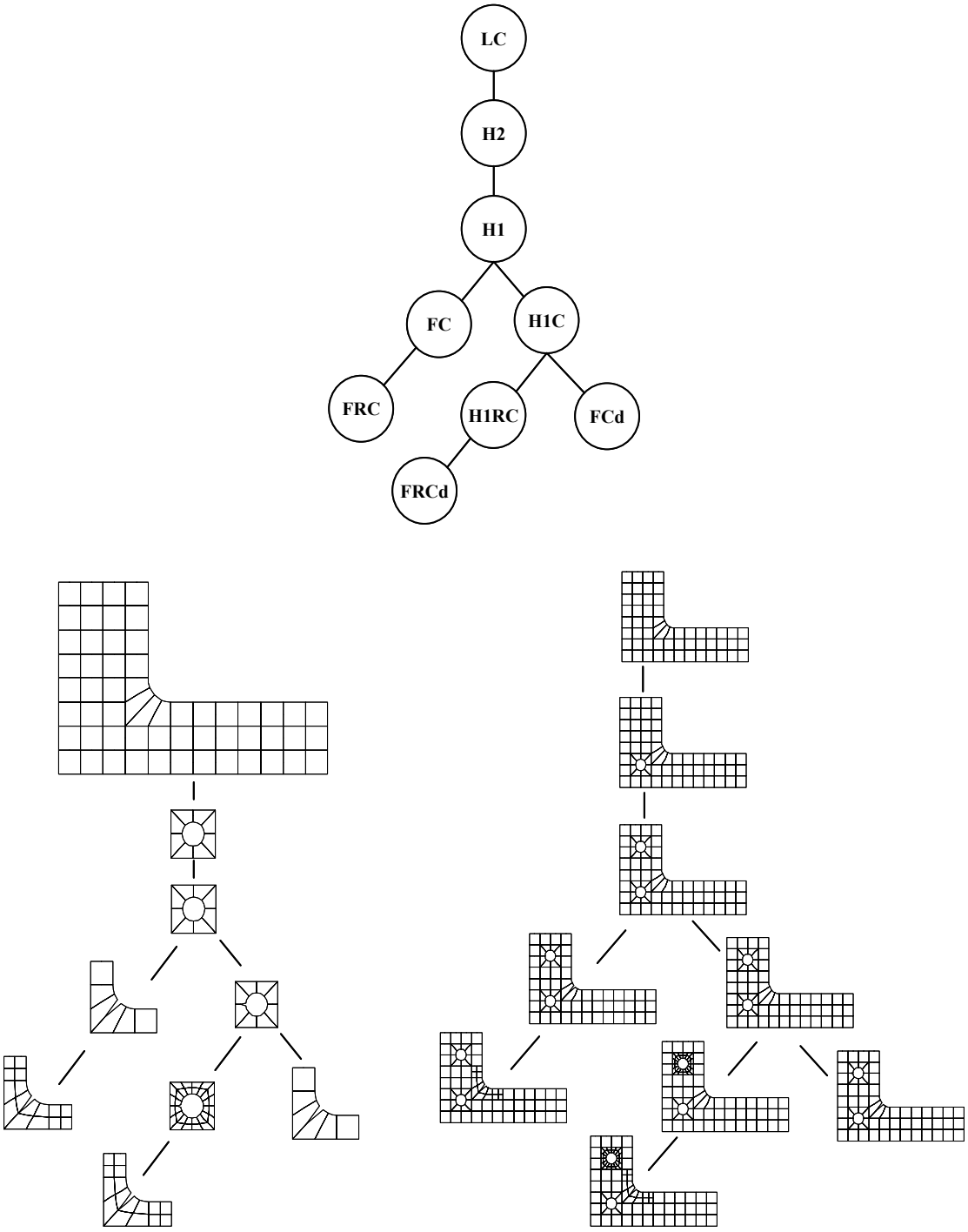

Figure 3: Inheritance tree with multiple branches

Comparison with Other Software

Existing commercial finite element analysis software that use some kind of hierarchy was compared to the philosophy described in this work. In DesignSpace by AnSys[46] , the ability to combine components to build models is limited in the sense that each hierarchy (or 'tree structure') represents a single model. In the Hierarchical System, each node in the model hierarchy defines a complete model and not just a component. The hierarchical system provides a framework that has the 'intelligence' for building a hierarchy of models. SIMBA (Simulation Manager and Builder for Analysts), developed by Sandia Labs [47], also builds FE models from various components but it does not address data flow between different models in the hierarchy. Using NextGRADE by NASA [48], which stands for Next Generation Rapid Analysis and Design Environment, you can rapidly build a structure using stock components and then analyze it. But even here, they are addressing only one model and do not address the interaction of data flow between multiple models. The hierarchical system deals with a collection of models that have the ability to interact, communicate and pass information with each other.

This system can be adapted to conduct virtually any type of analysis, since this philosophy is not bound to any specific kind of analysis. It provides a framework to manage different models and its results and more importantly, the interaction between the different models. Thus, it is ideal for many types of finite element analyses like global/local analysis and those that involve multiple scales and fields.

Information Management

A major task for the hierarchical system is information management. Since the system would be managing a hierarchy of models, it would have to store the information for all models. At the same time, the size of models can vary from very simple meshes with just a few elements to very large complex meshes that have over half a million degrees of freedom. The system should be designed such that it does not impose a limit on the size of the hierarchy or the models as long as there is enough memory on the

computer. In addition to the generic data like mesh information and other input to the analysis model, the system should also be able to store post-processing data that could very well take a large amount of memory space depending on the kind of output desired. Thus, the system should have a robust and efficient data management strategy that can handle numerous as well as large datasets.

Most operating systems like Windows have automated features that take care of memory management of the program. This is achieved by using a feature called virtual memory. Usually, the amount of Random Access Memory (RAM) on a computer is not enough to run all of the programs that most users expect to run at once. A typical high-end desktop computer has about 512MB of RAM which is not a whole lot when you consider the memory required by the operating system itself, let alone the memory required by a large analysis model which contains the mesh information, the system of equations to solve and the solver itself. Using virtual memory, the operating system creates the effect that the computer has more memory than the actually installed RAM. This is achieved by using the physical hard disk space to store information from the RAM that has not been used recently. This frees up space on the RAM to be used by a new application or those that need more memory. The OS does all this work behind the scenes so that the end-user does not feel the crunch for memory space. Having said this, it is still important to design the system so that it manages it information efficiently because heavy use of the virtual memory makes the system very sluggish. So, it is advisable to write information that is not required all the time to the hard disk memory. Information like post-processing data which is not required in the memory all the time can outputted to the hard disk and can be read back in when required. Another way to make efficient use of memory space is by using binary files. Binary files use lesser physical memory for storing the same amount of data. Also, input and output is much faster using binary data. The disadvantage of using binary files is that it cannot be viewed using a simple text editor like notepad because of the way the information is stored in the file.

Interfacing with External Software

Another important feature is the ability to interface with other software. There is a lot of legacy software on the market and each might have its own forte. It would be unwise to assume that all kinds of analysis work should be conducted within this environment itself. It would be advantageous to make use of special features that other software possesses. This is possible by exporting data into another format that can be understood by other commercial FEA software using some interface function. Presently, it is possible to export mesh data and other model information to the commercial software FEMAP by writing FEMAP Neutral files. In this way, the advanced mesh generation, analysis and post-processing features of the FEMAP software can be utilized. This capability can be extended to export analysis model information to other commercial FEA software like NASTRAN, ANSYS or software that use the EXODUS [49] Database Format. When exporting an analysis model to external software that is typically not hierarchically defined, a model has to be set up by combining the components in such a way that the external software is able to 'understand'. Similarly, when external software is used to conduct the analysis, the results are passed back into the hierarchical system through the interface, which maps the results back into the hierarchical format. This way the user is not forced to use only the hierarchical system and can make use of other features that are available in commercial software. Figure 4 gives a block diagram illustrating how the hierarchical system interfaces with the external software. ALPHA-HS denotes the solver resident within the Hierarchical System.

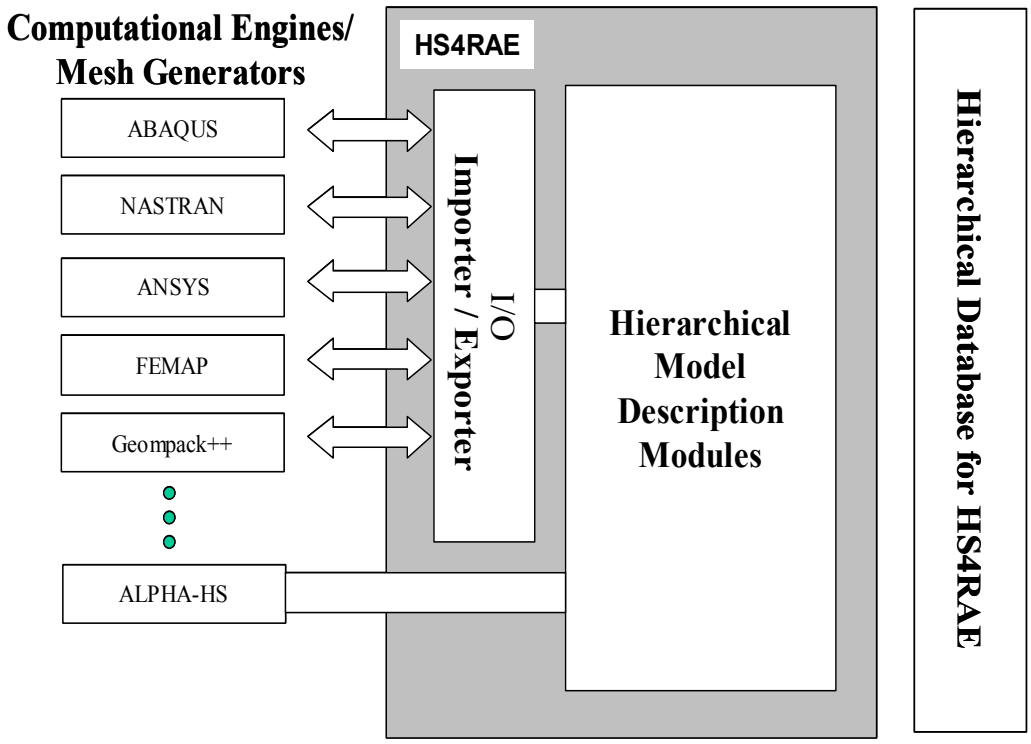

Figure 4: Interface to external software

Implementation

This section describes the implementation of the hierarchical system including the different classes and functions developed. It also discusses the different concepts involved in organizing the models in a hierarchical fashion.

The hierarchical analysis environment consists of a number of tools and modules that interact with each other. These are the key components of the system:

- Hierarchical definition module/library: this library contains the different classes and functions that implement the inheritance and storage of hierarchical data. It is the core of the hierarchical system that contains the model information.

- Visualization tool: this component is used to view the hierarchical models and for visualizing the results of the analysis and the post-processing data.
- Scripting Language: It is one of the ways the user can interact with the system using the command line or in batch mode. It is used to describe the relationships between different models and to control the analyses. The user can issue commands to the system using the scripting language. It can also be used to maintain persistence of data.
- Graphical User Interface (GUI): this is another way for the user to use the hierarchical system. Using the GUI, the user can interact with the hierarchical system in real-time with the help of the keyboard and mouse.
- Import/Export functions: these functions allow the transfer of data between the hierarchical system and other external software like mesh generators and FEA programs like FEMAP, ABAQUS etc.
- Solver: This is a set of classes and functions that are used to numerically solve the set of equations defined by the finite element model. The solver is integrated with the ModelHandler class, which is present in the hierarchical definition module.

It must be noted that the different tools in the hierarchical environment were not developed from scratch by the author. Such an undertaking would take quite a bit of manpower and time. Instead, existing in-house codes were modified to develop the required tools. The hierarchical framework, basic class design including the implementation of the global/local methodology was done by another individual involved in the research project. The author was involved in extending the system to analyze composites with non-uniform distribution of fibers. The author also developed a GUI for the system by modifying existing code for the visualization tool. Existing code was made use of to implement an ActiveX control version of the visualization tool, which was then used in a GUI developed using Visual Basic.

Software Platform

All code development for this project was done on the Windows platform. This kind of environment requires a number of tools to successfully analyze a problem. Each of the tools developed will be discussed in the following sections. The C++ programming language was mainly used to code most of the hierarchical environment. The object oriented-ness of the language was made use of extensively to implement the inheritance and hierarchical nature of the models. In addition to Visual C++, Visual Basic was also used initially to develop a prototype Graphical User Interface (GUI) for the environment. The collection of classes and functions that formed the core of the hierarchical environment were packaged in the form of a Dynamically Linked Library (DLL). All the classes and functions were written in C++ and compiled using Microsoft Visual Studio 6.0. The advantage of using a DLL to store the core of the environment is that it can be easily shared among different technologies and different versions of the GUI. In addition to the Visual Basic GUI, a command line version, an ActiveX control version and a Visual C++ version of the GUI were developed. These will be discussed in the following sections.

Rather than building the whole environment from scratch, an existing conventional in-house finite element code called Alpha was modified to add the hierarchical character. Alpha was built to run on both UNIX and Windows platforms. Therefore, although the current work was done on a windows platform, with some effort the hierarchical environment should be portable to the UNIX platform too.

Hierarchical Framework

As mentioned earlier, the object-oriented nature of C++ was instrumental in implementing the hierarchical framework. A hierarchy consists of a group of entities linked to each other in a particular fashion. In this case, the entity is a model, which is defined by a model class. By using a dynamic list of pointers to other models, it is possible to create a link from one model to another. The Model class will be discussed in

detail in later sections. All entities in a hierarchy have some common properties, which gives them the right to be in the hierarchy. In the hierarchy that we deal with, all the entities are models that have some common properties. For example, all models have data that defines its geometry. All models have mesh information, which contains elements and nodes. But, depending on the type of problem being analyzed, it is also possible to have specific types of models that have special properties in addition to the common properties like those mentioned above. For example, when analyzing the non-uniform distribution of fibers in a composite, special information like the position of the fibers in the matrix is needed. At the same time, it does not make sense to include the fiber information in the model class that defines the hierarchy since all the models in the hierarchy may not need the fiber information. This problem is solved using the abstraction mechanism of C++. By abstraction, it is possible to define a basic model class that has the data and functionality required of all types of models and then derive specific model classes from the basic class as required. Then by defining the hierarchy in terms of a list of pointers to basic model classes, we are able to achieve both the hierarchical character of the models as well as the specific properties of each individual model. Each model in the hierarchy has a pointer to it parent model and a list of pointers to the models that are derived from it. In this way, a hierarchy of models is set up.

The hierarchical framework is designed in C++, which is a high-level object oriented programming language and allows efficient management of memory resources. The models are stored in the memory as objects that are dynamically created and managed using pointers. As a result, the only restrictions imposed on the system with respect to the number of model and their size are the memory on the computer and the limits set by the programming language.

As explained in an earlier section, the sharing of information between the models in the hierarchy is a very important aspect of this philosophy. Each model in the hierarchy should be able to ‘interrogate’ another model for data that it needs. This kind of communication between models requires the ability to traverse the inheritance tree.

There are two approaches to traverse a tree – one is to start with a model in the hierarchy and work your way down till you reach the lowest model in the hierarchy. Another way is to start with a model and traverse up the tree till you reach the top model in the hierarchy. You choose one way or the other depending on what you want to achieve. For example, if the requirement were to deactivate all the models in the model path, the bottom-up approach would be used. On the other hand, if the requirement were to set the reference temperature for all the models in the hierarchy, the top-down approach would be used. Figure 5 shows the difference between the two approaches. The bottom-up approach on the left starts with the model H1RC through H1C, H1, H2 to the topmost model LC. The top-down approach on the right issues the command to all the models in the hierarchy. Complex recursive functions were written that could issue a command to a particular model or to a group of models using either approach. Functions were also written to issue a command to a specific model in the hierarchy.

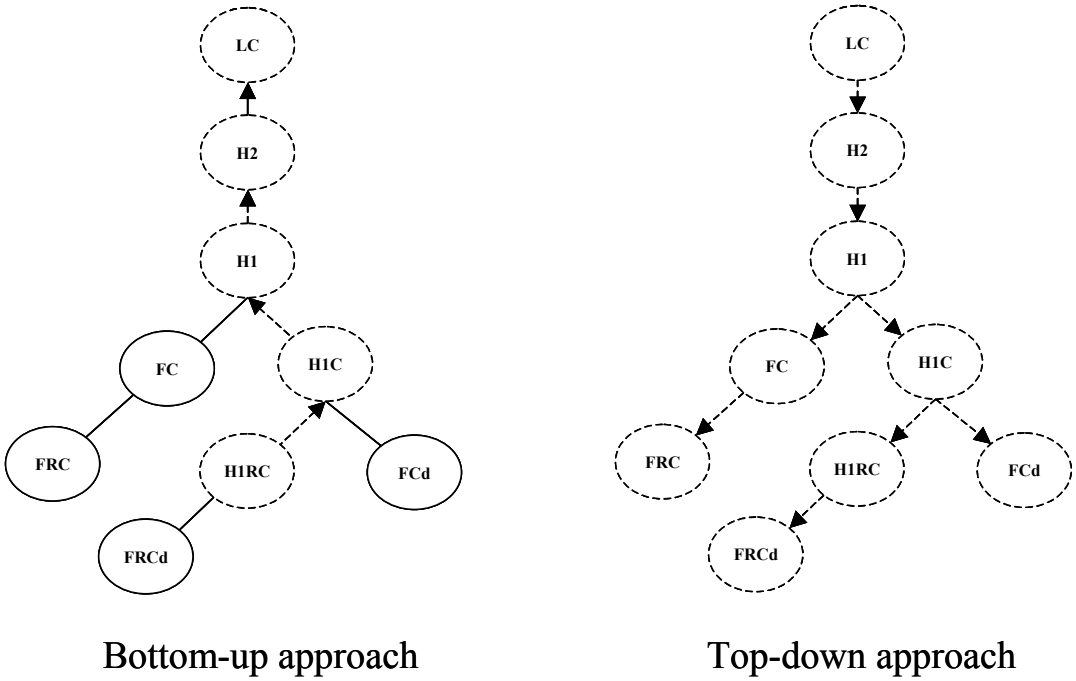

Figure 5: Inheritance tree traversal (Bottom-up and Top-down approach)

Model Class

The model class is the one of the most important classes developed for the hierarchical system. It is the used to define a model in the hierarchy. It has the capability to issue commands to other models in the hierarchy using the hierarchical framework. The model is defined in terms of all the components in its model path. The basic model class stores the component information that defines the model that it represents in the hierarchy. As explained earlier, the component is the information that differentiates the model with the model just higher up in the hierarchy.

The basic model class handles only inheritance of geometrical information. This means that the model can inherit geometrical information from the components in its model path and generate the mesh for its own model. This mesh that is generated from the sum effect of all the components in the model path is called the 'collective mesh'. This mesh information is also stored in the model class and is very useful for visualization purposes to view the geometry of a model in the hierarchy. The collective mesh generation follows a z-order component mesh association. This means that a component lower in the hierarchy replaces any part of a component higher up in the hierarchy that occupies the same region in 3D space. Figure 6 illustrates the inheritance of geometry for a simple case where there are only two models in the hierarchy. for simple two dimensional meshes, it is easy to consider the overlaying of component B on top of component A and it is easy to see that part of component A is overlapped by component B. This region in component A will be deactivated and replaced by component B. Figure 7 illustrates a more complex case that involves three components. The hole replaces a region on the panel but the patch replaces part of the hole as well as the panel and generates the collective mesh shown in the figure. Thus, a component can replace elements over a number of components in its model path and not just its immediate parent model alone. The procedure to generate a collective mesh gets harder when dealing with complicated geometries. Currently the system can automatically generate collective meshes for simple two-dimensional geometries. For some complex

geometry it is possible to ‘help’ the system by telling it what regions in the components to deactivate. This is accomplished by specifying a ‘region definition’ file when defining the model. This file contains a series of commands that defines a region. This tells the model to deactivate all elements in the components that fall within this region.

In keeping with the main objective of this project, that is to conduct rapid analysis, it is desired to automate as many functions as possible. One of the important steps in combining components to generate a complete analysis model is boundary detecting and matching. The model class carries out this function. The model has the ability to detect the boundaries in its collective mesh and sort it and match it with each other so that the different components can be combined to form a single seamless model. The nodes that make the boundary of a single mesh are detected by looking for the element faces that are not shared by more than one element. But, just as in the generation of the collective mesh, the process gets more complicated as each model in the inheritance tree is actually the sum effect of all the components higher up in the hierarchy. Figure 8 shows the stacked mesh view of the configuration described in Figure 7. The solid line on the topmost stack shows the boundary of the model while the dotted line shows the matched boundary segments that need to be ‘joined’ in order for the group of components to behave as a single model.

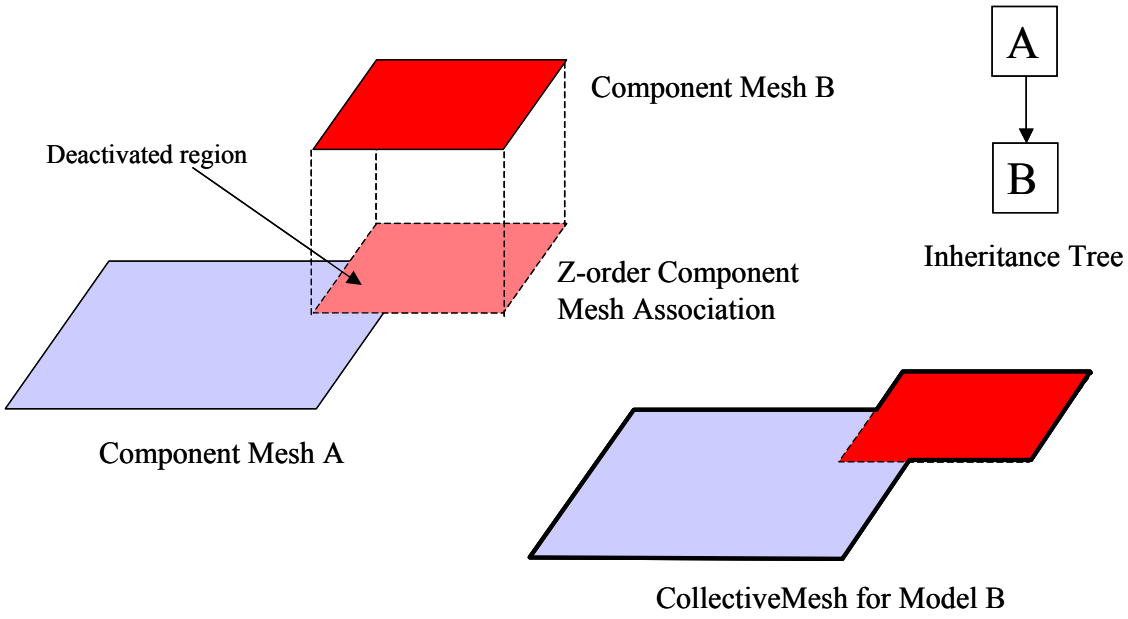

Figure 6: Inheritance of geometry (2 components)

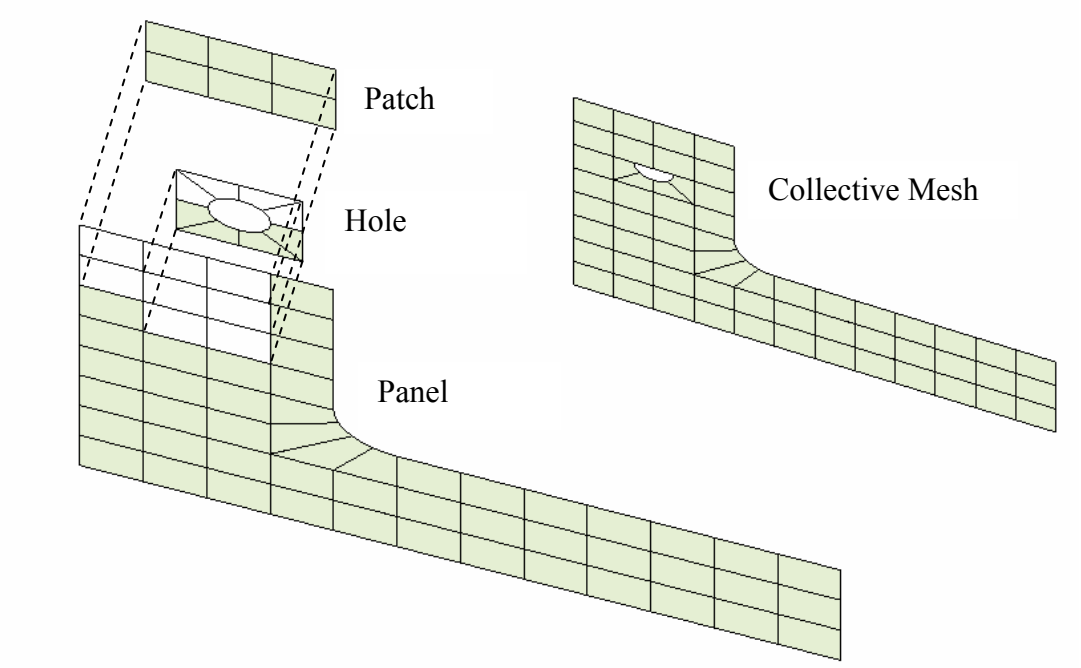

Figure 7: Inheritance of geometry -collective mesh (3 components)

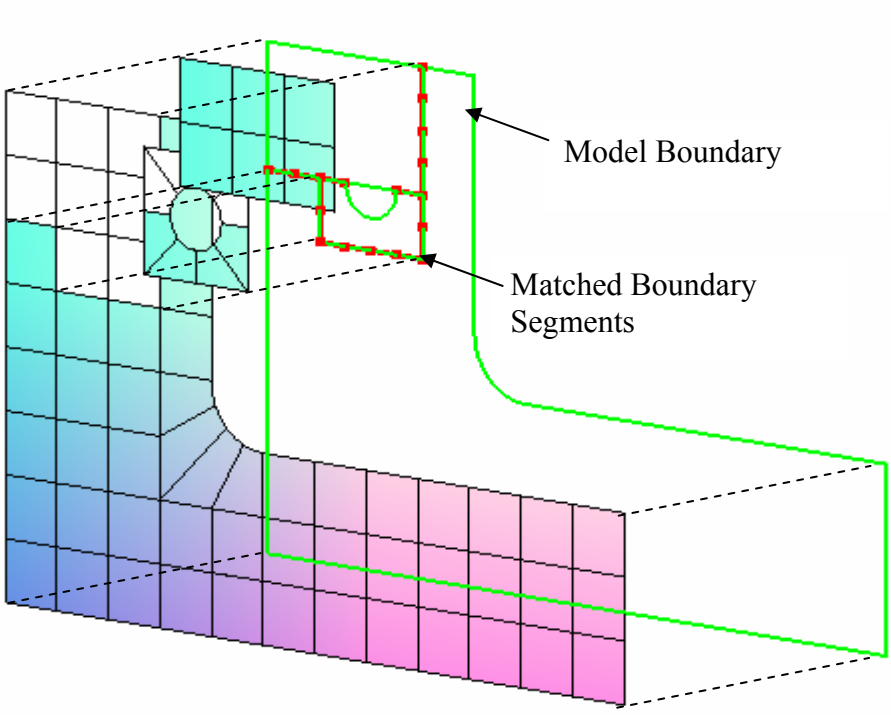

Figure 8: Boundary detection and matching

The basic model class by itself has no finite element analysis capability. This function is handled by a ModelHandler object, which is 'attached' to the model object when it is needed. The ModelHandler handles the inheritance of all non-geometric information like load conditions, boundary conditions and material properties.

The user communicates to the model through the Graphical User Interface (GUI) or the scripting language. There are two main types of scripts – the Model script and the ModelHandler script. The Model class processes the commands in the Model script while the ModelHandler class processes the commands in the ModelHandler script. The scripting language is dealt with in detail in a later section.

ModelHandler Class

The ModelHandler class is the executive for the finite element analysis, results retrieval and visualization. The model class has a pointer to a ModelHandler object.

When an analysis model needs to be generated, a ModelHandler object is instantiated and linked to the model in the hierarchy. The ModelHandler object holds the remaining analysis information such as the load conditions, boundary conditions and the material properties. The equation solver is also integrated with the ModelHandler. The ModelHandler reads in the load conditions, boundary conditions and the material properties from the ModelHandler script or is fed into the system through the GUI.

The ModelHandler has the ability to integrate the components in the model path to generate a complete analysis model. A list of the components in the model path is generated along with its properties such as the type of elements, the material properties and the number of degrees of freedom. The number of degrees of freedom (dof) for the complete model is calculated and a dof map is created that maps the dof of a particular node in a component to the corresponding dof in the list of dofs for the model. It must be noted that due to the inheritance of geometry, some of the nodes and elements in the components might be deactivated. And so this has to be accounted for when generating the list of dofs for the model.

In the current implementation, the components are joined by imposing Multi-Point Constraints (MPCs) on the degrees of freedoms of the nodes on the boundary interface of the components. This means that the degrees of freedoms of a node on a boundary segment is expressed in terms of the degrees of freedoms on the boundary segment that it matches with. Or, in MPC terminology, it is said that a node is ‘slaved’ to one or more nodes. The nodes that it is ‘slaved’ to are called ‘masters’. The constraints can be considered as ‘digital glue’ that is applied to join the components. By using multi-point constraints, we are trying to impose continuity of displacement along the boundary interface. But, in certain cases, MPCs do not work well, in that it does not impose complete continuity along the boundary interface and this will be explained later on in this section. MPCs are used to slave the more refined boundary to the less refined boundary. Initial work required for joining components, i.e. detecting the boundaries and matching them up with corresponding boundaries is already accomplished by the model

object. Figure 9 explains the process of joining two components. Component B on the right has to be joined to Component A on the left along the common boundary. For ease of understanding, the two boundary segments on the interface has been shown separately as two different lines and all the elements in the components are 4-node linear elements. The boundary segment on the right has more nodes than the segment on the left. Therefore, the nodes on the right will be slaved to the nodes on the left. For the simple case shown here, there is only one element on the left boundary, whereas there are four elements on right. The global coordinates of a node on the right should fall on the boundary segment on the left since both boundary segments coincide. For the two-dimensional case shown in Figure 9, it also follows that the node on the right would either coincide exactly on a node or fall on the side of an element on the left boundary segment. For the first case, the degrees of freedom for both the coincident node will be the same, which is obvious since they represent the same point in the global coordinates. For the second case, the displacement at that point has to be expressed in terms of the nodes on that element edge. This is done by getting the local coordinates of the point on the boundary segment and then interpolating the displacements at that point using the shape functions for that element. It will turn out that since the point is on the element's side, only the nodes on that side will contribute to the displacements at that point, since the shape functions for the other points will vanish. A generic equation for imposing these MPCs can be expressed as:

$$a(\mathbf{x}) = \sum N_i(\boldsymbol{\xi}) a_i$$

where a(x) is the degree of freedom of the node at point x on the left boundary segment, and N_i are the shape functions and a_i are the degrees of freedom at the nodes of the element on the left. Also, ξ is the coordinate of the point x in the local coordinate system of the element on the left.

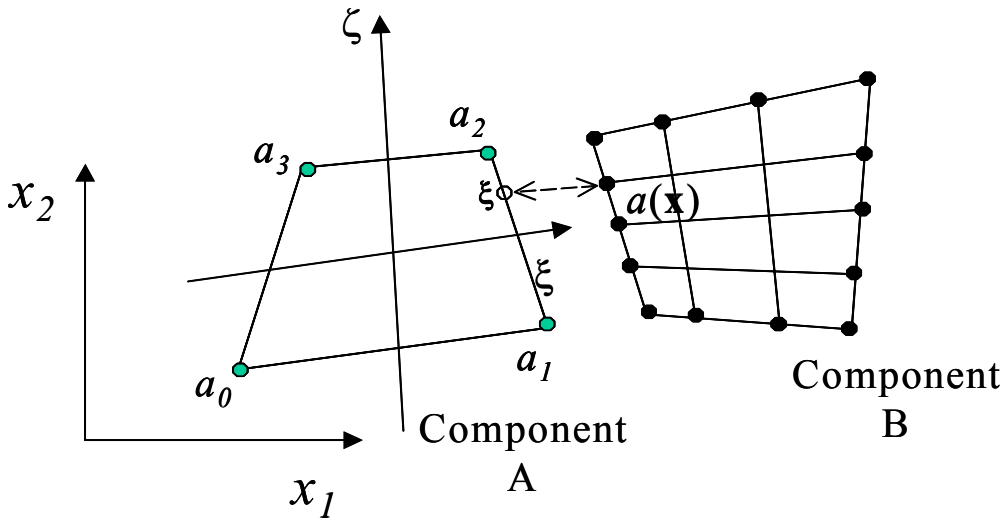

Figure 9: Joining components using MPCs

There could be cases where the two boundaries do not actually coincide but are very close to each other. This might happen due to rounding errors when reading the mesh information. In such cases, a tolerance is allowed so that two points can be considered as coincident points.

While multi-point constraints are used to try to impose continuity of displacement all along the boundary interface, in certain cases it is not completely achieved. Continuity of displacement at the nodes where the MPC are applied is assured but continuity is not assured all along the boundary. The MPCs work well for cases where the slave nodes coincide exactly with the master nodes and also for cases where all the slave nodes belonging to a single element are slaved to the nodes belonging to a single element on the master side of the boundary interface. A simple example is used to illustrate this problem (see Figure 10). In the case on the left, two elements on the right are ‘joined’ to the two elements on the right using multi-point constraints. In the case on the left, the two elements on the right are joined to a single element on the left.

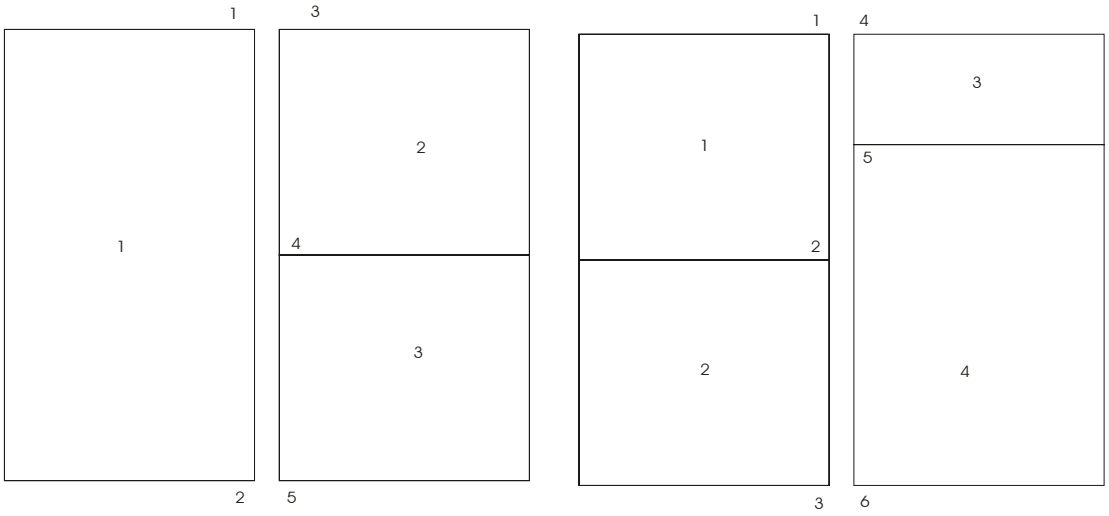

Figure 10: Example illustrating bad MPCs

For the case on the left, both the boundary nodes of element 2 are slaved to nodes of the element 1. Similarly both the boundary nodes of element 3 are slaved to the nodes of element 1. When the components in the left case are joined, the degrees of freedom for nodes 3 and 5 on the right boundary interface are set equal to the corresponding degrees of freedom for nodes 1 and 2 on the left boundary interface. The degrees of freedom for node 4 are expressed in terms of the dofs of node 1 and 2. For the case on the right, node 4 coincides with node 1 and node 6 coincides with node 3 while the degrees of freedom of node 5 is expressed in terms of node 1 and node 2. Therefore, the displacement field of element 4 is expressed in terms of the displacement field of two different elements (element 1 and 2). This causes an incompatibility in the displacement along the boundary.

Both the cases were analyzed for a condition of uniaxial stress in the x direction. While the case on the left gave the expected uniform stress state, the one on the right did not. The finite element method tries to arrive at a solution that brings the total potential energy of the system to a minimum. When a continuity of the displacement is not imposed on the boundary interface, the method finds a configuration where the total potential energy is minimum, which in this case is when the boundary interface opens

up. This is verified for the cases in Figure 10 and it turns out the case with the bad MPCs has a lower total potential energy than the one on the left. Figure 11 shows the deformed mesh for the case with the bad MPCs where the boundary interface opens up like a crack. Figure 12 shows a plot of the x displacement along the boundary interface. The dotted curve shows the displacement along the boundary segment on the right while the solid line curve shows the displacement along the boundary segment on the left. It can be seen that there is continuity of displacement along the interface from node 4 to node 5 but the displacement between node 5 and node 6 are not same on both sides of the boundary interface.

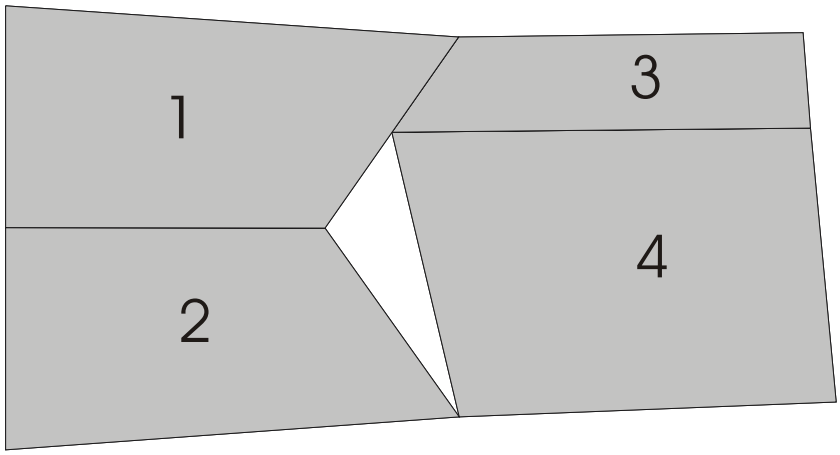

Figure 11: Deformed mesh for case with bad MPCs

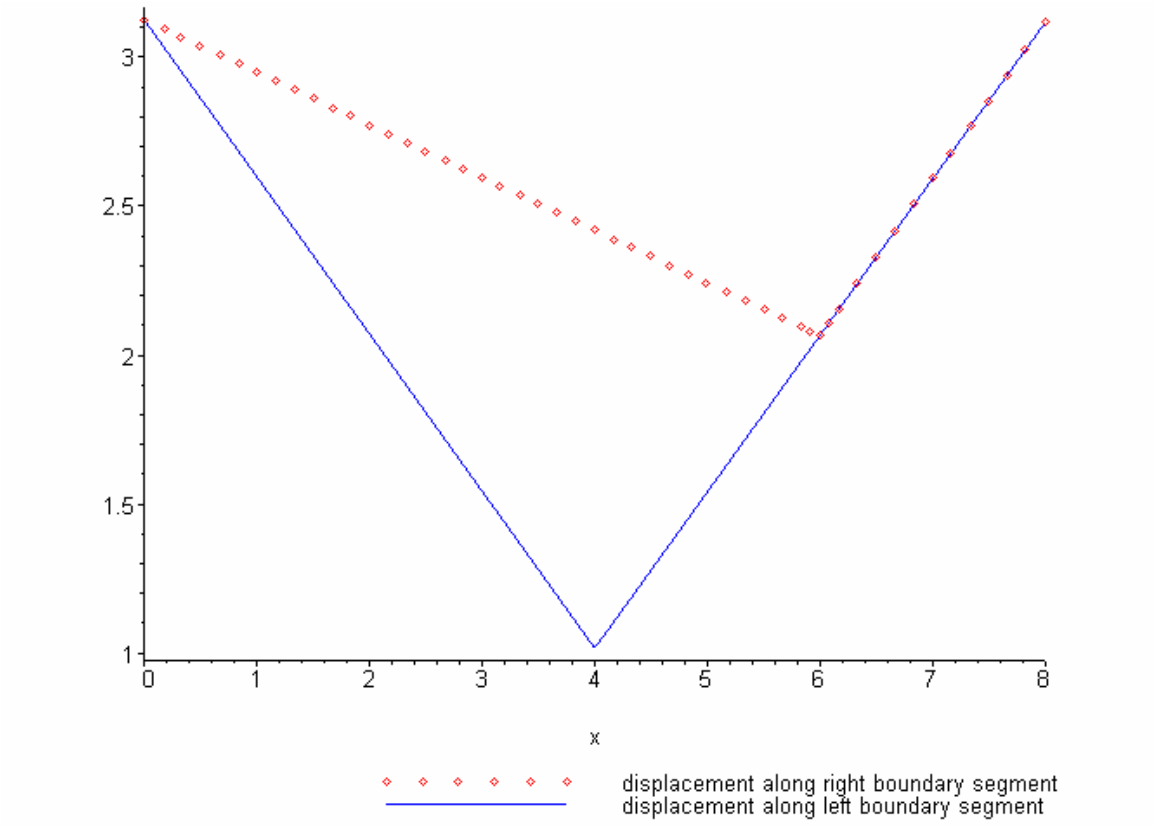

Figure 12: Plot of x displacement along boundary interface

The error in the solution caused by the problem with bad MPCs can be considered negligible if the mesh is large enough and the refinement is of the component is good. Although the results close to the boundary interface will be wrong, the perturbations will die out when you go further away from the interface. An alternative is to use a method to approximately satisfy continuity by using something like a least squares fit but this has not been addressed in this work. Another method is to use interface elements (developed by Ransom, Aminpour et al [50-53]), which uses a variational method to approximately satisfy compatibility at the interface.

The ModelHandler 'joins' the components using the multi-point constraints. Figure 13 illustrates the use of this 'digital glue' to join three components to form a single analysis model. It can be seen from the figure that a node in a model that is

situated bottom most in the inheritance tree can be constrained to a node in a model multiple ‘generations’ up the hierarchy and not necessarily to the immediate parent model. This is implemented by making use of the recursive functions in the hierarchical framework that allow tree traversal.

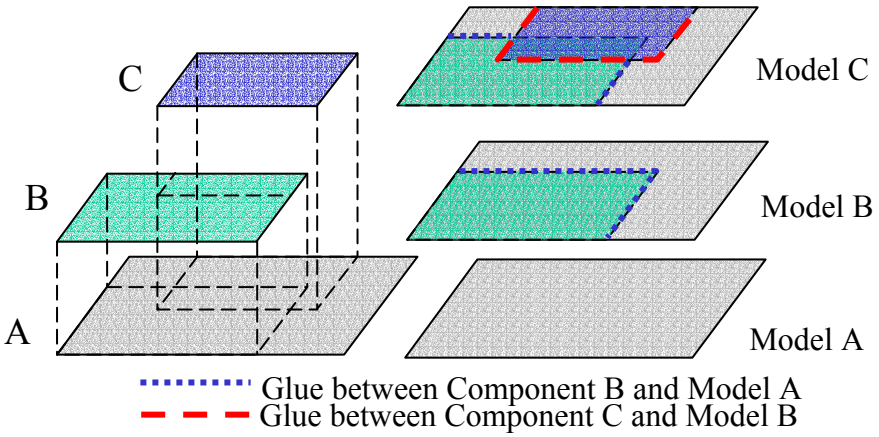

Figure 13: Joining components using ‘digital glue’

When a model is analyzed, certain data in the model path must be specialized for that particular model. An example of such a type of data in the current implementation is the degree of freedom (dof) map mentioned earlier, which is required to map the results back to the components. This becomes a problem when a component can be a part of two models, which can easily be the case as shown in Figure 14. In this case, component B is used both by model C and model D. Therefore, the need arises to set only one model as ‘active’ at any point in time. Thus, each time a model is made active, the dof maps stored in its components have to be refreshed, since the existing dof maps in those components could be the dof map for the previously active model. In this way, the information in a component is dynamically updated by the currently active model and the component has a dynamic buffer storing the dof map for the active model. At present, this is not a major limitation. In fact, it saves on memory and book-keeping of this data by retaining the data in the component and making use of the hierarchy. The need might arise in the future for more than one model to simultaneously require their

specialized data, which currently resides in the component. In that case, we would need to maintain the data with the model rather than store it in the component.

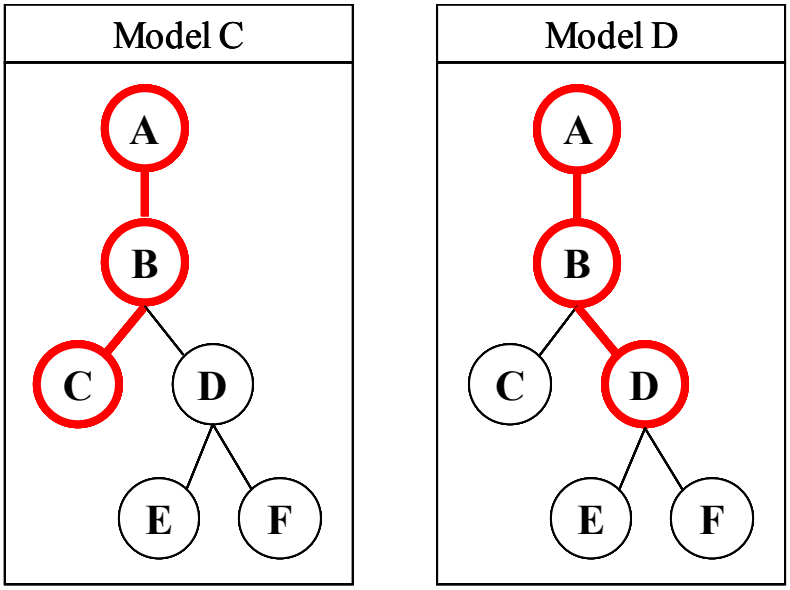

Figure 14: Sharing of components by two different models

Although the equation solver is incorporated in the ModelHandler class, it must be noted that it is just a conventional solver and does not solve the analysis problem hierarchically. Therefore, the ModelHandler class constructs the complete set of equations as a conventional model, asks the solver to solve it and then use the dof map to map the results back to the components in the hierarchy. There is a potential for a hierarchical solver but this idea has not been pursued as yet.

Figure 15 gives an idea about the organization of the data in the hierarchical models. The model class is the central data structure that defines each model in the hierarchy. It can be seen that while the geometrical information such as mesh data resides in the Model class, the analysis related information like material properties etc are handled by the ModelHandler class, which in turn is part of the Model class. The ModelHandler manages the storage of and access to the results. The handler can be created/modified using the model handler scripting language or the GUI.

Just as the abstraction mechanism is used to derive more specific models from the basic Model class, the same can be done for the ModelHandler class. A model can have more than one handlers associated with it depending on the type of analysis. Analysis-specific handlers can be derived from the base ModelHandler class depending on the requirement; for example, different handlers would be needed for linear and non-linear analyses. The ability to access the data in any model in the hierarchy and the robust communication between models open the market to conduct a wide variety of analyses within the same framework. Handlers could be designed that make use of this functionality to conduct parametric analyses. Specific ModelHandlers can be designed that can handle the inheritance of load conditions and other information like material properties and boundary conditions so that the same information need not be fed in each time a new model is derived.

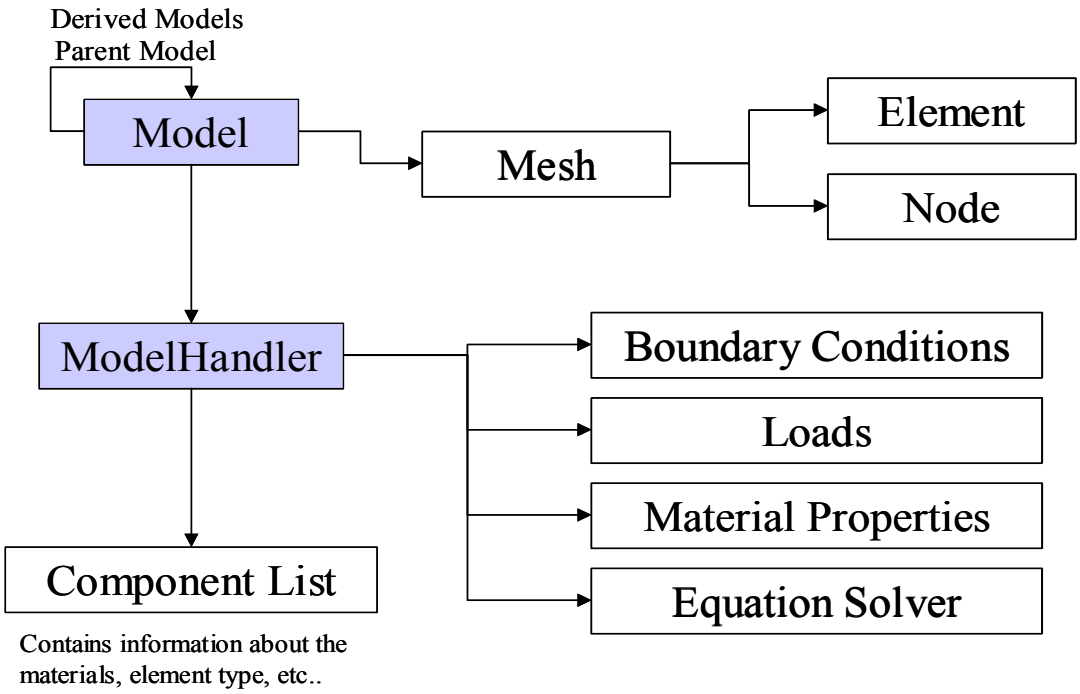

Figure 15: Organization of data in the hierarchy

A global/local analysis methodology that is described in the work by Ransom, Knight et al [21] has a global model which is analyzed and the displacement field

obtained from this analysis is used to generate the boundary conditions for the local model. The local model, which is more refined than the global model is then analyzed to get a more detailed response for the region of interest. This kind of methodology is ideal for the hierarchical framework because of the ability to access data residing in another model in the hierarchy. The ModelHandler that is implemented in this environment can be used to conduct global/local analysis as well, albeit in a different fashion. A coarse model can be analyzed initially to get the global response to the loading conditions. In order to get a closer look at what happens in a specific part of the global mesh, all that has to be done is derive a new model from the global mesh using a more refined mesh of the region of interest. The ModelHandler then combines the global model and the local model into one analysis model and provide a detailed response for the region defined by the local component. This kind of methodology can also be used to make modifications to an existing model by just adding more components; for example adding a fillet to alleviate the stress concentration at a corner in a structural component. A similar example will be discussed in detail in the following section.

Other Classes

This section discusses some of the other important classes that were developed to implement the hierarchical system. This list does not include each and every class that is used in the system. The first four classes - BoundaryFaceList, MatchedBoundarySegment, MatchedBoundarySegmentList and MPCGlueList are used to detect and sort boundaries and finally join the components to generate a complete analysis model.

BoundaryFaceList

This class stores the list of faces that make up the boundary. For a 2D mesh, the face would a line connecting two consecutive nodes in an element. For a 3D mesh, the boundary would be a surface and the face would be a face of a 3D element. When a hierarchical model is defined, a ‘findBoundary’ function in the model class is called that

detects and sorts the boundaries in the component mesh and the collective mesh. Figure 16 shows the hierarchical model FC along with its inheritance tree. A BoundaryFaceList object will be used to store the model boundary shown in the figure.

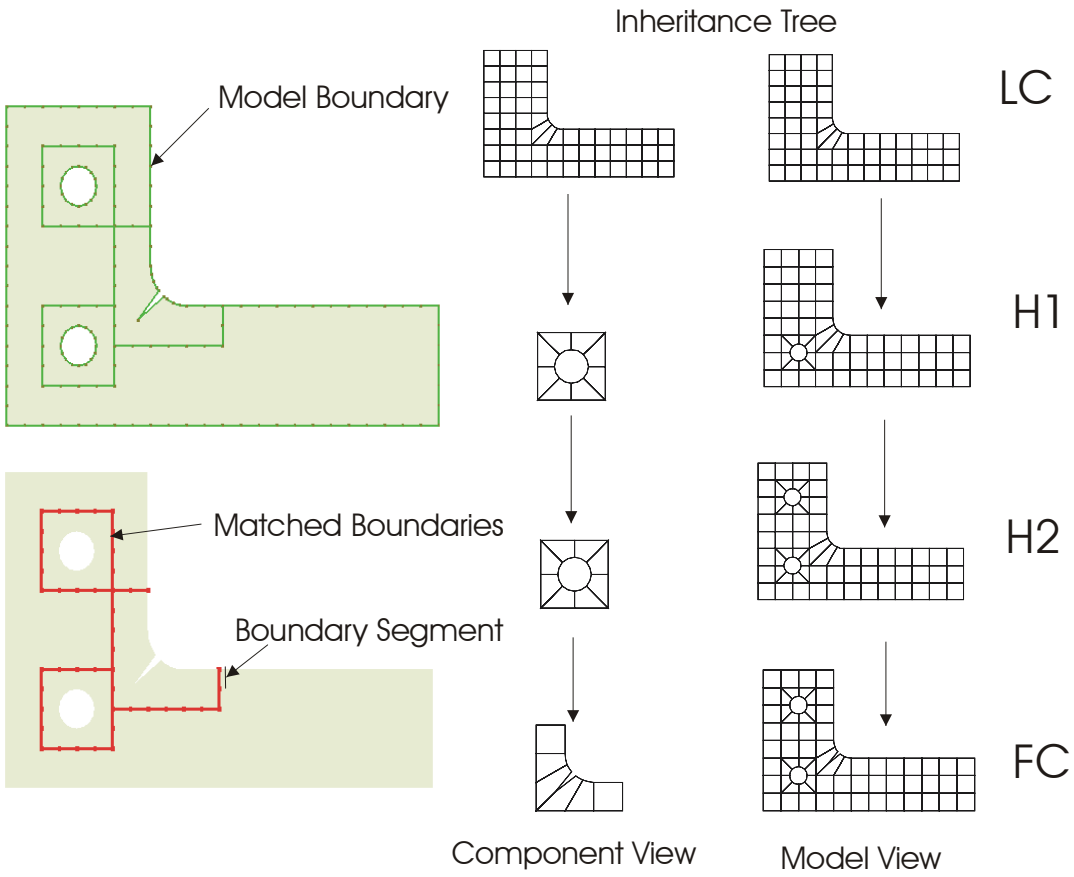

Figure 16: Visualization of boundary objects

MatchedBoundarySegment

This class is used to store information about how the boundary of a model's component matches with the rest of the components in the model path. In many cases a component might not share its whole boundary with another component. It might share part of it with one component and part with another, or it might not share its boundary at all; for example if it is on the periphery of the model. In Figure 16, the model being considered is model FC. The boundary of component FC is to be matched to parts of the

boundaries of component LC and component H1. Therefore, before the boundaries can be matched, they have to be divided into segments so that segments from different components can be matched. To make it even simpler, the matching boundaries are divided into segments such that the end points of a segment coincide with the corresponding segment in the matching boundary on the interface. The information about a pair of such matching boundary segments is stored in the MatchedBoundarySegment class.

MatchedBoundarySegmentList

As each boundary segment of the main model's component is matched to its corresponding boundary segment of another component, a MatchedBoundarySegment object is created to store this information. These MatchedBoundarySegment objects are collected and stored in the MatchedBoundarySegmentList object. This list of matched boundary segments is then used later by the ModelHandler to apply the multi-point constraints and join them up.

MPCGlueList

This class is used by the modelhandler while setting up the analysis model to 'join' the components by imposing the multi point constraints. The MPCGlueList object contains the list of nodes that need to be slaved and the nodes that it needs to be slaved to. This list is created from the MatchedBoundarySegmentList objects in each of the components in the model path. Hence, all the components are joined to the corresponding components higher up in its model path. The nodes in the replaced regions of components don't get slaved because the main model deactivates those nodes before this process is initiated. Once the MPCGlueList object is ready, the coefficients of the multi-point constraint equations are calculated and the MPCs are applied.

ComponentInfo

This class is used by the ModelHandler to store the information about the active components that are used to generate the current analysis model. It contains information about the element type, number of degrees of freedom per node and the material properties for the component. This information is then used to calculate the total number of degrees of freedom for the model, generate the dof map so that the results can be mapped back to the components, calculate the stiffness matrix for the active elements in these components and other steps like post-processing.

CompoundMesh

This class is used to store the collective mesh of the model. The collective mesh as mentioned earlier is used to describe the geometry of the derived model, which is built from the collection of components in its model path. Once the collective mesh is generated and stored in the model object, the model does not have to traverse the inheritance tree and go through the process of generating the complete geometry from the components in its model path each time the model needs to access its geometry. For example, when the model wants to detect its boundary, it can do so by reading the information in the collective mesh rather than generating its geometry from the components in its model path.

Group/ GroupItem

This is a generic storage data object that is used to store information like an element list or node list. It has a data member that is used to describe some attribute of the data it is storing. For example, it is used in the ComponentInfo class to describe the element type and material property of the elements in the component.

User Interface

This section describes the different tools that let the user communicate with the environment. There are basically three components – the scripting language, the visualization tool to view the plots and results such as stress contours, and the GUI that lets the user control the visualize the hierarchy of models.

Scripting Language

The scripting language is an important feature that forms the link between the hierarchical system and the user. The user inputs data and issues commands to the system using the scripting language. The scripting language is the only mode of input for the command line version of the hierarchical system. Using scripts, it is possible for the user to completely define a hierarchy of models and control their analysis, without the actual need for a Graphical User Interface (GUI). Like several commercial applications, the GUI can also be designed in such a way that the user has the option of entering script commands directly onto a command window in the GUI, rather than use the menu-driven options and mouse. This provides the seasoned user with the flexibility to use the scripting language in the GUI to control the hierarchical models rather than deal with dialog boxes etc.

There are basically two different scripts files that are needed to completely define a hierarchical model and run an analysis. The first type of script file is called the model script. The model script file is used to define the hierarchy and to set up the relations between the different models. The model script file has a .rae extension to differentiate it from other files. This script usually has commands like ‘createModel’ that create models in the hierarchy. The model script can also be used to output current organization and information related to the models and the hierarchy to an output file.

While the model script file defines models and issues commands to analyze the hierarchical models, it does not contain the commands that control the analysis of each model. This is just a decision of choice and based on the readability of the scripts and the

design of the hierarchical models. The Model class processes the commands in the model script file through a ‘processCommands’ function. The model script file issues the command to instantiate a ModelHandler object and associates it with the model that needs to be analyzed. The ModelHandler then controls the analysis by receiving input from the second type of script file called the ModelHandler script file. The command parser and other utility functions that were used to implement the script language were imported with minor modification from an existing in-house FEA code.

The ModelHandler script file has an .hdl file extension to identify what kind of script file it is. Just like the Model class, the ModelHandler class processes the commands in the ModelHandler script file through its ‘processCommands’ function. The ModelHandler script has the commands that tells the ModelHandler all that it needs and how to conduct the analysis. It contains information like material properties, boundary conditions, load conditions etc. It tells the ModelHandler what type of solver to use and also what kind of post processing to do.

As more specific Model classes are derived from the basic Model class, special commands can be issued in the model script file that the specific model class can process. for example, a special micromechanics model was derived that did not need the usual mesh information file but creates its own mesh information on the fly based on some parameters. These special commands could be present in the same model Script file and could be processed by the special model object. The same idea goes with the ModelHandler class. As mentioned earlier, specific ModelHandler classes can be derived from the basic ModelHandler class that can be used for specific analyses. The model class just needs to instantiate the specific ModelHandler object and this new object can conduct the analysis based on the commands present in the ModelHandler script that was written for that analysis.

Like almost all software, the need arises to save your work to the disk for later reference or for resuming the work at a certain point in time. Data persistence is achieved by using the scripting language. The current organization of the hierarchical

models can be saved to the disk using the scripts and the scripts can be read by the system later to load the models back into the memory. Hierarchical models that have been already analyzed can be viewed at a later point in time by reloading the models back in to the hierarchical system. The scripts can also be used to store the state of analysis so that one could revisit/resume analysis later. If the same models are being loaded with out any modification, then there is not need to run the analysis again. In such cases the system can load the results directly from the files that were generated the last time the analysis was conducted. The output files for different analysis models are stored systematically by making use of the computer's file system, which has a tree structure. Therefore, it is easy for the hierarchical system to locate the results for a specific analysis. This way is it easy to transfer complete hierarchical models from one location to another including the results by just copying a folder.

Typical Model and ModelHandler script file formats will be discussed in the following sections that will go through a couple of applications of this hierarchical environment. A complete specification of the scripting language is given in **APPENDIX A**

Visualization

The visualization tool called Plotter2002 (or simply Plotter) is a modification of an existing in-house tool called Plot2000. It is written in C++ and uses the Microsoft Foundation Classes. The Plotter has two versions – a stand-alone version and an ActiveX control version. Both these versions are built upon the same underlying functional code that is in the form of a Dynamically Linked Library (DLL) called PlotInterface.dll. The plotter has three different ways to display the output – in GIF format, PostScript format or render it directly onto the monitor screen using OpenGL.

The program has been designed in such a manner that it can be easily adapted or enhanced to display different types of data without the need for major code re-writing. This is done by using an interface class called the PlotInterface class that provides an

interface between the data to be displayed and the output mode. Figure 17 gives a block diagram of how the plotter is organized. The three output modes are encapsulated in a virtual class called the DrawMethod, which is a member of the PlotInterface class. The DrawMethod contains a group of primitive drawing functions that are implemented by the three output modes. Hence, the PlotInterface does not 'see' beyond the DrawMethod class to see how exactly it is implemented or which output mode is actually plotting the data. If an entity such as a model needs to be plotted, all that the model needs to know is how to draw itself using the primitive drawing functions provided by PlotInterface's DrawMethod class. In this way the plotter does not need to 'know' what it is plotting. Once, the user decides what output mode to use, all the PlotInterface does is act as a go-between for the object to be drawn and the output mode. As shown in the figure, the Model object reads in the model information and/or the analysis results such as stress files and generates the contour plots by just interacting with the PlotInterface class.

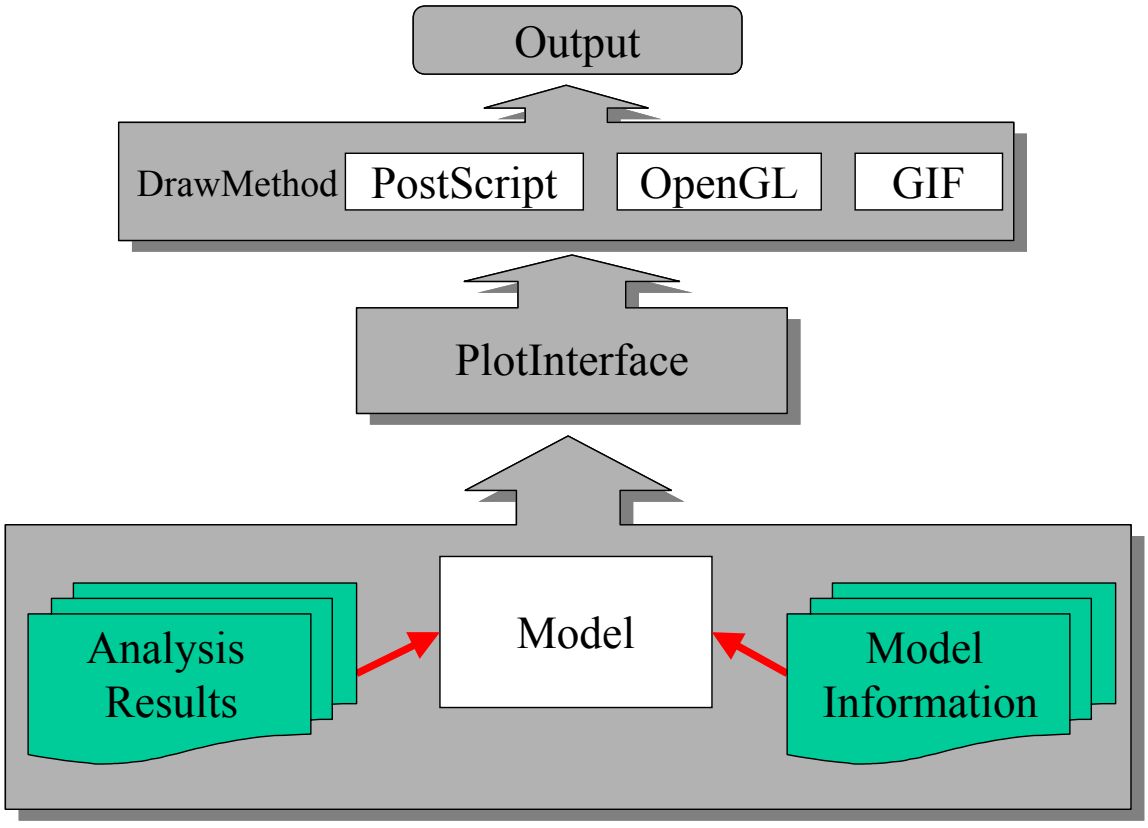

Figure 17: Plotter organization

There are a few advantages of this type of organization. If at a later point in time, it is decided to add another output mode to the plotter's capabilities, it will be very easy to modify the plotter. All that will have to be done is implement the primitive drawing functions in the DrawMethod class using the new output mode. This way, the plotter can still display all the objects that it previously did without making any modification to the design for that object. Similarly, if it is required that the plotter plot a new object, all that needs to be done is implement the plot function to display the object using the PlotInterface's drawmethod functions. One does not have to worry about how to plot to the different output modes. For example, if a hierarchical model is to be plotted, the only major work that is needed is to implement a function in the model class to plot using the PlotInterface.

The Plotter is able to read the mesh files that are used by Alpha and the Hierarchical System. The plotter can also read result data from analyses and give contour plots of displacement and stresses. It has various visual features that make it more effective and easy to interpret the results of an analysis. Figure 18 shows the stress contour plot of a quarter unit cell of a hexagonal array unidirectional fiber composite under uniaxial stress.

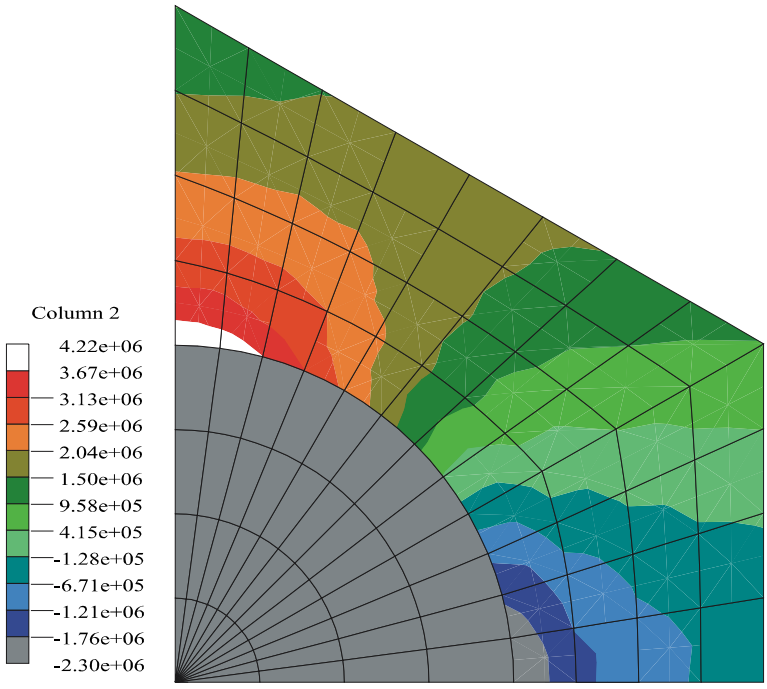

Figure 18: A stress contour plot using plotter

The plotter has the basic features that most mesh viewers possess such as labeling of nodes and elements and highlighting different elements based on user input. It is also possible to zoom in to observe the plot in detail. Another feature is the ability to rotate the mesh to get another perspective of the plot. The plotter employs its own hidden line removal algorithms to plot three-dimensional meshes or complex two-dimensional meshes.

For more complex analysis of results, it is possible to select and plot different elements from a mesh based on its material group. Other useful features include making specific elements transparent by directly specifying the element numbers or defining a region in the mesh. Contour plots can be tweaked by modifying the contour ranges and the magnification scale factor.

The plotter has a Graphical User Interface (GUI) version that allows you to see the object on the computer screen directly from within the program. It has dialog boxes

that let you make changes to the plotting settings and view the updated plots instantly. It also has a command line/batch version that uses a script file to read in commands. This is useful when a sequence of plots need to be generated from an analysis output. Another advantage of the command line version is that it can be used in the UNIX platform as well.

Graphical User Interface

The Graphical User Interface (GUI) for the hierarchical system is built on the existing visualization tool called Plot2000. The existing code was modified to be able to link with the Hierarchical definition module and thereby 'recognize' the new classes that were developed for the Hierarchical system. New dialog boxes were added to let the user interact with and visualize the hierarchical model. The GUI makes it possible for the user to create and analyze hierarchical models on the fly as opposed to using a scripting language. However, the scripting language is very important because it is an easy way to save the current state of a hierarchical model by letting the program write a script. This script can then be used by the program to load the hierarchical model back into the memory later on. Also, there may be cases where the user wants to run a sequence of analysis and doesn't really need to load a GUI. In such cases, it is best to use the command line version and run a batch file that does the job. Another advantage is that the command line version can be run on UNIX machines too while this GUI is built for Windows machines.

One of the most important components used to help the user visualize the hierarchical model is the tree control. This control is added to the main dialog box of the GUI and shows the hierarchical tree in a form similar to the directory structure in the hard disk. The different 'nodes' in the tree denote each model in the hierarchical tree. Right clicking on a model pops up a menu that gives a number of options to choose from. Figure 19 shows a tree control that contains a hierarchy of models and the pop-up menu. One of the options is to choose which model-related view is to be plotted in the plot window. There are basically two kinds of views- the complete model or the

component denoted by the node in the hierarchical tree. Two additional plots that are related to the model are the “basemodel copy” and the “basemodel”. The basemodel is the component view of the parent model of the currently selected model. The basemodel copy is the region in the parent model that is replaced by the current component.

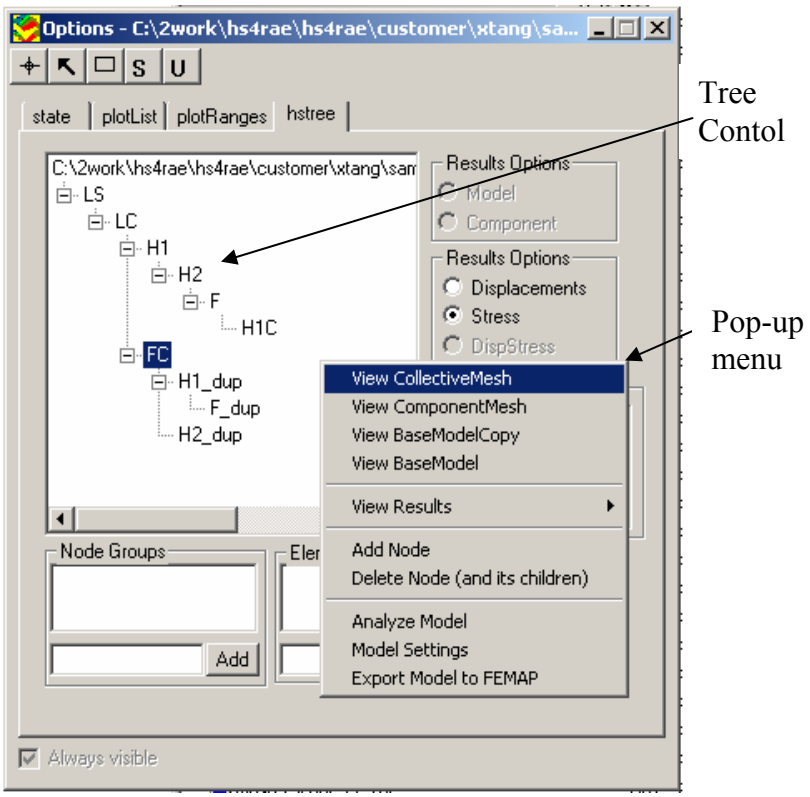

Figure 19: Tree control used to visualize hierarchy

There are a number of list boxes that show the different entities associated with a model like the element groups and node groups. On selecting any existing element group or node group, the corresponding entities in the mesh are highlighted on the plot window. It is also possible to create new groups using the GUI. One of the new features added to the plotter is the ability to use the mouse to select and highlight elements in the plot window. This selection can be saved to a group. Alternatively, you can also use the keyboard to specify which elements/nodes need to be in the group.

The main dialog box also shows the different result sets that are available to view i.e. the results of the different analyses that were conducted on the models. To go with that are different result visualization options, for example, whether to view the results for the component or the complete model, and whether to view the displacements or the stresses.

The GUI can be used to add or delete models in the hierarchical tree. This can be done by choosing the corresponding menu item in the pop-up menu of the tree control. When adding a model, the region in the current model that will be replaced by the new component has to be defined. This can be done by simply selecting the region with the mouse. There are two ways to do this. One is to do a simple point and click to select an element and repeat the process to select more elements. Clicking on the same element again will deselect the element. Another option is to choose the rectangle tool in the toolbar. This tool lets you define a rectangular region with two clicks – denoting any two opposing corners of the rectangle. This lets you select all the elements that fall within this rectangle. The ‘select’ and ‘unselect’ button in the tool bars let you use the rectangle tool to correspondingly select or unselect buttons. The ‘Add model’ menu option brings up a dialog box that asks for a name for the new model and a number of options to specify the mesh file for the new component. One option is to use an external mesh generator to create a mesh file. The path to the mesh generator can be entered in the textbox provided and the program can be launched. One thing to remember is that once the mesh has been created, it has to be converted to a format that can be understood by the hierarchical system. Another option makes use of an external mesh generator called GeomPack++ that has been interfaced with the hierarchical system. The GeomPack++ utility is an object-oriented program that runs on the command line and does not have a graphical user interface. This mesh generator can be used to refine the region that has been selected. The Hierarchical system is also interfaced to the FEMAP software. Using this option exports the selected region to FEMAP and launches the program. The extensive mesh generating tools of the FEMAP program can now be used to create the component for the new model. Once the mesh is created, it can be exported from

FEMAP to a NEUTRAL file, which can then be read by the hierarchical system. Finally, there is also the option of directly providing the path to the mesh file if it exists already. Choosing the 'delete model' menu item deletes the selected model and all the models that are derived from it.

Another important menu item is the 'Model Settings'. This option lets the user define the parameters required for conducting the analysis that would typically go into the ModelHandler script when using scripting. This brings up a comprehensive dialog box that collects information about the components, materials, constraints, loads and analysis and output options. The element group, node group and material group to be used for the analysis are selected from the list of groups that have been defined for the component. This process is repeated for all the active components in the model path of the current model. The material tab of the dialog box displays the material library and lets the user add or delete materials to the library. Currently, only the PlaneStress material can be entered using the GUI. The Constraint tab displays the number of constraints on the model and also allows the user to add or delete constraints. Similarly, the Load tab lets the user add, delete or modify the load conditions acting on the model. Only the PointLoad and Point Constraints can be entered using the GUI currently. The Analysis tab let the user choose which type of solver to use for the analysis, for example, Sparse, Olaf or Profile. Other parameters that can be included in this tab are the optional outputs, but this has not been implemented. For now, if the GUI is used to conduct the analysis, all the possible output files are created.

After all the parameters required for the analysis are defined, the system is now ready to conduct the analysis. To do this using the GUI, the user can choose the 'Analyze Model' menu item. Another option the user has is to export the model to FEMAP and let the FEMAP software do the analysis. This is made possible by interfacing the hierarchical system with the FEMAP software through the NEUTRAL files. It should also be possible to import the results back from FEMAP into the

Hierarchical system after the analysis has been conducted but this has not been implemented.

Another version of a GUI called Hviewer was developed which is very useful in illustrating the features of the Hierarchical Definition module and the boundary matching routines. This version is also written in C++ and uses the Microsoft Foundation classes but is built from scratch as compared to the other GUI, which is a modification of the existing plotter. HViewer uses the built-in hidden line removal features of OpenGL for plotting where as the Plotter uses in-house algorithms for this purpose. Figure 20 gives a screenshot of the HViewer program.

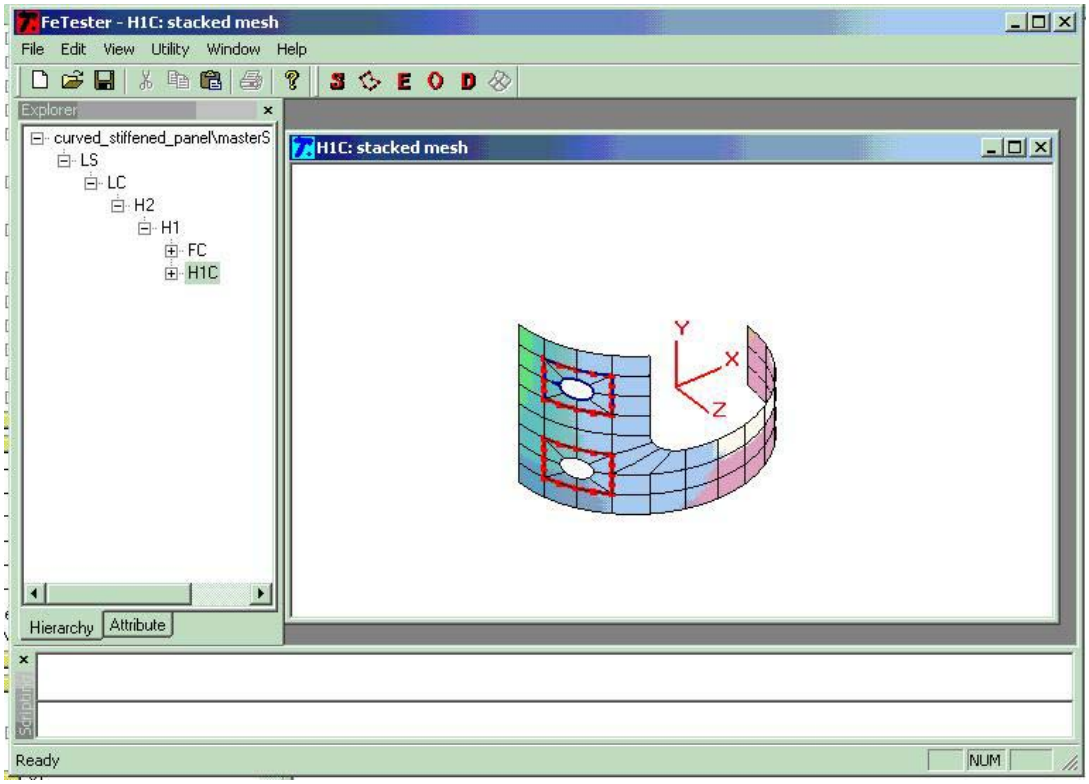

Figure 20: HViewer screenshot

Like the Plotter, the HViewer is also linked to the Hierarchical Definition Module and can understand all the hierarchical classes developed for this system. The HViewer also makes use of a tree control to visualize the Hierarchical tree and can

generate different views of the models like the model view, component view and the basemodel copy. The HViewer does not have the capability to set up an analysis model interactively but uses the scripting language. The HViewer's forte is its plotting features for visualizing hierarchical models. It can generate stacked mesh plots of a hierarchical model and outline boundaries including those between components that are matched. The HViewer uses the external Plotter program to view the results of an analysis like the displacement and stress contour plots.

A Visual Basic version of a GUI was also developed early on during the project in order to check the performance of the ActiveX control version of the Plotter. It did not have the ability to set up an analysis model interactively either. The GUI interfaced with the Hierarchical definition module using exported functions. Like the other GUIs, this one also used a tree control to display the hierarchical tree. Figure 21 gives a screenshot of the Visual Basic version of the GUI.

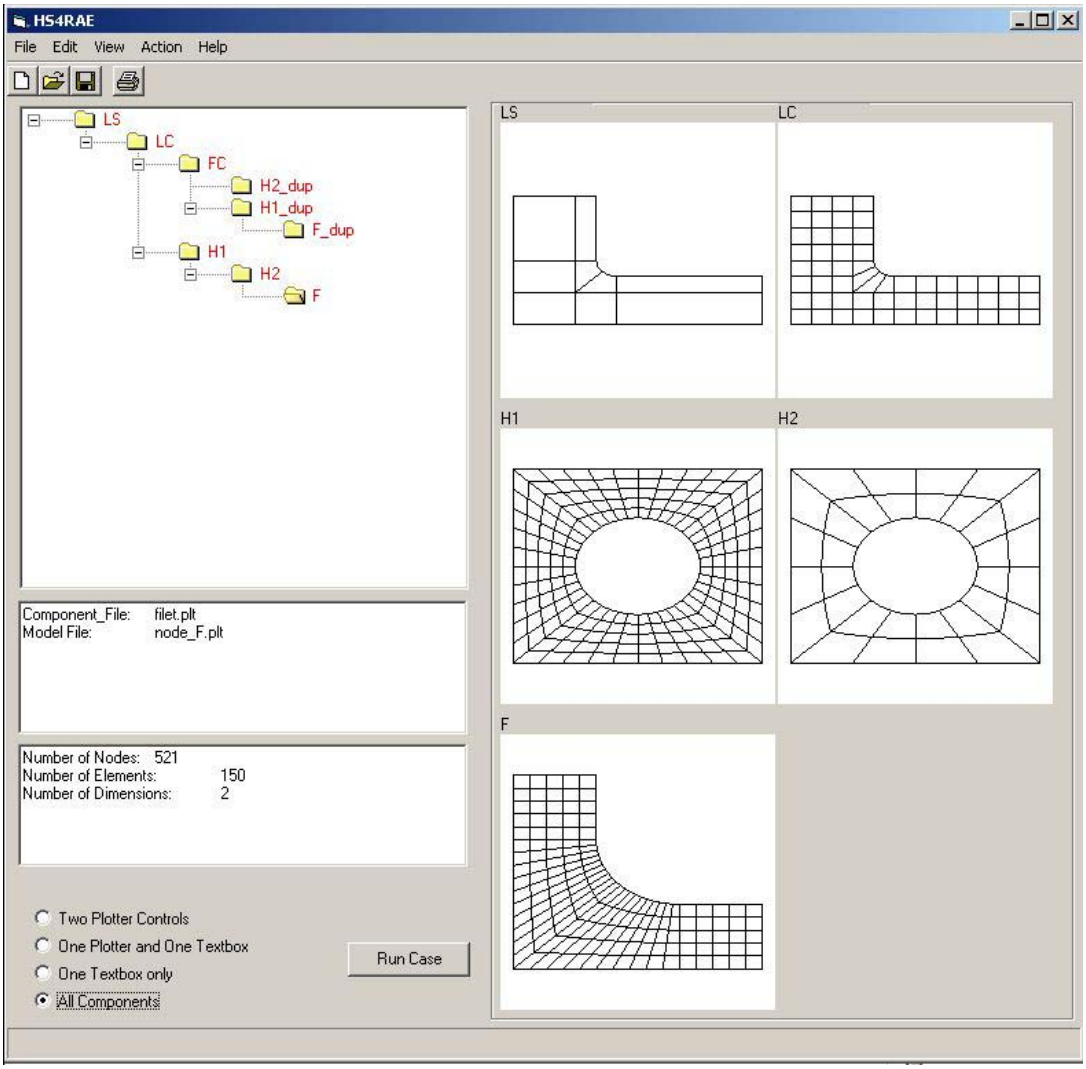

Figure 21: VB GUI screenshot

It has four different visualization options – use the plotter controls to plot a complete model view as well as a component view, plot the model view and display the numerical mesh data in a textbox, display the numerical data alone and lastly plot all the components in the model path of the currently selected model. The VB GUI made use of dynamic creation of ActiveX controls to create a number of plotter controls during run-time as per the number of models in the model path. The GUI also accesses information in the hierarchical model through the library functions and displays mesh information for the currently selected model in a textbox.

Micro-mechanical Analysis of Composites with Non-uniform Distribution of Fibers

The hierarchical system was initially developed for analyzing structural configurations using the global/local methodology but it was extended to detect critical regions in a non-uniform composite microstructure. This approach shows the ability to adapt the hierarchical system to analyze different types of problems.

Composite microstructures often contain non-uniformly distributed fibers having different sizes. The properties and performance of composites are related to their microstructure. It has been shown that plastic deformation, damage evolution and fracture of composites depend on the spatial distribution of fibers [40-45]. Analysis methods with models assuming periodic arrangement of the fibers is not enough to capture the response of composites with non-uniform fiber distribution. Work has been done to approach this problem by developing computer-simulated models that are statistically equivalent to the actual non-uniform microstructure of the composite [45].

The non-uniform distribution of fiber in the matrix is usually governed by the manufacturing process parameters. So, it is of interest to develop micromechanical models that can predict the response of composites with non-uniform distribution of fibers. Such models can be used to optimize the manufacturing processes and provide useful input for virtual integrated prototyping of the materials processes.

A three-level hierarchical analysis system is proposed to analyze unidirectional composites with non-uniform distribution of fibers. Computer-generated random microstructures are used to simulate the actual microstructure. For simplicity, the microstructure is assumed to have fibers of uniform size.

Figure 22 gives a schematic of the different levels in the hierarchy. The first level in the hierarchy identifies fibers and regions like matrix pockets for further analysis. This level is important because it is this level that decides which regions are analyzed using the finite element method. Therefore, the analysis at this level has to be

'intelligent'. What that means is that it should be able to 'intuitively' predict which regions are critical without actually running a finite element analysis.

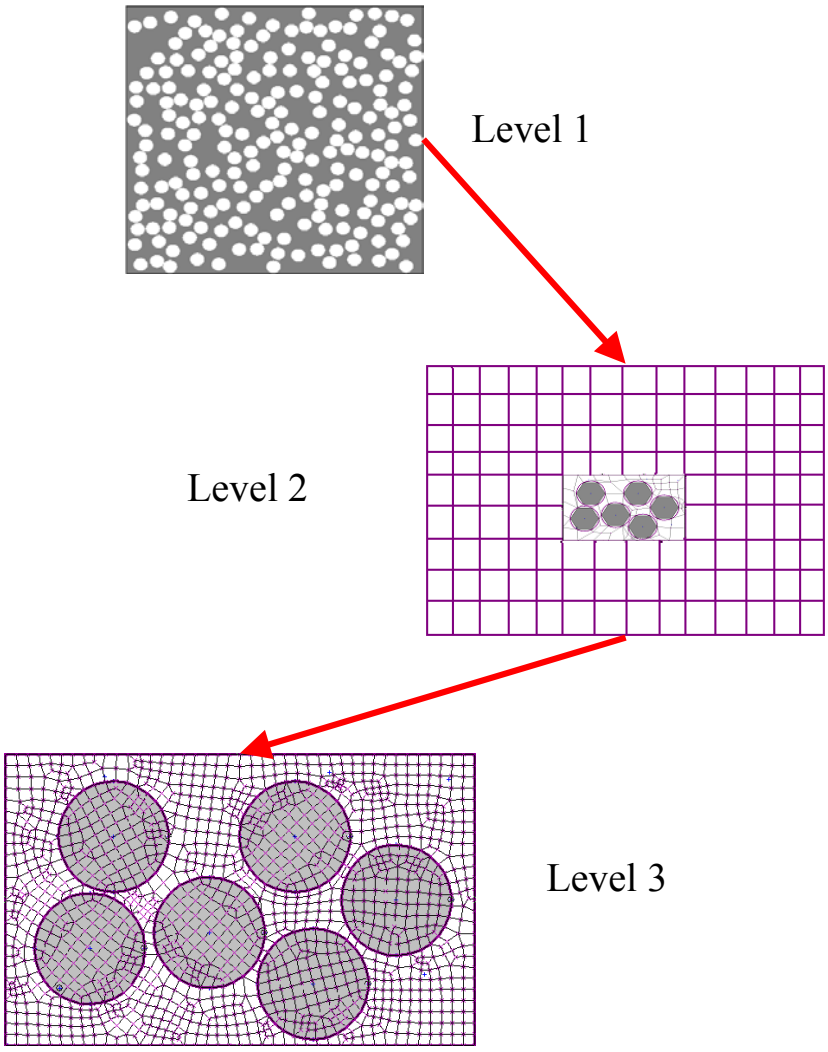

Figure 22: Hierarchy for analysis of non-uniform distribution

For example, high stress concentrations can be expected in regions where fibers are clustered very close to each other. Matrix pockets also initiate failure in composites. Algorithms can be developed to detect matrix pockets within a microstructure. Information obtained from the microstructure like the nearest neighbor distances of fibers are used to identify which regions will be analyzed further. The intelligence of this

low-level analysis can be further improved by including the effect of the loading conditions and the positions of the fibers on the selection of the critical regions. This is illustrated in Figure 23. Shown are two regions from the mesh of a microstructure with non-uniform fiber distribution. The region on the left has two fibers arranged along the y direction while the one on the right has two fibers arranged along the x direction. If the composite were to undergo uniaxial tensile loading in the x direction, the region on the right would experience higher stresses than the one on the left. Therefore, algorithms can be developed that take this kind of information into account when selecting critical regions. In the current implementation of the hierarchical system, only nearest neighbor distances are considered when selecting critical regions.

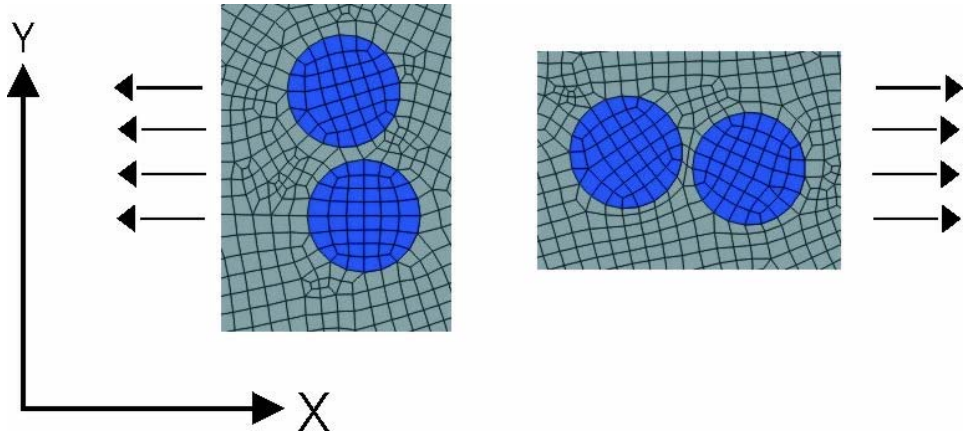

Figure 23: Effect of fiber position and loading in critical region selection

The next level conducts FE analysis on the regions identified in the first level. This is very valuable in terms solving time because the critical regions are so much smaller in size when compared to the entire microstructure. These savings can be used to analyze a more refined mesh. Just as in the structural global/local analysis methodology, a global model is first used to model the microstucture coarsely. Two ways to do this is by using effective homogenous properties or special elements that handle varying material properties [39] or maybe even a combination of both.

The region of interest is then modeled discretely and this becomes the local model. There are some parameters that can be tweaked to optimize the analysis. For example, instead of using the same effective properties through out the global model, the microstructure could be divided into regions and each region could be modeled using its own effective properties based on the fiber volume fraction in the region. By this method, the response of the microstructure due to fiber-rich or weak regions can be captured.

The size of the local model should be large enough so that the errors from the boundary due to the abrupt change in material properties or refinement do not affect the results from the region of interest. On the other hand, the model should not be so large that time is wasted in analyzing regions that probably do not contain any critical areas. Another tweak that can be done is to vary the refinement making the mesh denser as the region of interest is approached. This way there is no abrupt change in the refinement.

In addition to the level two analyses, a level three analysis can be conducted depending on the requirement. For example, a highly refined mesh can be used to get an even better feel for the stress distribution in the matrix. Cracks can also be introduced in order to study the progression of damage.

Properties such as strain energy density can also be used to determine ‘hot spots’ that are passed on to the next level for further scrutiny. If a zoom-in to a region is required, another local model can be generated and the hierarchical system will take care of joining it to the rest of the global model. This region could be a very small part of the whole microstructure that contains just three to four fibers. This methodology identifies a number of regions that could possibly initiate failure rather than come up with a single region that will be the cause of the failure. Each level can have sub-levels depending on the zoom-in area and the refinement required. Therefore, checks and limits have to be built into the system to make sure that the hierarchy does not become too large and time consuming as well as overwhelming for the user to interpret the results.

Implementation

Special models can be derived from the basic model class that has its own special properties that are tailored to meet the requirements of the problem. The same is true with the ModelHandler class. Specialized ModelHandlers can be derived from the basic ModelHandler class that can be used to analyze the problem at hand.

Two such classes were developed to approach this problem. One is the FDModel class, which is derived from the basic Model class, and the other is the FDModelHandler class, which is derived from the ModelHandler class. The FDModel class denotes the model in the first level of the proposed hierarchy for approaching the problem. The FDModel can store the microstructure information such as the position and radius of the fibers in the composite. It can also conduct the low-level analysis since it does not require a finite element analysis to calculate the nearest neighbor distances of each fiber.

Once the FDModel does the low-level analysis and comes up with a certain region that could have a high stress concentration, the model creates a global mesh and a local mesh. The hierarchical system does not have an advanced mesh generation capability and so it needs to use some external mesh generator. In this implementation, FEMAP was used to generate the mesh. Functions were written to export the geometry in the form of a FEMAP Neutral file. Along with the geometry, two dummy material properties are also exported so that the user can differentiate between the fiber and the matrix when generating the mesh. The geometry in terms of boundary surfaces were created that would make the fibers and the matrix regions. The user would then have to mesh these regions using the corresponding material properties. Depending on the type of mesh generator used, this process could perhaps be fully automated. Once the mesh is generated in FEMAP, some standard procedures must be done before it can be exported in a form that can be accepted by the import function. The mesh has to be checked for coincident nodes that were created when meshing the different boundary surfaces. Another thing that needs to be done is renumber the nodes so that the list of node numbers in the mesh is in sequence. This is done because the hierarchical system is

designed such that the node numbers have to be in sequential order. This is something that can be taken care of within the hierarchical system when importing the mesh with an additional function, but the renumbering feature already exists in the FEMAP software so it was decided to make use of it.

Now that both the global and local mesh has been generated, the two models must be defined in the hierarchy. The FDModel also creates the ModelHandler script for the two new models on its own. The two new models will be analyzed using the specialized FDModelHandler objects. These special modelhandlers account for that fact its parent model is not a basic model but an FDModel that represents a microstructure instead.

Figure 24 gives a schematic of the different classes that are used to define the models in the hierarchy. The material properties of the fiber and the matrix are provided in the ModelHandler Script file for the FDModel, which is the model that defines the microstructure. The FDModelHandler, which is used to run the FE Analysis of the global and local models, has the capability to inherit the Material properties from the FDModel. When the FDModel creates the global model for the FE analysis, the user has the option of choosing what type of material properties the global model should have. One of the options is to give the same effective properties through out the mesh. Another option is divide the global mesh into a number of regions and the each region will have its own effective properties depending on the volume fraction of the fibers in the region. Another option is to use macro elements to model the global model but this has not been implemented as yet.

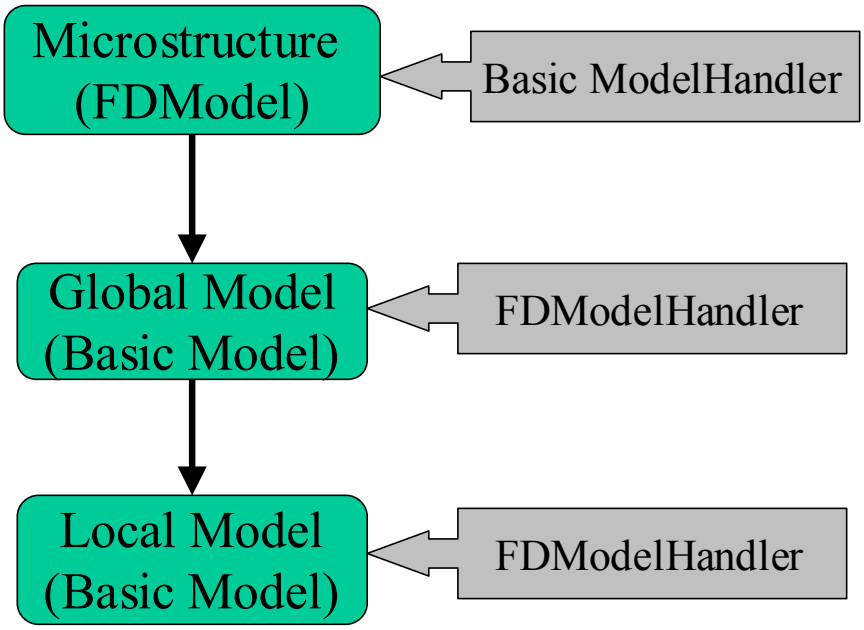

Figure 24: Classes in hierarchy

Currently, we are looking at only one type of loading, i.e. uniaxial stress. The FDModel applies the load and constraint conditions automatically to the two derived models. This is the simplest form of the hierarchy for this model. It is also possible to apply a gradient in the refinement of the global mesh with a more refined mesh closer to the local mesh. This is can be easily done by changing a parameter in the model script file. The stresses in the microstructure can be then visualized by viewing the contour plots using the GUI.

A sample case is discussed in Section 4 where a microstructure is analyzed to detect critical high stress regions that could initiate failure.

3. A STRUCTURAL GLOBAL/LOCAL ANALYSIS

This section covers a sample case using the hierarchical system to approach a problem. A structural problem is defined and it is analyzed using the global/local methodology that has been implemented in the hierarchical system.

Problem Definition

A practical problem is considered where the side panel of an airplane fuselage is analyzed. Figure 25 shows a Boeing 767 commercial airliner with the region of interest encircled. The aim is to model the side panel with the holes for the windows as well as the holes for the rivets that hold the windows intact and look at the stress concentrations in the region.

Figure 25: Analysis region in a typical commercial airplane (Source: Photo obtained from http://www.boeing.com)

Analysis Procedure

The problem is simplified by modeling the curved panel as a flat plate. The plate is subjected to a biaxial loading corresponding to the hoop stress and the longitudinal stress experienced by the fuselage due to the internal pressure in the cabin.

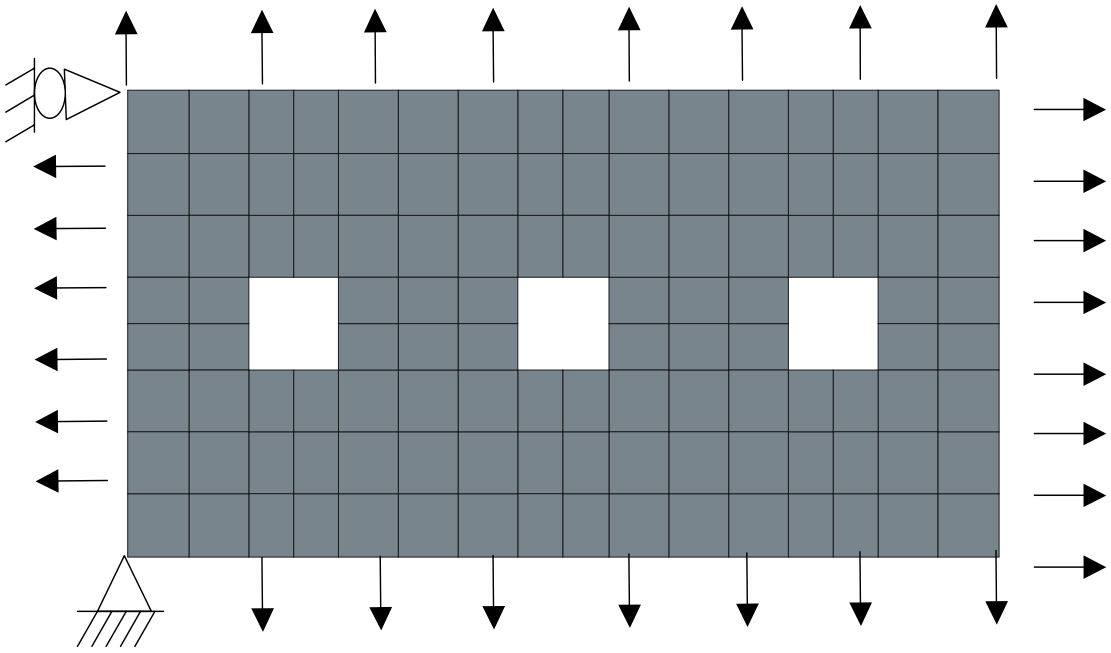

Figure 26: Model with constraints and loading conditions

A coarse global mesh is first used to model the panel. Figure 26 gives the constraints and loading conditions imposed on the global model. Three square holes are present in the global model to represent the windows. It does not need to be modeled perfectly in the global model but as we proceed to the local meshes the windows will have to be modeled more accurately.

The hierarchical models can be defined using the script files or through the GUI. An external mesh generator is used to create the meshes. Figure 27 shows the hierarchy of models that were generated. An external mesh generator called GeomPack++ was

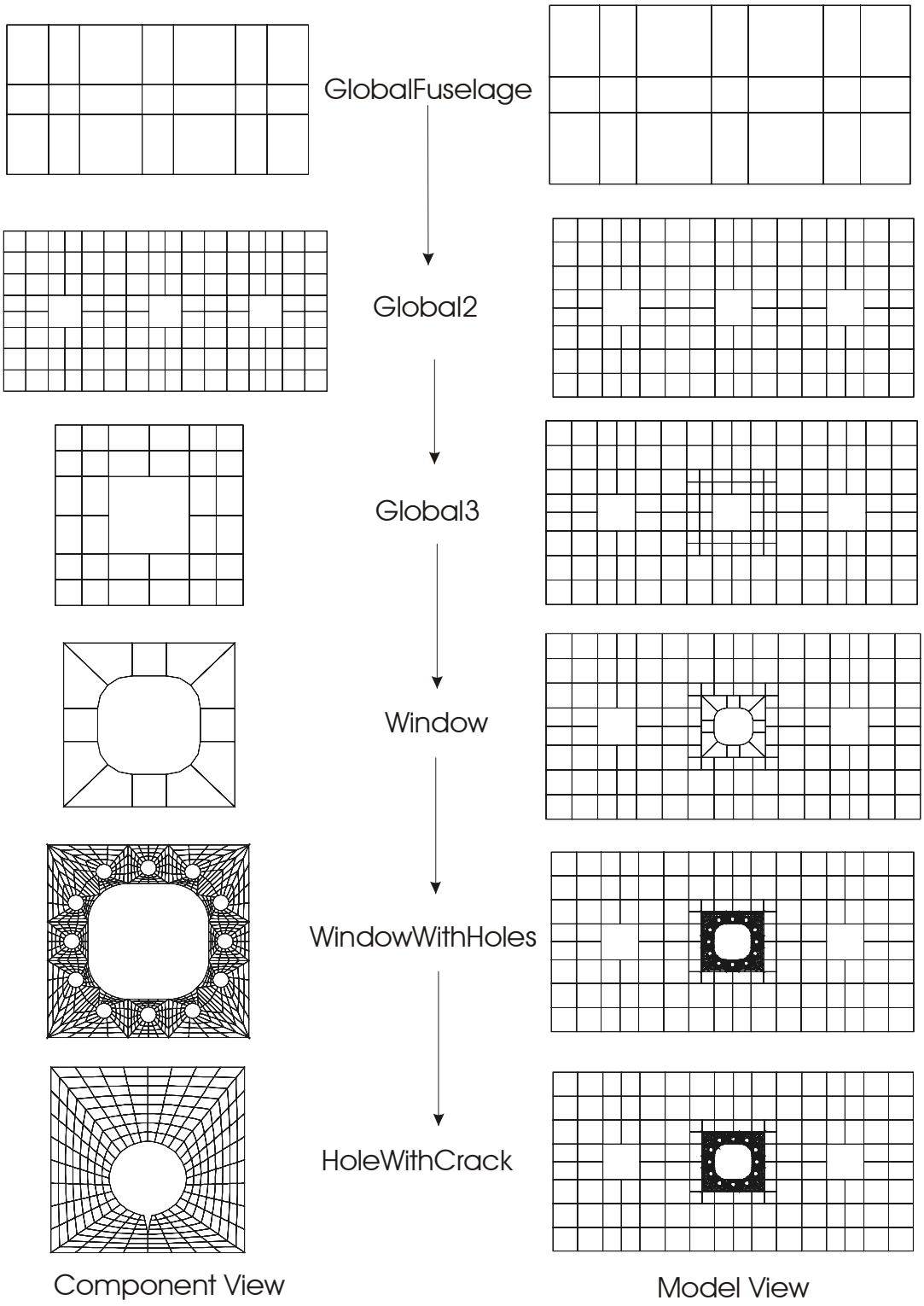

Figure 27: Hierarchy of models for analysis of side-panel

used to generate most of the meshes. Global2 and Global3 are intermediate models used to increase the refinement of the global mesh. The window is modeled over the center hole in the global mesh. The Window component takes out the corners from the inner edge of the window and makes them curved. The next component adds the rivet holes around the inner edge of the window. The GlobalFuselage model and the WindowWithHoles model are analyzed using the hierarchical system. The stresses have been normalized by the hoop stress to give the stress concentration factor (see Figure 28). It can be seen that since the global model is very coarse, the contours do not give much information. The second plot, which is the plot for the WindowWithHoles model shows a better idea of where the high stresses are. The third plot which is the a close up view of the WindowWithHoles model clearly shows that the high stress concentrations are in the region near the rivet holes on the four corners of the window.

The analysis results show the regions in the panel where there is the most chance for a crack to initiate. Further analysis can be done in terms of adding cracks in the panel and studying its propagation in the panel. In component HoleWithCrack, a crack is introduced at the boundaries of one of the rivet holes. The stress concentration contours of its analysis result (see Figure 29) shows very high stresses at the crack tip which is as expected.

The results of the analysis can be used to calculate the stress intensity factor for studying the fracture mechanics of the model. This example shows that the system can be developed for solving practical real-life problems.

If any modifications need to be made to the hierarchical models, for example, if another region in the panel needs to be analyzed, this can be done easily by generating the meshes and creating new models for the new region. The current state of the hierarchy can always be seen using the GUI.

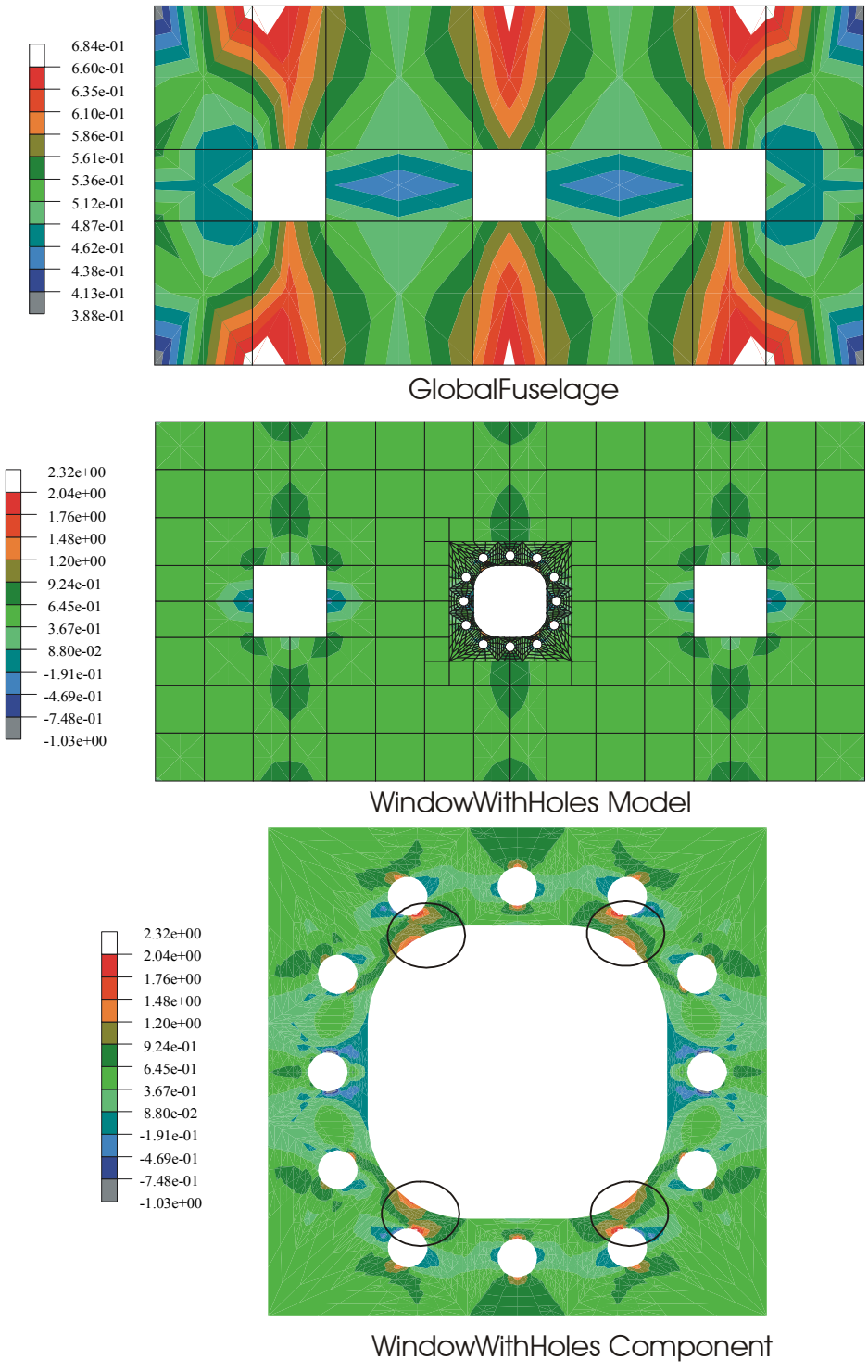

Figure 28: Stress concentration contours for GlobalFuselage and WindowWithHoles models

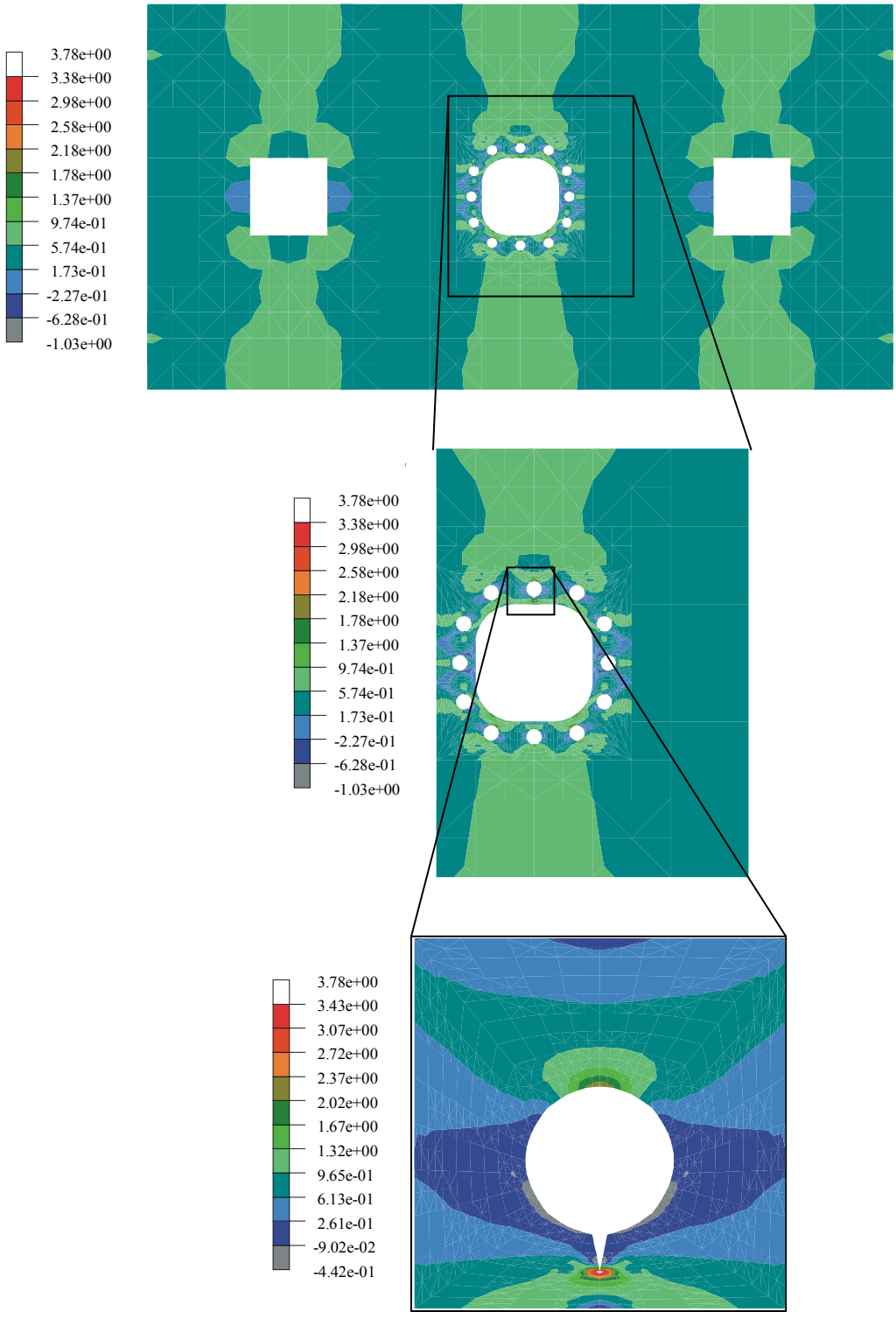

Figure 29: Stress concentration contours for HoleWithCrack Model

4. MICRO MECHANICAL ANALYSIS OF COMPOSITE WITH NON-UNIFORM FIBER DISTRIBUTION

This section goes through an application of the hierarchical system to conduct micro-mechanical analysis on a composite with non-uniform fiber distribution. The philosophy to this approach and its implementation in the hierarchical system is explained in Section 2.

Problem Definition

The aim is to find the critical regions in a microstructure with non-uniform spatial distribution of fibers in the matrix. A computer-generated microstructure of 30% fiber volume fraction is used to illustrate this problem. For the sake of simplicity, the size of the fiber is kept constant. The materials that form the fiber and matrix are assumed to be isotropic. In this example, only a uniaxial transverse loading is considered.

The results of the analysis are compared to a discrete model of the whole microstructure for accuracy. They are also compared to the results of analysis on a uniform microstructure with the same fiber volume fraction.

Analysis

As in the previous case, the script file can be used as a way to feed information to the hierarchical system to analyze this problem. An external utility is used to generate an artificial microstructure that randomly places fibers in the matrix till it reached the specified fiber volume fraction. Figure 30 shows the microstructure that was used.

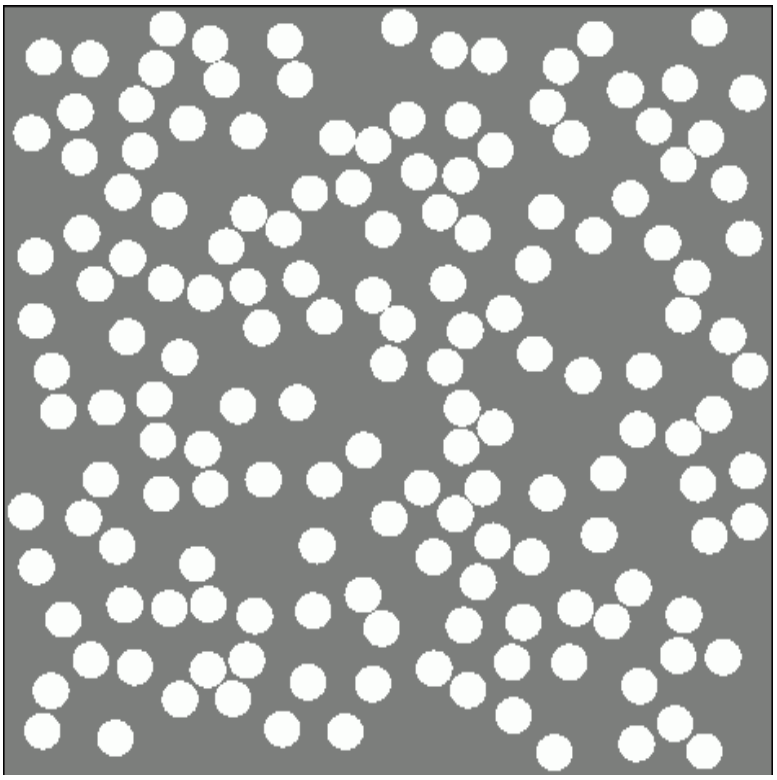

Figure 30: Computer-generated microstructure with 30% fiber volume fraction

An FDModel object is used to read the microstructure into the hierarchical system. The FDModel conducts the level one analysis and a list of fibers are generated which denote the region where the high stresses are most probable to occur. Upon specifying the region that needs to be analyzed further, the FDModel object then automatically generates the global meshes. By changing some parameters, the FDModel can create the global meshes using various methods. The user has the option to create the global model with the same effective properties through out the global model. the effective properties can be calculated on the fly using a micromechanics analysis of a uniform array of fibers in a matrix with the corresponding volume fraction. Another option is to use the Mori-Tanaka method to get the effective properties for a given fiber volume fraction. Other parameters include generating the global model with different properties for different regions based on the effective volume fraction for that region.

Since the hierarchical system does not have an advanced mesh generation utility, the features of the FEMAP software are used for this purpose. Export functions were written that generate a FEMAP Neutral file containing the geometry and material information of the region that needs to be meshed. After the mesh has been generated in FEMAP, it can then be imported back to the hierarchical system using a corresponding import function written for this purpose. The local model is then defined using this mesh and a ModelHandler script is then generated automatically by the FDModel object. This shows how special features can be added to models by simply deriving them from the basic model class. A special ModelHandler object called the FDModelHandler is used to conduct the global/local analysis. One of the characteristics of this ModelHandler is that it can automatically inherit the material properties from the FDModel. The results for the analysis by the hierarchical system are compared to a separate conventional analysis result of the whole microstructure.

Figure 31 shows the comparison of results between the region analyzed using the hierarchical system with the same region analyzed conventionally. As you can see, the stress plots are almost exactly the same. Figure 32 shows the full microstructure analyzed. Figure 32 gives the stress distribution for the whole microstructure. The high stress regions are marked with a white square. Although the hierarchical system was able find a few of the critical regions, some of them were not captured by the analysis. This shows that the low level analysis needs to be more intelligent, as mentioned earlier. It should take into consideration the position of the fibers and the type of loading conditions.

The plots are also compared with the equivalent composite with uniform distribution of fibers. Figure 33 shows that the highest stress in the periodic array is less than half the highest stress in the microstructure with non-uniform distribution. this proves that models that assume periodic boundary conditions cannot be used to capture the response of composites with non-uniform distribution of fibers.

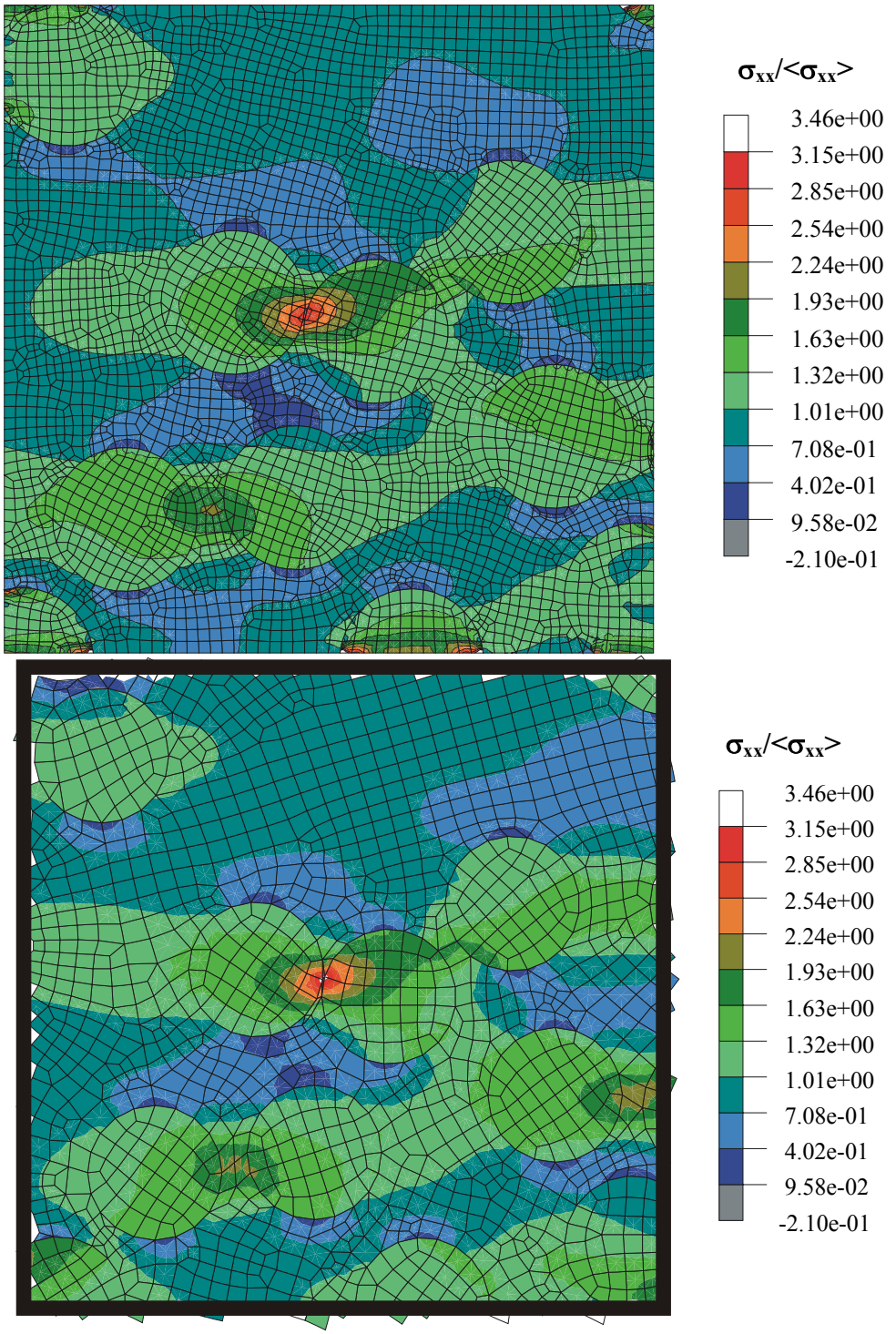

Figure 31: Comparing stress concentration factor results using hierarchical system (top) with corresponding region in the whole microstructure analyzed conventionally (bottom)

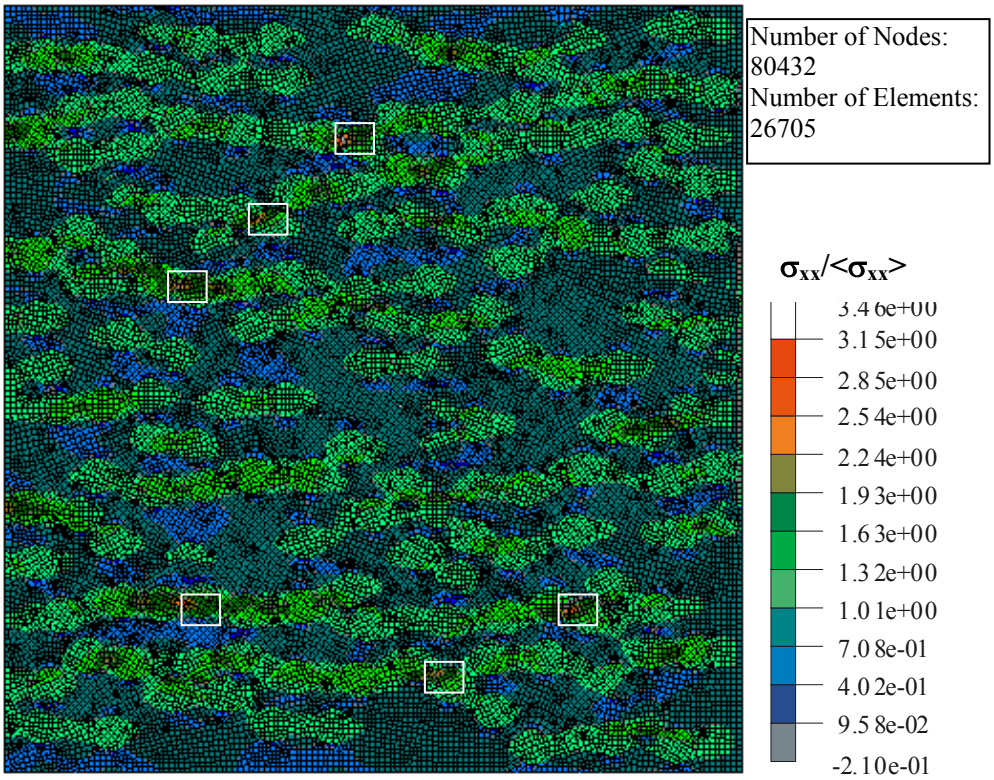

Figure 32: Stress concentration factor distribution for the whole microstructure analyzed conventionally

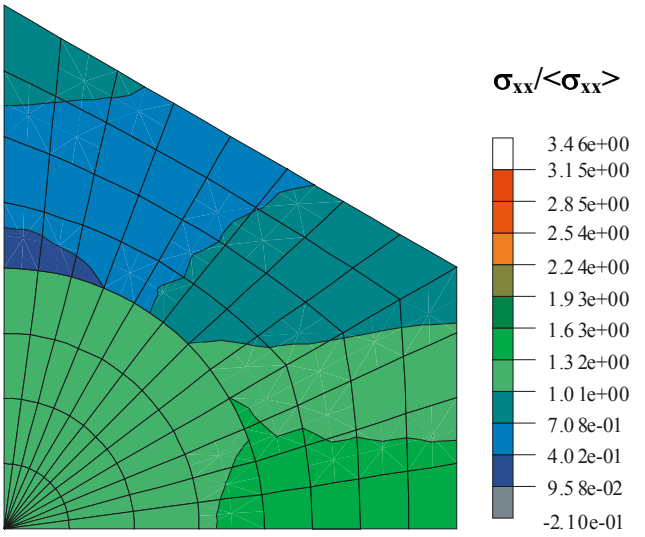

Figure 33: Stress concentration factor distribution using a periodic array composite

5. CONCLUSIONS AND FUTURE WORK

A new philosophy is introduced where the natural hierarchical character of model descriptions and simulation results are exploited to expedite analysis of problems. The effort resulted in a mix of mechanics, well-designed data structures and software - interfaces that forms a rapid analysis environment. Models can be organized using a hierarchical strategy with the capability to interact with other models. The system's forte is its hierarchical framework that allows models to communicate with each other and share information with one another. The application examples presented in this work show that the hierarchical system could be easily adapted to solve a wide range of problems.

This kind of technology would not only expedite analysis of structures, but also analysis and design of complex materials on scales ranging from that needed to design with nanotubes to textile composites. The environment could be adapted to analyze problems that involve multiple scales as well as multiple governing equations. One good example for such a case is the research of carbon nanotubes, which is a very promising material for the structures of tomorrow's world. Since the nanotubes are extremely small in size, the governing equations are no longer the same as in continuum mechanics. At an atomistic scale, the quantum mechanics and molecular models are more prevalent while the continuum laws work best in the macro-scale. As we try to bridge different scales to build materials that use nanotubes, there is a need for the technology to model these materials and study their properties. This requires coupling of different methodologies and theories and involves data transfer from one scale to another. This kind of approach makes ideal for the hierarchical system.

The philosophy using a hierarchy description seems to have a big potential for solving finite element analysis problems. It can easily reduce the human effort in generating new models and making modifications to existing models. It is especially ideal for global/local analysis due to the inherent hierarchical nature of the method. This

kind of an environment involves many tools such as mesh generators, finite element solver, visualization tools and the hierarchical definition module.

The utilities developed for this project are meant for illustrating this philosophy and cannot compete with commercial FEA software that has been developed over a long period of time. With more effort, many useful features can be included in the hierarchical system to make it more effective and enable rapid finite element analysis.

The focus of this research has been on developing the framework and the methodology to implement this philosophy. Two types of problems were analyzed by using this environment. One was a standard structural problem, which was analyzed using the global local methodology using the hierarchy. The other problem pursued was the analysis of unidirectional composites with a non-uniform spatial distribution of fibers. This problem is a very relevant one because in reality it is very unusual to come across composites that have a very periodic microstructure. Although, the mentioned problems were chosen to give an example of how the environment can be used, the power of the hierarchical strategy is not limited to solving these problems alone.

More features can be added to the visualization tool to assist the user in interpreting results and making better decisions. A problem with the multi-point constraints was discussed in this work. A reliable, robust and accurate method needs to be developed to ‘join’ different components in the hierarchy. In addition to using conventional MPCs, the interface technology [46] can be used to ‘combine’ components forming the complete analysis model. Currently the developed system can handle only two-dimensional cases. It would not be complete without robust boundary detection routines for three-dimensional geometries.

REFERENCES

1. Mathematical and Information Sciences website - http://www.cmis.csiro.au/bfe/UniqueOptions.htm (accessed: September 14, 2003)

2. Pinkerton, Steven D., 1993, "Optimization of hierarchical structures", *Journal of Information Processing and Cybernetics*, Vol. 29, No.4, pp. 221-231.

3. Stilman, B., 1994, "Hierarchical networks for space navigation", *Proc. of the IX Int. Conf. on Applications of Artificial Intelligence in Engineering (AIENG'94)*, Penn State Univ., Malvern, PA, July 1994, pp. 339-348.

4. Oden, J. T., Vemaganti, K. and Moes, N., 1999, "Hierarchical modeling of heterogeneous solids", *Computer Methods in Applied Mechanics and Engineering*, Vol. 172, pp. 3-25.

5. Saito, K., Araki, S., Kawakami, T., Moriwaki, I., 1995, "Global-local finite element analysis of stress intensity factor for a crack along the interface of two phase material", *Proceedings of the 1995 10th International Conference on Composite Materials*, Cambridge, UK, Aug 14-18 1995, pp 261.

6. Whitcomb, John D., Woo, Kyeongsik, 1991, "Evaluation of iterative global/local stress analysis", *American Society of Mechanical Engineers (Publication) NDE, Enhancing Analysis Techniques for Composite Materials*, Vol. 10, pp. 201-205.

7. Woo, Kyeongsik, Whitcomb, John, 1994, "Global/local finite element analysis for textile composites", *Journal of Composite Materials*, Vol. 28, No. 14, pp. 1305-1321.

8. I. Babuska, B. A. Szabo and I. N. Katz, 1981, "The *p*-version of the finite element method", *SIAM J. Numer. Anal.,* Vol. 18 , pp. 515-545.

9. I. Babuska and B. Szabo, 1982, "On the rates of convergence of the finite element method", *International Journal for Numerical Methods in Engineering*, Vol. 18, pp. 323-341.

10. Zhu, D. C., 1986, "Development of hierarchical finite element methods at BIAA," *Proceedings of the International Conference on Computational Mechanics*, Tokyo I, pp. 123-128.

11. Zienkiewicz, O. C. and Taylor, R. L., 1991, *The Finite Element Method*, 4th ed., McGraw-Hill, London.

12. J. Fish, 1992, "The s-version of the finite element method", *Comput & Struct*, Vol. 43, pp. 539-547.

13. J. Fish and R. Guttal, 1996, "The s-version of finite element method for laminated composite shells, *Internat. J. Numer. Methods Engrg.,* Vol. 39, No. 21, pp. 3641-3662.

14. J. Fish, 1992, Hierarchical modeling of discontinuous fields, *Comm. Appl. Numer. Methods,* Vol. 8, pp. 443-453.

15. C.D Mote, 1971, "Global-Local finite element", *International Journal for Numerical Methods in Engineering,* Vol. 3, pp. 565-574.

16. S. R. Voletia, N. Chandraa, and J. R. Millerb, 1996, "Global-local analysis of large-scale composite structures using finite element methods", *Computers & Structures*, Vol. 58, pp. 453-464.

17. K.M Mao and C.T Sun, 1991, "A refined global-local finite element analysis method", *Internat J. Numer. Methods Engrg.*, Vol. 32, pp. 29-43.

18. J.D. Whitcomb, 1991, "Iterative global-local finite element analysis*", Comput. & Structures*, Vol. 40, pp. 1027-1031.

19. I. Babuska, T. Strouboulis, C. S. Upadhyay, and S. K. Gangaraj, 1995, "A Posteriori estimation and adaptive control of the pollution error in the h-version of the finite element method", *International Journal for Numerical Methods in Engineering*, Vol. 38, No. 24, pp. 4207-4235.

20. J.N. Reddy and D.H. Robbins, 1994, "Theories and computational models for composite laminates", *Appl. Mech. Rev*., Vol. 47, pp. 147-169.

21. N.F. Knight, Jr., J.B Ransom, O.H. Griffin and D.M. Thompson, 1991, "Global/local methods research using a common structural analysis framework", *Finite Elem. Anal. Design*, Vol. 9, pp. 91-112.

22. A.K. Noor, W.S Burton and J.M Peters, 1990, "Predictor- corrector procedures for stress and free vibration analyses of multilayered composite plates and shells", *Comput. Methods Appl. Mech. Engrg.,* Vol. 82, pp. 341-363.

23. Lee, K., Moorthy, S., Ghosh, S., 1999, "Multiple scale computational model for damage in composite materials", *Computer Methods in Applied Mechanics and Engineering,* Vol. 172, No. 1-4, pp. 175-201.

24. Noor, Ahmed K., Starnes, James H. Jr., Peters, Jeanne M., 2000, "Uncertainty analysis of composite structures", *Computer Methods in Applied Mechanics and Engineering,* Vol. 185, No. 2-4, pp. 413-432.

25. Ransom, J. B., Knight, N. F. Jr., 1990, “Global/local stress analysis of composite panels.”, *Computers and Structures*, Vol. 37, pp. 375–395.

26. A.K. Noor , 1986, “Global-local methodologies and their application to nonlinear analysis”, *Finite Elements Anal. Design*, Vol. **2**, pp. 333–346.

27. Fish, J., Suvorov, A., Belsky, V., 1997, “Hierarchical composite grid method for global-local analysis of laminated composite shells”, *Applied Numerical Mathematics*, Vol. 23, No. 2, pp. 241-258.

28. J. Bramble, R. E. Ewing, J. E Pasciak and A.H. Schatz, 1988, “A preconditioning technique for the efficient solution of problems with local grid refinement*”, Comput. Methods Appl. Mech. Engrg,* Vol. 67, pp. 149-159.

29. A. Brandt, 1977, “Multi-level adaptive solutions to boundary-value problems”, *Math. Comp.,* Vol. 31, pp. 333-390.

30. J.Fish and V. Belsky, 1995, “ Multigrid method for periodic heterogeneous media. Part 2: Multiscale modeling and quality control in multidimensional case*”, Comput. Methods Appl. Mech. Engrg.,* Vol. 126, pp. 17-38.

31. J. E Flaherty, P.K. Moore and C. Ozturan, 1989, “Adaptive overlapping methods for parabolic systems, in: J.E Flaherty, P.J Pslow, M.S. Shephard and J.D Vasilakis, eds., *Adaptive Methods for Partial Differential Equations*”, *SIAM*, Philadelphia, PA.

32. S.D. McCormick, 1989, “Multilevel Adaptive Methods for Partial Differential Equations”, *SIAM Frontiers in Applied Mathematics*, Philadelphia, PA.

33. S.F McCormick and J.W. Thomas, 1986, “The fast adaptive composite grid (FAC) method for elliptic equations”, *Math. Comp.,* Vol. 46, pp. 439-456.

34. T. Belytschko, J. Fish and A. Bayliss, 1990, “The spectral overlay on finite elements for problems with high gradients”, *Comput. Methods Appl. Mech. Engrg*., Vol. 81, pp. 71-89.

35. J. Fish and R. Guttal, 1995, “The p-version of finite element method for shell analysis”, *Comput. Mech Internat. J.,* Vol. 16, pp. 1-13.

36. Fish, J., Markolefas, S., 1994, “Adaptive global-local refinement strategy based on the interior error estimates of the h-method*”, International Journal for Numerical Methods in Engineering,* Vol. 37, No. 5, pp. 827-838.

37. J. Fish and S.Markolefas, R. Guttal and P. Nayak, 1994, “On adaptive multilevel superposition of finite element meshes for linear elastostatics”, *Appl. Numer. Math*., Vol. 14, pp. 135-164.

38. H. Yserentant, 1986, “On multilevel splitting of finite element spaces”, *Numer. Math.*, Vol. 49, pp. 379-412.

39. Woo, K and Whitcomb, J. D., 1993, “Macro Finite Element using subdomain integration”, *Communications in Numerical Methods in Engineering*, Vol. 9, pp. 937-949.

40. Whited, W., Gokhale, A. M. and Deshpande, N.U., 1994, “Characterization of thermally induced microcracks in a metal matrix composite”, *Microstructural Science,* Vol. 21, pp.107-120.

41. J.R.Brockenbrough, S.Suresh and H.A.Wienecke, 1991, “Deformation of metal-matrix composites with continuous fibers: Geometrical effects of fiber distribution and shape*”, Acta Metall. Mater.,* Vol. 39, pp. 735-752.

42. Pyrz, Ryszard, 1993, "Interpretation of disorder and fractal dimensions in transversely loaded unidirectional composites*", Polymers & Polymer Composites,* Vol. 1, pp. 283-295.

43. B.F. Sorensen, R. Talreja and O.T. Sørensen, 1992, “Residual stresses in ceramic fiber composites: effects of non-uniform fiber distribution”, *Proc 4th Int Symp on Ceramic Materials and Components for Engines*, pp. 743–756.

44. S. Yang, Asim Tewari And Arun M. Gokhale, 1997, “Modeling of non-uniform spatial arrangement of fibers in a ceramic matrix composite”, *Acta Materialia,* Vol. 45, pp. 3059-3069.

45. Shichen Yang, A. M. Gokhale and Zhaohui Shan, 2000, "Utility of microstructure modeling for simulation of micro-mechanical response of composites containing non-uniformly distributed fibers", *Acta Materialia*, Vol. 48, pp. 2307-2322.

46. ANSYS, Inc., 1999, “DesignSpace User Manual for Release 5.x”, July 1999.

47. Paul Nielan, 2002, *Sandia Lab News*, Vol. 54, Special Issue, Feb 2002.

48. NextGrade (Next Generation Rapid Analysis and Design Environment), http://www.ara.com/nextgrade.htm (accessed: September 14, 2003)

49. Schoof, L., Yarberry, V., 1995, “Exodus II – A Finite Element Data Model”, http://endo.sandia.gov/SEACAS/Documentation/exodusII.pdf, SAND92-2137, Printed November 1995.

50. Ransom, J. B., McCleary, S. L., and Aminpour, M. A., 1993, “A New Interface Element for Connecting Independently Modeled Substructures,” *AIAA Paper*, pp. 93-1503.

51. Ransom, J.B., 2001, “On Multifunctional Collaborative Methods in Engineering Science”, NASA/TM-2001-211046, September 2001.

52. Ransom, Jonathan B., 1997, “Interface technology for geometrically nonlinear analysis of multiple connected subdomains”, *Collection of Technical Papers - AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference,* Vol. 3, pp. 1862-1872

53. Wang, John T., Ransom, Jonathan B., 1997, “Application of interface technology in nonlinear analysis of a stitched/RFI composite wing stub box”, *Collection of Technical Papers - AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference,* Vol. 3, pp. 2295-2310.

APPENDIX A

Scripting Language

This section explains the scripting language and its syntax. There are three kinds of scripts that can be written. One is the model scripting language, which is used to define the hierarchical tree. This script does not contain the information needed to conduct a complete analysis. The model script for a hierarchical model 'points' to the corresponding model handler script for a particular model in the tree, which has the detailed information needed for the system to conduct an FEA analysis. The last kind of script is the region definition script file that is a set of commands used to define a region in a mesh.

All scripts are terminated with the 'end' keyword. The following are the keywords in the model scripting language:

DefineModelName (or DefineName): Defines the name of a tree node.
Format:
[execModel::]DefineModelName|DefineName model modelName
If "model" is presented, "execModel" is discarded. Otherwise
"execModel" or the current model is to be assigned "modelName".

DefineModelMesh (or DefineMesh): Reads mesh for component.
Format:
[execModel::]DefineModelMesh|DefineMesh modelName meshFileName

CreateModel: Creates a tree node.
Format:
[execModel::] CreateModel modelName [meshFileName] [Region Definition FileName]

OutputModel (or OutputNodalModel) : outputs a conventional mesh for the complete model association with the tree node.
Format:
[execModel::]OutputModel [ModelName] MeshFileName

OutputComponent : outputs mesh for component.

Format:
[execModel::]outputComponet [modelName] outputName active|deactive

SystemCall: to run system commands from within the script.

Format:
[execModel::]systemCall

command

PrintModelTreeInfo: prints graphic tree, info for each tree node and outputs model component for each tree node and creates input file for VB GUI.

Format:
PrintModelTreeInfo

setModel (or setReader or setCommandProcessor) : Set current tree node. Pairs with an 'ExitModel' or 'exitUseModel' keyword.

Format:
setModel | setReader | setCommandProcessor modelName

…

…

…

ExitModel | exitUseModel

moveModelTo: moves position of model in tree.

Format:
[execModel::]moveModelTo [fromModelName] toModelName

toModel is the new baseModel of fromModel

fromModel can not be the rootModel

createModel_reuseMesh (or duplicateModel or copyModel) :duplicates a component and adds into tree, eventually add offsets, transformations , mirroring

Format:

[execModel::]duplicateModel | duplicateModel | copyModel newModel

duplicatedModel

outputAllModels (or outputAllNodalModels) :outputs all models in tree.

Format:

outputAllModels | outputAllNodalModels

DefineElementGroup: defines a group of elements in a model
Format:
[execModel]::DefineElementGroup groupName selection attribute

DefineNodeGroup: defines a group of nodes in a model
Format:
[execModel]::DefineNodeGroup groupName selection attribute

ModelHandler: To create a ModelHandler and attach it to a tree node.
Format: [?::]ModelHandler execModel modelScriptFileName [HandlerType]
"execModel" must be specified.
If "HandlerType" if presented, it will be used to generate appropriate Handler.

The ModelHandler Scripting language is used to control the analysis of a model and contains the necessary data for conducting a finite element analysis. The ModelHandler script typically specifies the required parameters/information in the following order:

1. Solver
2. For all components, specify
    - Element type
    - Node DOF
    - Material
3. Material Library
4. Constraints
5. Loads
6. Generate DOF map
7. Glue (MPC)
8. Analysis command

9. Output options

The syntax for specifying each of the above mentioned parameters are given below:

1. Solver:

SetStorageMethod

SPARSE | OLAF | PROFILE

2. Specify Component information:

A data block with this format is written for each component in the model:

ComponentSettings <Component Name>

material <Material GroupName>

element <Element GroupName>

DOF <Node GroupName>

exitComponentSettings

3. Material Library: a data block with the following format is written for each material to be added to the library. Within each block the format of the information would differ depending on the type of material being defined. The example below gives the format for a Plane Stress material.

ReadMaterials

PlaneStress

<Material Library Index> <MaterialName>

<E11> <E22> <nu12> <G12> <a11> <a22>

exitReadMaterials

4.Constraints: this section starts with 'ReadConstraints' and is terminated with 'exitReadConstraints'. A data block is written for each Constraint type to be added to the constraint Set. Within each block the format of the information would differ depending on the type of constraint being defined. The example below gives the format for a Point Constraint.

ReadConstraints

PointConstraint

<Component name> <node number> <1 | 0> [<1 | 0> <1 | 0> … for each DOF]
exitPointConstraint
exitReadConstraints

5. Loads: this section starts with 'ReadLoads' and is terminated with 'exitReadLoads'. A data block is written for each Load type to be added to the Load Set. Within each block the format of the information would differ depending on the type of Load being defined. The example below gives the format for a Point Load.

readLoads
PointLoad
<Component Name> <Node Number> <magnitude> <direction>
exitPointLoad
exitReadLoads

6.Generate DOF map: this directive is issued by the command 'setComponentDofMap'

7.Glue (MPC):

setComponentMPCGlue <Glue type>

8. Analysis command: this directive is issued by 'DoAnalysis'

9.Output options: this section starts with 'doOptionalOutput' and is terminated with 'exitOptionalOutput'. A line is written for each additional output option such as displacement and stress. Examples are given for both types of outputs.

displacement [<component Name> <nodegroup Name>]
stress [<component Name> <element groupName>]

If the optional information is not supplied, then the output for the whole model is obtained.

The Region Definition files are used to define a region in a mesh. The program reads and interprets the commands in the region definition files to create a list of elements that define the region. The different commands and its syntax is explained below:

1. **DefineRectangleRegion**: this command uses the bottom left corner and top right corner of a rectangle to add a rectangular region to the defined region.

Syntax:

DefineRectangleRegion

<X1 Y1 (bottom left coordinate)>

<X2 Y2 (top right coordinate)>

2. **AddElement:** this command adds an element of a specified component to the list of elements in the defined region

Syntax:

AddElement <component name> <element number>

3. **RemoveElement:** this command removes an element of a specified component to the list of elements in the defined region

Syntax:

RemoveElement <component name> <element number>

4. **AddElementList:** this command adds a list of elements to the defined region

Syntax:

AddElementList

<Component name> <element number>

…

…

End

VITA

Julian Varghese was born in India. He obtained his Bachelor of Technology degree in mechanical engineering from the College of Engineering, University of Kerala, Kerala, India in May 2000. Before joining Texas A&M University in the fall of 2001, he worked as a systems analyst for a year at Cognizant Technology Solutions, Chennai, India. Upon completion of his master's work, he plans to pursue a PhD degree to further this research.

His permanent address is:
Kuttikadan House,
Nandikara P.O.,
Thrissur, Kerala– 680301
INDIA.
Phone: (91)-480 2752337
e-mail: julvar@tamu.edu
julvar@royalmexx.com