F. A. Cosso and E. J. Barbero, Computer Aided Design Environment for Composites, SAMPE 2012 Conference and Exhibition, Baltimore, May 21-24, 2012.

COMPUTER AIDED DESIGN ENVIRONMENT FOR COMPOSITES

Fernando A. Cosso and Ever J. Barbero
West Virginia University
Morgantown WV 26506-6106

ABSTRACT

Preliminary design of composite laminate structures (CLS) involves a set of calculations starting from micromechanics, macromechanics, and structural analysis such as beam theory. The design variables for CLS include fiber and matrix properties, fiber volume fraction, stacking sequence, section geometry, and so on. When one of these variables is changed during the design process, all the calculations to model the CLS have to be redone. This paper discusses the use of object oriented programming (OOP) and relational databases (RDB) to mimic the design process of CLS and its elements, including laminate, lamina, and fiber. This approach ensures consistency of data and property values calculated at different design stages by enforcing known relations between the different objects in the CLS. The ultimate objective is to create a software application allowing the designer to optimize the composite without the burden of repeating calculations. The application, www.cadec-online.com, attains maximum user convenience and real time deployment of software updates.

1. INTRODUCTION

Composite materials design requires the optimization of several variables such as fiber volume fraction, thickness, stacking sequence, etc. When one of these variables is changed, the designer has to recalculate the entire properties dependant on it. Recalculations are extremely tedious and error prone. The question that arises is how to automate the design process.
The use of a procedural programming language such as FORTRAN or C presents some inconveniences. For example, the use of global variables allows any function to change the state of the program, which makes it difficult to predict the effect of inevitable code updates. Second, plain text files are used to preserve data. But plain text files cannot enforce referential integrity. As an example to illustrate the lack of referential integrity, suppose there is a file named *Laminate.txt* that stores the laminate definition (basically the stacking sequence) and each line in the file contains lamina orientation, lamina thickness, and the path to the file where the lamina is defined, for example *Lamina.txt*. If the user deletes the file *Lamina.txt*, the file *Laminate.txt* is no longer valid because the path in the stacking sequence field points to a nonexistent file *Lamina.txt*. If referential integrity was enforced, deleting *Lamina.txt* would not be a problem.

In the last fifty years or so, computer science engineers have created a myriad of techniques to cope with inconveniences similar to the ones described above. In other engineering fields that regularly produce code to solve problems (composite materials design is an example) these techniques are not always used in a regular basis. However, this situation is beginning to change. More engineering software is incorporating OOP concepts in it and consequently users will want to know these concepts. As a matter of fact, commercial engineering software, such as Matlab®

and Abaqus®, have already incorporated some OOP concepts. Other examples are shown in [1-4].

Thus, the objective of this paper is to show the use of OOP and RDB to develop software used to perform calculations in engineering design. For the CLS design engineer, this work can be seen as an update to traditional numerical methods courses. In the experience of the authors, the time consumed in learning these software development techniques is by far compensated by the advantages that these techniques bring. It ultimately changes the approach to composite design.

1.1 Audience

This paper is aimed to non computer science engineers with background in procedural programming.

1.2 Background

1.2.1 Object oriented programming (OOP)

It is not the objective here to review the history of OOP; more thorough reviews are available in the literature [5]. However, it is worth mentioning that the first formal object oriented language was SIMULA 67, a language developed at the Norwegian Computing Center with the objective of performing physical simulations. Some other languages included this new paradigm, most notable, C++. In the early '90s, object oriented programming was already widespread and the choice for graphical user interfaces (GUIs).

Nonetheless, FORTRAN, a procedural language, dominated the scene in scientific applications, for almost fifty years. Even today it is still one of the programming languages used to teach numerical methods [6]. For other engineering applications it is common practice the use of MATLAB® which is a computer language that shares similarities with FORTRAN. Only recently MATLAB® has incorporated OOP concepts [7].

In a procedural programming language the program is written as of a set of instructions that the computer executes sequentially. In contrast, in OOP the programmer defines the classes, by defining their behavior and structure, and the interactions between classes.

A class represents things from real life, such as class *Fiber* represents fibers, class *Matrix* represents matrices and class *Lamina* represents laminae in a laminate. Because there are many kinds of fibers in the real world such as Kevlar49®, T300 carbon, and E-Glass, to name a few, each fiber is defined uniquely by the values in the fields defined in each class. The structure in the class contains data fields corresponding to the properties. For example, the Young Modulus and the Poisson's ratio define the class *Fiber*. The E-glass object, which uses the Fiber class, has a Young Modulus of 72 GPa and a Poisson's ratio of 0.22.

However, a class is not only a collection of properties, it also features behavior. The behavior of a class is defined by methods. For example, the class *Lamina,* which has the properties *Fiber*, *Matrix* and *Fiber Volume Fraction*, also has a method that calculates the reduced stiffness matrix [Q]. These methods are analogous to subroutines in procedural programming with the difference that the methods are built into the class. If there is a lamina called *MyLamina* with an E-glass fiber, epoxy matrix, and 0.4 for the fiber volume fraction, the method to calculate the matrix [Q]

of *MyLamina* is built inside the object. The method is accessed using the name of the object and a dot followed by the method's name, such as *MyLamina.Q()* and using the values stored in the properties of *MyLamina*.

There are different types of lamina based on the fibers' layout: Unidirectional, Continuous Strand Mat, Fabric, etc. The models to predict each of the lamina properties need to take into account the different layouts; therefore, there has to be a different class for each type of lamina. In spite of the different fibers' layout, all types of lamina share behavior. For example, a fabric lamina behaves similarly to a unidirectional lamina in the sense that both laminas have a reduced stiffness matrix [Q] associated with them. In OOP, these similarities can be exploited using inheritance.

Inheritance means that a class that is defined as a subtype of a parent class can inherit data fields (properties) and methods (behavior) from the parent class. The importance of inheritance is that promotes code reuse and helps when updating the code.

Unidirectional lamina and *Fabric* lamina inherit from *Lamina* the method that calculates the reduced stiffness matrix, preventing the programmer to write this function twice, one for each type of lamina. Additionally, if the project is extended to include another kind of lamina (e.g. chopped fiber) the previous definitions remain unchanged.

Polymorphism is tightly related to Inheritance. Any child class can be used in place of its parent class. In a laminate (essentially an ordered list of laminas) the program can loop through the laminae list to request the reduced stiffness matrix of each lamina regardless of its subtype. This is useful to predict the mechanical behavior of the whole laminate.

Consequently, the preceding example features abstraction. Abstraction consists of hiding irrelevant details of the particular implementation inside the object (by one of its methods). In the laminate loop example, the details of the model used to calculate the elastic properties of each type of lamina are irrelevant to the computation of the reduced stiffness matrix. All the required information that each child class requires to calculate the mechanical properties is encapsulated (built-in) into the class. This is known as encapsulation, which restricts one object from accessing another object's internal structure.

In brief, the core concepts of OOP (inheritance, polymorphism, encapsulation, and abstraction) can be exploited in CLS design.

1.2.2 Relational Databases

The need to store the information that defines the laminate system in a consistent fashion demands using databases.

A *database,* which is nothing more than an organized collection of data for one or more purposes, usually in digital form, has to meet certain requirements in the software industry: Availability, Performance, Isolation between users, Recovery from failure and disaster, Backup and Restore and Data Independence all of which ensures data consistency.

One concept that deserves special attention is data independence. It involves two different concepts: logical independence and physical independence. Logical independence means that adding more properties or relationships among entities does not affect the queries in the application program. To visualize this concept, referring back to the plain text files example, suppose now it is required to add to the laminate definition the degradation factor of each lamina. This is often useful to simulate damage evolution of a lamina in a laminate. Adding this new field in a plain-text file would cause errors in previously coded subroutines referring to the definition of the laminate because the file syntax changes would require changing all the subroutines that use the new laminate definition file syntax. Using databases solves this inconvenience since subroutines are independent of the information structure.

Physical data independence means that the details of how the information is stored in the hard drive do not affect the application queries. Database programs exploit this advantage to optimize and speed-up the queries. As a matter of fact, database programs split large files and create files for the indexes, all of this without the user noticing. This feature allows scaling of applications improving performance.

Over the years, several models have been proposed to achieve the aforementioned industry requirements. The most used approach for databases today is called *relational model*. The relational model, introduced by E. F. Codd in 1970, is a mathematical model defined in terms of predicate logic and set theory [8].

It is clear that to meet all these requirements a specialized piece of software is needed. This software is called a database management system (DBMS) and includes all the tools and services required to control the creation, maintenance, and use of databases.

1.2.3 Web application

A CLS design program can be embedded into a web application. The advantages of the web application are many. First, a web application, such as www.cadec-online.com, does not require installation by the end user. For example, in a composite design course the professor does not need to spend precious time dealing with the installation of engineering software in each computer. The professor only needs to tell the students the web address of the application. Second, centralized updating eliminates the inconvenience of periodic, time consuming updates. Also, a web application permits access to personal files from anywhere with internet access, previous work can be stored and resumed anytime, anywhere. A web application promotes peer collaboration because it acts as a centralized place where users can share and discuss information. Also, it is platform independent. Finally, a web application allows a user to access his documents from several devices without the need of synchronization, since the information is stored in the cloud.

Currently, there are an increasing number of scientific web applications [9-13] and we expect this number to grow due to the aforementioned benefits.

2. CASE STUDY

This section presents a practical implementation of the concepts outlined in the introduction section. It encompasses micromechanics, macromechanics, and shell theory applied to the case of a laminated composite depicted in Figure 1.

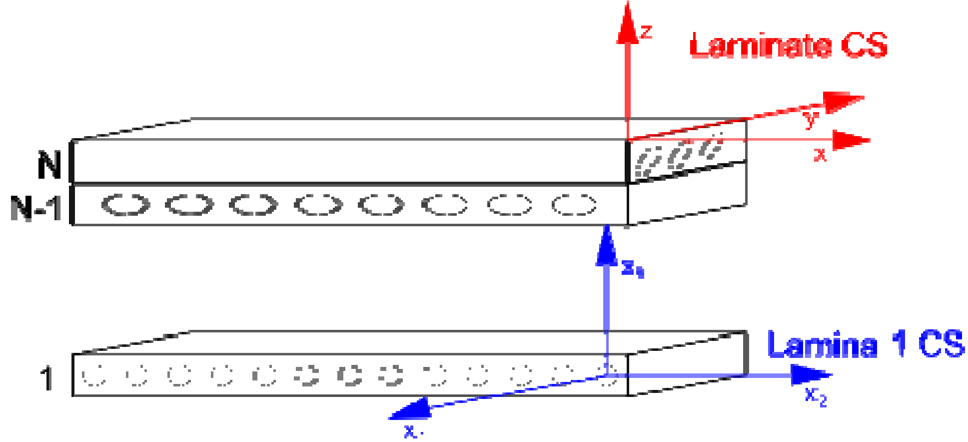

Figure 1 Laminate and Lamina Coordinate Systems

A laminate composite subjected to external loads – normal load N, moments M, and shear V – using first order laminate theory, can be modeled by

$$\begin{Bmatrix} N \\ M \end{Bmatrix} = \begin{bmatrix} A & B \\ B & D \end{bmatrix} \begin{Bmatrix} \varepsilon_0 \\ \kappa \end{Bmatrix}$$
$$\{V\} = [H]\{\gamma\} \quad [1]$$

The matrices A, B, D and H describe how the loads on the shell are found as a function of the shell's deformed geometry – described by the normal strains ε_0, the curvatures κ, and the shear strains γ. The inverse problem, i.e. finding the deformed geometry for a given loading can be retrieved inverting Equation 1. The values in matrices A, B, D and H on a composite laminate depend on the laminate stacking sequence (LSS), and the material properties of each of the laminas. The LSS, defined by the set of geometry, position, and material orientation of each lamina, allows calculating the in-plane stiffness matrix A, bending-extension coupling matrix B, the bending stiffness matrix D, and the transverse-shear stiffness matrix H. For example, A is calculated using

$$[A] = \sum_{k=1}^{N} [\overline{Q}]_k t_k \quad [2]$$

where $[\overline{Q}]_k$ is the transformed reduced stiffness matrix, which contains the stiffness in the laminate coordinate system for each of the lamina (subscript k), and t_k is the thickness (geometry) of lamina-k. Furthermore, the transformed reduced stiffness matrix comes as the result of rotating the reduced stiffness matrix [Q] of the lamina from the lamina coordinate system to the laminate coordinate system using

$$[\overline{Q}] = [T]^{-1}[Q][T]^{-T}$$

$$[T]^{-1} = \begin{bmatrix} m^2 & n^2 & -2mn \\ n^2 & m^2 & 2mn \\ mn & -mn & m^2 - n^2 \end{bmatrix} \qquad [3]$$

$$where \quad m = \cos(\theta) \quad and \quad n = \sin(\theta)$$

The rotation matrix [T], used to rotate the reduced stiffness matrix is a function of the lamina orientation defined by an angle θ, one for each lamina. The orientation of the lamina is defined as the angle that the laminate axis and the fiber direction make as depicted in Figure 1. The calculation of the reduced stiffness matrix can be done using models that depend on the lamina layout. For example, a general model for a thin transversely isotropic plane-stress lamina satisfies

$$Q = \begin{pmatrix} \frac{E_1}{\Delta} & \frac{\nu_{12} E_2}{\Delta} & 0 \\ \frac{\nu_{12} E_2}{\Delta} & \frac{E_2}{\Delta} & 0 \\ 0 & 0 & G_{12} \end{pmatrix}; \; with \; \Delta = 1 - \nu_{12}{}^2 \frac{E_2}{E_1} \qquad [4]$$

where E_1 is the Young modulus in the fiber direction, E_2 is the transverse in-plane Young modulus in the lamina, ν_{12} is the Poisson's ratio coupling the fiber and transverse directions, and G_{12} is the in-plane shear modulus. The values of E_1, E_2, ν_{12} and G_{12} can be calculated using micromechanics models such as the rule of mixtures (ROM) and the inverse rule of mixtures (IROM) [14], or other models depending on the lamina description. For example, for a unidirectional lamina (consisting of continuous fibers aligned in a single direction embedded in matrix), E_1 can be calculated using the ROM

$$E_1 = V_f E_f + (1 - V_f) E_m \qquad [5]$$

However, a continuous strand mat, which behaves as an isotropic material in the plane of the lamina ($E_1 = E_2 = E$) is calculated using

$$E = \frac{3}{8} E_1 + \frac{5}{8} E_2 \qquad [6]$$

where E_1 and E_2 are the elastic modulus in the fiber direction and transverse to it respectively for a fictitious unidirectional lamina made up with the same fiber, matrix and fiber volume fraction of the continuous strand mat.

In this section, the implementation of the LSS problem is presented using C# and Transact SQL for OOP and RDB languages, respectively. First, a description of the objects to formulate the problem in a programming scheme is presented followed by the description of the database tools used.

2.1 Object oriented implementation

2.1.1 Micromechanics

The laminas of a laminate can be of different materials with different properties. When available, experimental values are preferred. Otherwise, a model has to be used. For each kind of lamina there is a corresponding model: continuous strand mat, unidirectional, chopped strand mat, fabrics, etc. All these laminas, however, have unique elastic properties, either experimental or calculated. Regardless of how these properties are obtained or calculated, only the values of these properties can be accessed by the laminate object. This is an example of encapsulation. The reduced stiffness matrix can be calculated using those values; this is an example of inheritance.

Figure 2 shows the class diagram for this example. The modeling of a computer system can be documented using a standard called UML (Unified Modeling Language) which specifies several diagrams that helps the interaction between the team members (Architect, Designer, Programmer, Tester and so on.). [15].

Figure 1 does not follows exactly the UML standard but rather Microsoft implementation in Microsoft Visual Studio 2010®. In this diagram the classes that make up the system and the relations between them are shown. Each class is represented by a box, where the class name is on top and under it the name of the parent class or if it is an abstract class the keyword *abstract*. The properties are also shown under the class name along with the methods.

An abstract class cannot be instantiated. A lamina is an abstract concept and therefore lacks a definition of its mechanical properties. Since the different fiber layouts define the properties, only the child classes can be instantiated.

In a class diagram, an empty arrow that goes from the subclass to the parent class indicates inheritance.

The simple arrow that goes from one class to the other denotes association. For example a micromechanics lamina “has” a matrix. If it is a double arrow instead, it is a collection association “A Laminate has many LaminaInStackingSequence” (Shown in Figure 3).

A class diagram is just a graphic representation; all the classes and relationships are coded in files.

In the following discussion E_1 refers to the mechanical property of a lamina, while E1 refers to the implementation of said property in the computer language. This difference in notation is due to the restriction in variable and method names imposed by the compiler.

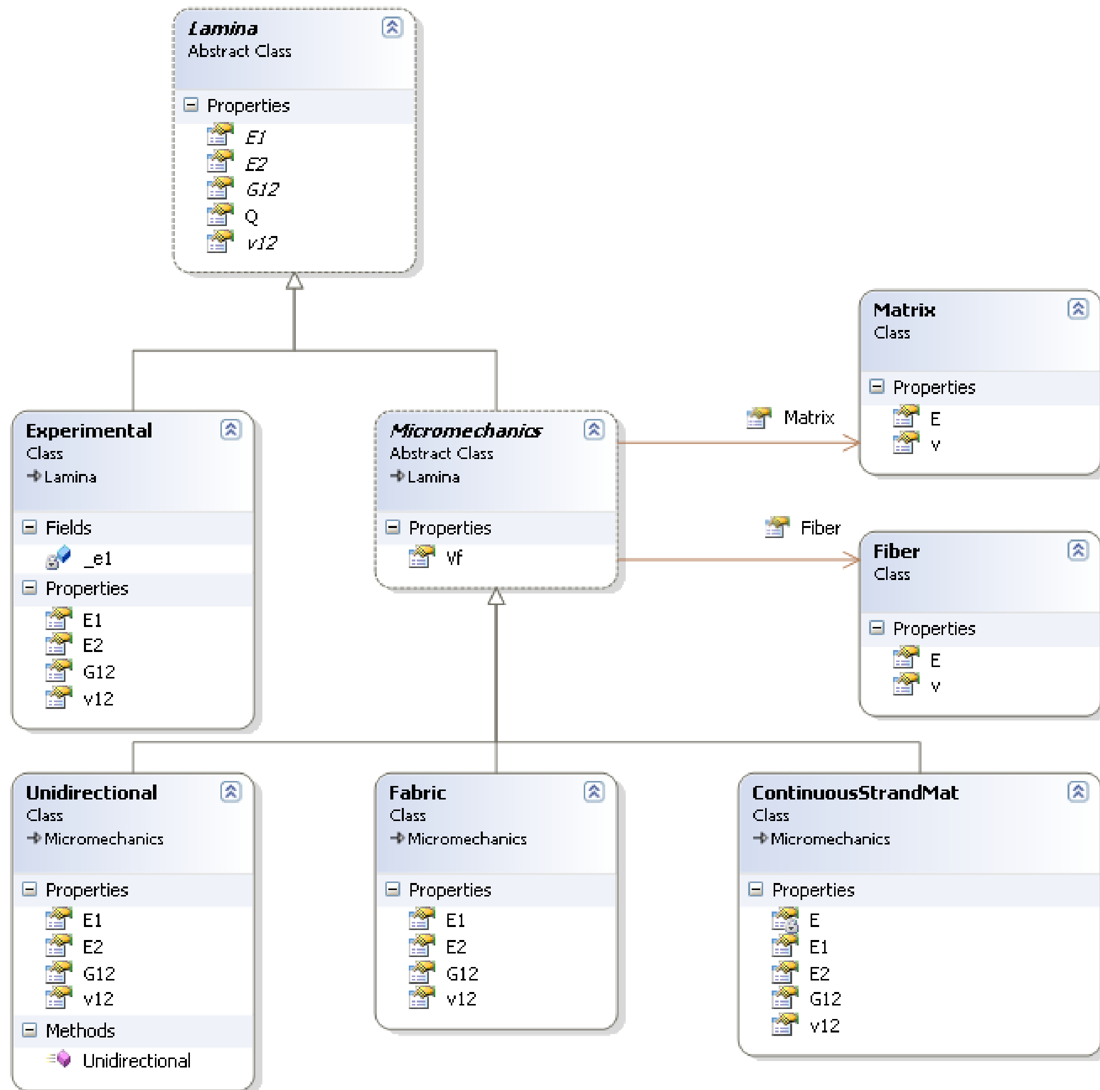

Figure 2. Class Diagram where the relationships between the classes that model the micromechanics are shown.

What follows is the implementation of the Young modulus and how each class plays a role.

2.1.1.1 Lamina

```
public abstract double E1
{
    get;
}
```

Definition of E1 as *public* means that other objects can access this value because it is a property; *abstract* means that child classes have to implement this property, and only the *get* accessor is present because the property is read only.

The keyword *get;* in conjunction with the *abstract* keyword can be thought of as a placeholder. It tells the compiler that a property called E1 has to be declared by the child classes and it is read-only.

2.1.1.2 Experimental

```
public override double E1
{
    get
    {
        return _e1;
    }
}
```

The *override* keyword tells the compiler that this property is the implementation of an abstract property. E_1 is the value provided by the user, stored in the private field _e1 which it is encapsulated in the Experimental class. It can only be modified through the methods that the Experimental class exposes.

2.1.1.3 Fabric

```
public override double E1
{
    get
    {
        return /* Result from some model */
    }
}
```

The Young modulus for a Fabric composite can be calculated from the weave pattern, but this process can take considerable time, even using a queue system.

2.1.1.4 Continuous Strand Mat

```
private double E
{
    get
    {
        Unidirectional FL = new Unidirectional(Fiber, Matrix, Vf);

        return (3/8)*FL.E1+(5/8)*FL.E2;
    }
}

public override double E1
{
    get
    {
        return E;
    }
}
```

The equation implemented to calculate E_1 for a continuous strand mat lamina is Equation 6. The *new* keyword followed by the name of the class calls a special method called a constructor, which creates a new instance of a class, in this case FL.
The *private* keyword hides the property E to the rest of the world. It is only visible to itself.

To access an object's property the dot is used, for example FL.E1 tells the compiler that the property E1 of the object FL is required.

2.1.1.5 Unidirectional

```
public override double E1
{
    get
    {
        return Vf * Fiber.E + (1 - Vf) * Matrix.E;
    }
}
```

There are several ways to calculate E_1 for a unidirectional composite. In this example Equation 5 is used.

2.1.1.6 Reduced Stiffness Matrix

Once all the properties are known, either because they are calculated with a model or provided by the user, the reduced stiffness matrix can be assembled, using the plane stress assumption (Equation 4).

```
public double[,] Q
{
    get
    {
        double Δ = 1 - (v12 * v12) * E2 / E1;
        return new double[,] { {       E1 / Δ, v12 * E2 / Δ,   0 },
                               { v12 * E2 / Δ,       E2 / Δ,   0 },
                               {            0,            0, G12 }
                             };
    }
}
```

Note that the methods E1, E2, v12 and G12 that are executed depend on the lamina type. These are methods, not global variables as in procedural programming. The details of their implementation are hidden (polymorphism and abstraction).

2.1.2 Macromechanics

The stacking sequence of a laminate is a many-to-many relationship. One laminate has many laminas, while a lamina can be used in several laminates. Each lamina in the laminate stacking sequence has its orientation and thickness. This relationship can be modeled with a link class. In this example, LaminaInStackingSequence, which stores the particulars of each association, that is, the orientation and thickness of a lamina in a particular laminate.

The class ShellLoad encapsulates the quantities that define a load state for a laminate, namely the forces per unit length along the boundary of the laminate (N), the moments per unit length (M) and the shear forces per unit length (V).

ShellStrain stores the quantities that define the strain state of a laminate; namely the mid-surface strain {ε0}, the laminate curvatures {κ} and the transverse shear strains { }.

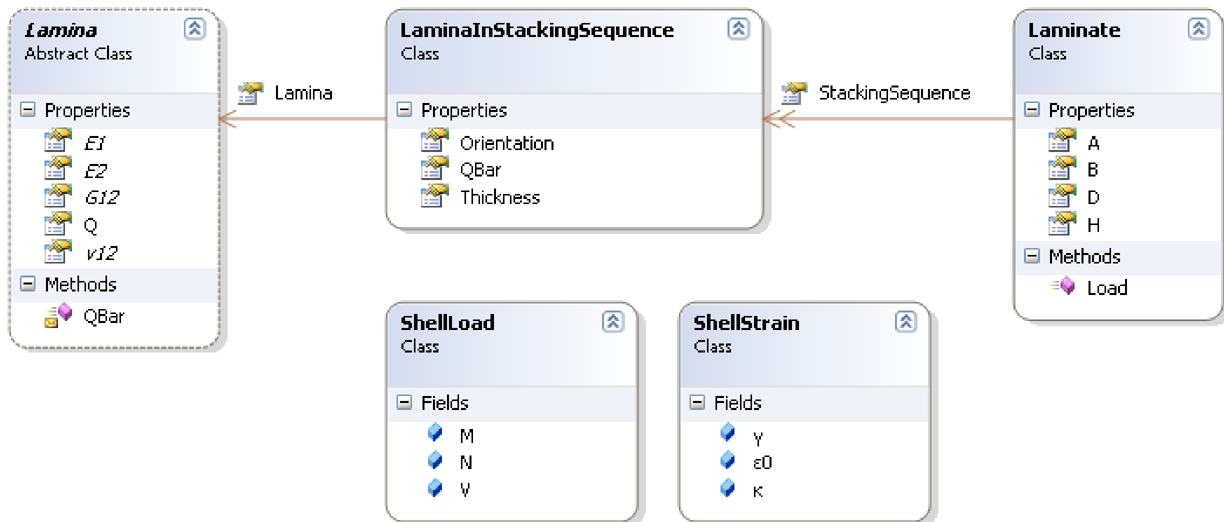

Figure 3. Class diagram for the macromechanics model.

LaminaInStackingSequence exposes a property QBar which returns the reduced stiffness matrix rotated in the particular orientation of the lamina. To accomplish this task, QBar calls the method QBar of the lamina with its orientation.

```
public double[] QBar
{
    get
    {
        return Lamina.QBar(Orientation);
    }
}
```

Lamina exposes a method called QBar, this method is responsible for rotating the Q to any coordinate system, using the transformation matrices as in Equation 3.

```
public double[,] QBar(double θ)
{
    double m = Math.Cos(θ);
    double n = Math.Cos(θ);

    double[,] invT = new double[,] {
    { m * m,  n * n,     -2 * m * n },
    { n * n,  m * m,      2 * m * n },
    { m * n, -m * n,  m * m - n * n }
    };

    return invT * Q * Math.Transpose(invT);
}
```

To calculate the laminate A matrix, the transformed reduced stiffness matrix of each lamina in the laminate coordinate system is used, (see Equation 2) as follows:

```
public double[,] A
{
    get
    {
        double[,] A = new Double[3, 3];
        foreach (LaminaInStackingSequence L in StackingSequence)
        {
            A = L.QBar*L.Thickness;
        }
        return A;
    }
}
```

The matrices B, D, and H can be calculated using a similar approach.
The tensile and shear forces per unit length along the boundary of the plate element can be calculated, along with the moments per unit length, in terms of the mid-surface strains and curvatures, using the stiffness equations (Equation 1) as follows:

```
public ShellLoad Load(ShellStrain S)
{
    ShellLoad L = new ShellLoad();

    L.N = A * S.ε0 + B * S.κ;

    L.M = B * S.ε0 + D * S.κ;

    L.V = H * L.γ;

    return L;
}
```

2.2 Relational Database

The preceding section dealt with the manipulation of objects in memory. These objects, as variables are erased as soon as the program terminates. The best way to maintain this information is to dump it in a Relational Database for later use. The term used in database jargon is CRUD, Create, Read, Update and Delete. All these operations are stored in the database as stored procedures. The language that is used in relational databases is called Standard Query Language (SQL).

SQL is a declarative language, rather than a procedural language. SQL describes what the program should do, but not how. It is up to the relational database manager system (RDMS) to find the best way to accomplish the task.

The following procedure illustrates the use of a store procedure to query the database for laminas that depend on a particular fiber. This method is useful when the user wants to delete a fiber. In this fashion, a list of dependent objects can be presented to the user followed by a request for an action from the user (as seen in Figure 4).

```
CREATE PROCEDURE Fibers_Dependencies_Lamina
       (
       @Id uniqueidentifier
```

```
        )
AS
SELECT Id, Name FROM Laminas WHERE Fiber = @Id
```

The SELECT statement returns all the laminas from the Laminas table where the Fiber field corresponds to the Id given. An analogous procedure can be used to retrieve the laminates that depend on the fiber.

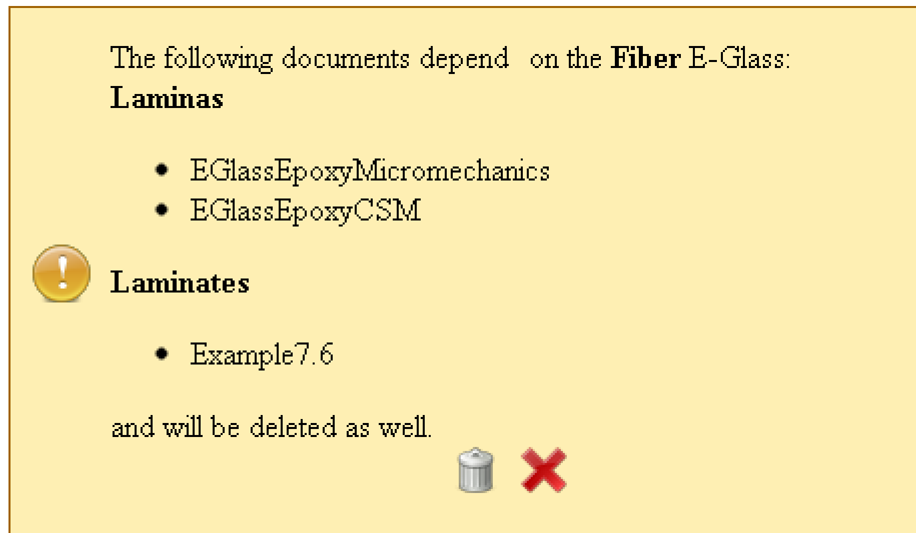

Figure 4. Example prompt asking the user to confirm deletion of the E-Glass fiber document.

3. CONCLUSIONS

The design of composite materials is a complex task that requires automation due to the amount of computations involved. The present work outlines the benefit of developing such a tool using technologies that have been traditionally associated to computer science and therefore have not been extensively used in scientific applications. Object oriented programming allows code reuse and simplifies the analysis of a system. Relational databases allow the designer to focus on needs rather than implementation and offer a full set of tools that guarantees data independence and reliability. Implementing the system on the internet allows global access and eliminates the need for user installation of updates.

4. ACKNOLEDGEMENTS

We wish to express our appreciation to the WV Higher Education Policy Commission for financial support under Award #EP509-01 WV Energy Materials Program and to Fritz Campo for his review of this paper.

5. REFERENCES

[1] Tisell, C., Orsborn, K. “System for multibody analysis based on object-relational database technology” (2000) Advances in engineering software, 31 (12), pp. 971-984

[2] Lani A., Quintino T., Kimpe D., Deconinck H., Vandewalle S., Poedts S." The COOLFluiD Framework: Design Solutions for High Performance Object Oriented Scientific Computing Software" Lecture Notes in Computer Science, 2005, Volume 3514/2005, 798-801, DOI: 10.1007/11428831_35

[3] Norton, C. D., "Object-oriented programming paradigms in scientific computing" Rensselaer Polytechnic Institute Troy, NY, USA

[4] Langtangen H. P., Munthe O. "Solving systems of partial differential equations using object-oriented programming techniques with coupled heat and fluid flow as example"

[5] Capretz, L. F., "A Brief History of the Object-Oriented Approach" Department of Electrical & Computer Engineering. Software Engineering Notes, 28(2), 1-10. 2003

[6] Dr. C.-K. Shene. "CS3911 Introduction to Numerical Methods with Fortran" 01/19/2012

http://www.csl.mtu.edu/cs3911.ck/www/Home.html

[7] MathWorks. "Compatibility with Previous Versions" 01/19/2012

http://www.mathworks.com/help/techdoc/matlab_oop/brqzfth-1.html#brqzfth-9.

[8] Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387. doi:10.1145/362384.362685.

[9] M. Weeks, "An Internet-Based Scientific Programming Environment", Proc. Digital Information and Communication Technology and its Applications (1), 2011, pp.1-12.

[10] Schwarz K., Blaha P., "Solid state calculations using WIEN2k" Computational Materials Science 28 (2003) 259–273

[11] Pirooznia, M., Gong, P., Yang, J.Y., Yang, M.Q., Perkins, E.J., Deng, "Y. ILOOP - A web application for two-channel microarray interwoven loop design". (2008) *BMC Genomics*, 9 (SUPPL. 2), art. no. S11

[12] Pöcher, C., Batrašev, O., Norbisrath U., Vainikko, E. "DougFlow— Offering Scientific Applications via Web Services" 2009 Fourth International Conference on Internet and Web Applications and Services. Venice/Mestre, Italy. May 24-28, 2009

[13] Marlon Pierce, Geoffrey Fox, "Making Scientific Applications as Web Services," Computing in Science and Engineering, vol. 6, no. 1, pp. 93-96, Jan./Feb. 2004, doi:10.1109/MCISE.2004.1255829

[14] Barbero E. J. Introduction to composite materials design-Second Edition, CRC Press, Boca Raton (FL) 2010.

[15] ISO/IEC DIS 19505-2 "Information technology -- OMG Unified Modeling Language (OMG UML) Version 2.1.2 -- Part 2: Superstructure"