

An Internet-Accessible Software Package for Modeling Viscoelastic Flow *

J. Barr von Oehsen, Eric C. Cyr, Christopher L. Cox and Brian A. Malloy

Center for Advanced Engineering Fibers and Films

Clemson University

Clemson, SC 29634

U.S.A.

{vonoehse,clcox}@ces.clemson.edu

ecyr@clemson.edu, malloy@cs.clemson.edu

Abstract

The focus of our research and development is the creation of a web-based C++ software package that numerically models viscoelastic flow in polymer processing. In order to design software that best satisfies the needs of fiber and film process modelers in industry and academia, the following capabilities must be included: (1) multi-scale simulation, from molecular through continuum levels, taking into account length and time scales which vary by several orders of magnitude, (2) accurate three dimensional simulations that can only be achieved on massively parallel computing platforms, (3) monitoring of the simulation during each of the stages using a variety of 2D and 3D visualization tools, and (4) a user-friendly web-based interface (Java Applet).

To integrate these components we have designed a highly distributed object-oriented system that exploits the power and expressivity of the Extensible Markup Language, XML, Extensible Style Language, XSL, and the XSL transformations, XSLT, to provide a uniform interface for graphical representation of data. We use Java to build the web-based graphical user interface, which allows easy interaction with our software. To make our system highly distributed we take advantage of Java servlets, which enables simple access to the code via the Internet.

We also discuss a proof-of-concept project that involves creating a web-based graphical user interface written in Java to run FORTRAN legacy code. To achieve this interoperability, we place a C-language wrapper around the FORTRAN code and then create a Java servlet that enables the Java Native Interface (JNI) to interact with the C code. The user interface runs the FORTRAN code through this servlet.

1 Introduction

The Center for Advanced Engineering Fibers and Films (CAEFF), an NSF Engineering Research Center, is developing multiscale simulations of polymer processes to improve the production of polymeric fibers and films. CAEFF industry partners will access the Center's parallel computing clusters via the internet in order to use viscoelastic flow modeling to develop new process lines and optimize existing systems. The development of this modeling package necessitates the integration of

*This work was supported primarily by the ERC program of the National Science Foundation under Award Number EEC-9731680.

several software tools to address a variety of needs, including modeling, data handling and internet access.

In this paper we present the components of our modeling package and discuss their integration to form a cohesive working code. Modeling of the viscoelastic flows encountered in polymer processing for fiber and film applications is presented in Section 2. In Section 3, the software tools which are essential for our integrative package are presented. Then in Section 4, we present examples of how these tools are used in the simulation package. First we present a GUI-driven mesh generator for the finite element solver. Second, the use of XML for data representation is discussed. In particular, we present the representation of data structures used in the finite element code. In Section 5, we draw conclusions and describe our ongoing work.

2 Polymer Process Modeling

Polymeric fiber and film-forming process lines begin at the hopper where plastic chips are loaded. The chips are melted and mixed as they pass through the screw extruder. Melt-spinning, illustrated in Figure 1, is the most common method for producing polymeric fibers. In this process the melt is pumped through a filter and then through the die or spinnerette. The fiber bundle is air-quenched and then drawn over a series of spindles or godets until being collected on a bobbin. The process of film-casting is similar through the filtration step. The spinnerette is replaced by a rectangular, or coat-hanger die, which produces a very thin wide sheet which is drawn over the chill roll, as shown in Figure 2, and on through additional drawing stages which lock-in, or set the film properties. More information on these and other types of polymer processing can be found in [2].

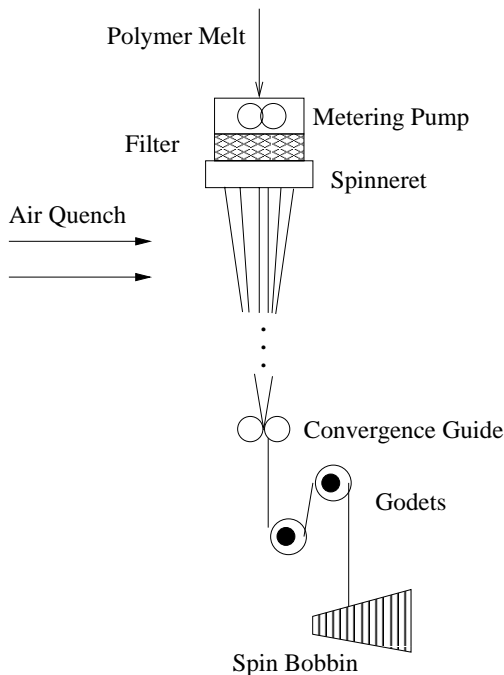


Figure 1. Fiber Melt Spinning Process

The fluid flows associated with polymer processing are classified as *viscoelastic*, because they exhibit properties of both a viscous fluid and an elastic solid. A mathematical model for the flow of a viscoelastic fluid consists of the standard momentum, mass, and energy balance equations, plus a

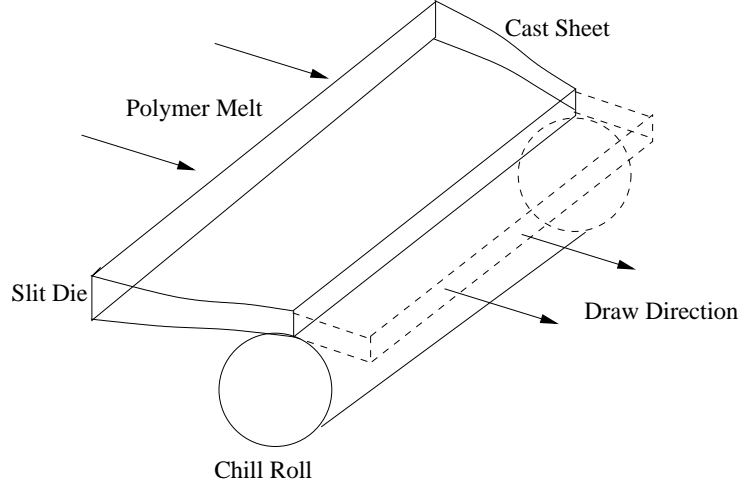


Figure 2. *Cast Film Process*

constitutive equation representing the manner in which the stress depends on the velocity gradient. The form of the latter equation distinguishes polymeric fluids from Newtonian flows (e.g. air or water). A common form of the general model for isothermal flow, with inertial terms dropped, is [1],

$$\rho \frac{\partial \mathbf{v}}{\partial t} - \nabla \cdot \boldsymbol{\tau} + \nabla p = \mathbf{f} \quad (1)$$

$$\nabla \cdot \mathbf{v} = 0 \quad (2)$$

$$\boldsymbol{\tau}_p + \lambda_1 \boldsymbol{\tau}_{p(1)} - \alpha \frac{\lambda_1}{\eta_p} \{ \boldsymbol{\tau}_p \cdot \boldsymbol{\tau}_p \} = -\eta_p \dot{\boldsymbol{\gamma}}. \quad (3)$$

In (1), ρ is the fluid density, \mathbf{v} is the velocity vector, p is pressure, $\boldsymbol{\tau}$ is the extra stress tensor, and \mathbf{f} is the forcing term vector. The incompressibility condition is imposed using (2).

The extra stress has been split according to $\boldsymbol{\tau} = \boldsymbol{\tau}_n + \boldsymbol{\tau}_p$ where $\boldsymbol{\tau}_n$, the Newtonian part, is a constant multiple of the rate-of-strain tensor, $\dot{\boldsymbol{\gamma}} = \nabla \mathbf{v} + (\nabla \mathbf{v})^T$. The polymeric part, $\boldsymbol{\tau}_p$, satisfies a nonlinear differential or integral equation. For example, (3) is known as the Giesekus model, where λ_1 , α and η_p are fitted parameters and

$$\boldsymbol{\tau}_{p(1)} = \frac{d\boldsymbol{\tau}_p}{dt} - (\nabla \mathbf{v})^T \cdot \boldsymbol{\tau}_p - \boldsymbol{\tau}_p \cdot \nabla \mathbf{v}.$$

The finite element solution is computed in three steps: mesh generation, assembly, and solution of the matrix system. Object-oriented programming techniques enable abstraction of the solution procedure to a level at which the user has great flexibility yet does not need a detailed understanding of the algorithmic or software underpinnings of the code. Options for mesh generation include the object-oriented package QMG [8], and Gambit, from Fluent Inc. [4]. An efficient, modular formulation of the assembly stage is especially important for this code because it must eventually be applicable to a range of fluid characterizations and a variety of physical settings. In other words, the code must have flexibility with respect to the form of (3) and geometry/boundary conditions. Assembly on each triangular element is carried out with the standard technique of mapping to a canonical element, using isoparametric elements to allow for curved boundaries [5]. The use of this mapping has a significant impact on data structures. A variety of solution techniques are available to the user, to best suit the needs in terms of nonlinear solvers, computing platform and postprocess visualization.

3 Software Tools

In this section, we provide background about XML, Java servlets, and the Java Native Interface.

3.1 XML

The extensible markup language, XML, has gained wide-spread popularity for a variety of software applications, including web authoring and web programming. Like its HTML counterpart, XML is derived from the standard generalized markup language, SGML. However, while HTML is a markup language used for displaying information content, XML, on the other hand, is a markup language for creating markup languages. HTML is a markup language for marking documents using tags for headings and paragraphs, whereas XML enables the creation of new tags for marking anything, such as mathematical formulas, molecular structure of chemicals, music scores and any other document. HTML limits the user to a fixed collection of tags, used primarily to describe the content that is displayed in a browser.

An XML document consists of a list of element types (tags), together with their attributes. The relationships of these elements to each other can be specified by an optional XML *Schema*. XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents. A schema is not required for a document but is recommended for document conformity. By combining an XML document with its corresponding XML Schema, an XML parser can determine the content and structure of an XML document. We have created our own fiber and film XML Schema - which we hope will become the industry standard.

For web authoring, HTML has emerged as the technology of choice for describing the content of a web page. Cascading style sheets, CSS, have emerged as the technology of choice for describing the form of an HTML document. Similarly, the extensible style language, XSL, was developed as a technology for describing the form of an XML document. An XSL style sheet provides the rules for displaying or organizing an XML document's data. XSL also provides elements that define rules for describing how to transform one XML document into another document. For example, an XML document can be transformed into an HTML document. The facet of XSL that addresses the problem of transforming XML documents is called XSL transformations, or XSLT.

3.2 Java Servlets

Java Servlet technology provides web developers with a simple, consistent mechanism for extending the functionality of a web server and for accessing existing software systems. A servlet is a dynamically loaded module that services requests from a web server. The servlet does not depend on browser compatibility because the servlet runs entirely on the server side rather than the client side. Servlets are to servers what applets are to browsers. Unlike applets, however, servlets have no graphical user interface. Java servlets provide the means to make the simulation code internet-accessible.

Another reason we chose servlets as the bridge between the internet and our code is that servlets allow us to have access to the entire family of Java APIs. Servlets can also access a library of HTTP-specific calls and receive all the benefits of the Java language, including portability, performance, and reusability. In order to run a Java servlet we use Tomcat, which is a free, open-source implementation of Java Servlet and JavaServer Pages technologies developed under the Jakarta project at the Apache Software Foundation [9].

3.3 Java Native Interface

The Java Native Interface (JNI) is the native programming interface for Java that is part of the Java Development Kit. The JNI allows Java code that runs within a Java Virtual Machine (VM) to operate with applications and libraries written in other languages, such as C and C++. Programmers use the JNI to write native methods to handle those situations when an application cannot be written entirely in the Java programming language. Using native methods and the JNI is ideal if you already have a library or application written in another programming language (FORTRAN, C, C++ ...) that you wish to make accessible to Java applications - in our case a Java applet.

Native methods can also easily call Java methods. The native method, using the JNI framework, can call the existing Java method, pass it the required parameters, and get the results back when the method completes. For example, in our servlet we use the *popen* function to open a process. The return value from *popen* is a normal standard I/O stream which we pass to a Java PipedWriter write method (see Figure 3) .

```
(1) jclass class_PipedWriter = env->GetObjectClass(p_writer);
(2) jmethodID id_write = env->GetMethodID(class_PipedWriter,"write","(I)V");
(3) jmethodID id_flush = env->GetMethodID(class_PipedWriter,"flush","()V");
(4) cmdlen = env->GetStringUTFLength(jcmd);
(5) cmd = (char*) malloc(sizeof(char)*cmdlen);
(6) if(cmd != NULL) {
(7)   strcpy(cmd,env->GetStringUTFChars(jcmd,0));
(8)   if((ptr = popen(cmd,"r")) != NULL) {
(9)     setbuf(ptr,NULL);
(10)    while((c =getc(ptr)) != EOF) {
(11)      env->CallVoidMethod(p_writer,id_write,(jint)c);
(12)      fflush(ptr);
(13)      env->CallVoidMethod(p_writer,id_flush);
(14)    }
(15)  }
(16) }
```

Figure 3. Native Interfacing. On line (1) we get a handle on the Java PipedWriter. On lines (2) and (3) we access the write and flush methods of the PipedWriter class. Line (8) shows how we pass a command to popen and then on line (11) we write the standard I/O to the PipedWriter write method.

4 Example Applications

In this section, we present an overview of the viscoelastic flow simulation package, then present particular examples from the package illustrating web-access to the code and the representation of data using XML.

4.1 Overview of the Software Package

Figure 4 summarizes our system to provide access to the the simulation code via the web. In order to achieve this we make use of JNI and Java servlets.

The box on the left of the diagram represents the client side of our application where a user will interact with our code using an applet embedded within a web-browser.

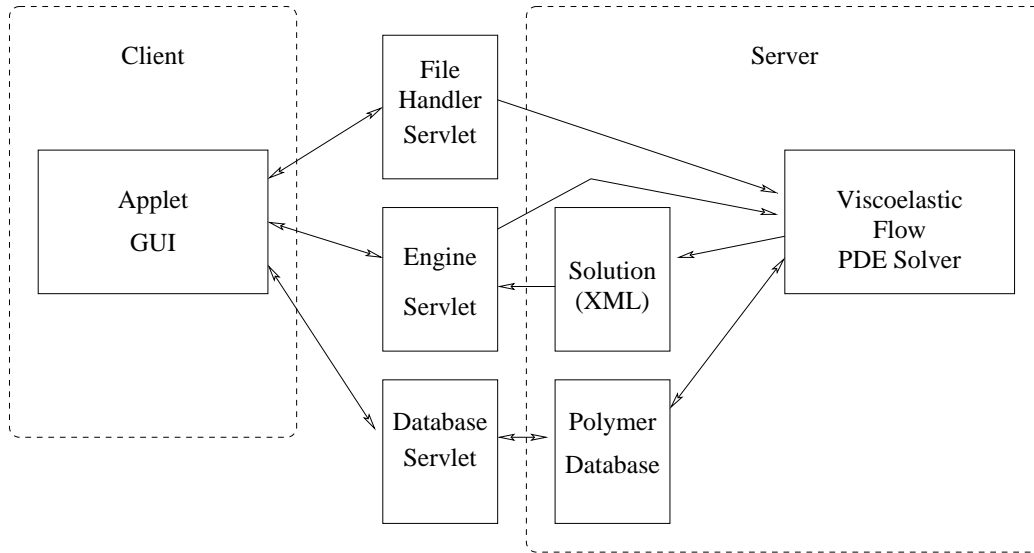


Figure 4. *System Overview.* This figure illustrates the flow of information and the key elements in our internet-accessible package for modeling Viscoelastic Flow.

4.2 Mesh Generation Graphical User Interface

Figure 5 contains a snapshot of a GUI-driven mesh generator. For this example, the mesh generation software consisted of routines from a FORTRAN legacy code. The user uses the GUI to build the mesh by specifying the shape of the domain and the vertices of each rectangular (in this case) or triangular element. At another input stage, the user will set polymer-specific parameters, boundary conditions, and any other important information related to solving a visco-elastic problem. This example serves as proof-of-concept for a web-accessible simulation package.

4.3 Polymer Database

The database servlet allows the applet to query the Polymer Database (after the user has selected a polymer) for physical properties so that the material parameters needed for the simulation code can be set. The database, which is XML-based, serves a dual purpose. Not only do we use it to provide input parameters for the simulation, but because of the power of XML and the tools associated with XML, we have a web-accessible repository of polymer data. A snapshot of the GUI for the database is shown in Figure 6.

4.4 XML Representation of Solution Data

Once the user has set up the problem, the information is then passed to the File Handler Servlet, located in the center of Figure 4, which creates all the files necessary for running the simulation. The Engine servlet, which uses JNI, takes on the task of executing the mesh generator, assembly, solution, and translation of the solution, written in XML. Figure 7 illustrates our use of XML to describe the finite element solution of a viscous flow problem.

The finite element XML Schema includes XML tags to describe nodes and elements used in finite element analysis. These tags are represented by `NODES` and `ELEMENTS` on lines 5 and 20 in the figure. The children of `NODES` are one or more `NODE` elements, which are composed of coordinate vectors,

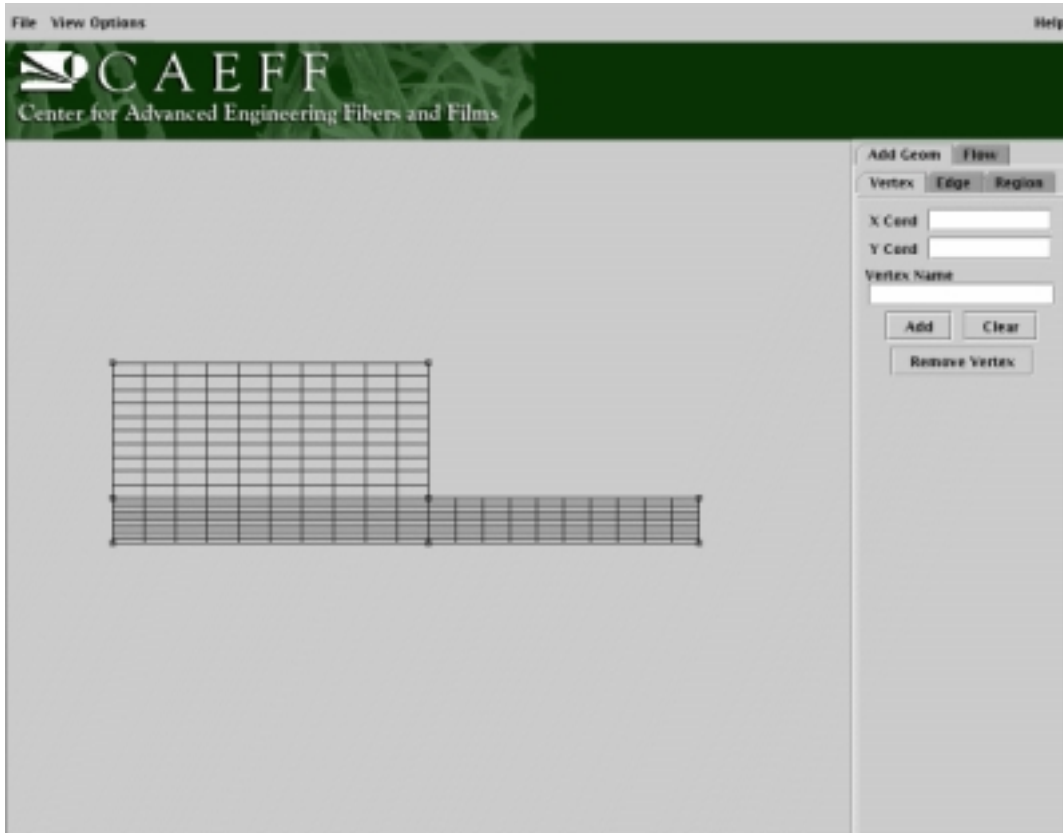


Figure 5. *The Graphical User Interface Applet for Generating Meshes. A snapshot of our graphical user interface.*

velocity vectors and pressure, illustrated on lines 6 through 18 in the figure. We have abbreviated the XML representation but in the actual system these XML elements completely describe our finite element solution. Unfortunately, the solution is almost meaningless if one can not use other tools to view the results. Here lies the problem - different tools require different file formats. It is precisely this reason that flexibility in translation is absolutely essential.

Depending on the solution of the linear system and the visualization tool that we use, we sometimes need to create more than one formatted file. By exploiting the XML tools made available by the Apache XML project (Xerces and Xalan), [9], the Mozilla Organization (Rhino - JavaScript for Java), [6], and the Simple API for XML (SAX), [7], we have designed our system so that most decisions of this type are transparent to the user. For example, we accomplish the transparency of creating multiple formatted files by first parsing the XML file to check tags and then set certain values dependent upon the parsed outcome. Once completed, our system then reads in a given XSLT style sheet and substitutes our set values into parameterized conditional statements. The translation only takes place where the conditional statements are true (see Figure 8).

As shown in Figure 4, once the system of equations has been solved, the results are output into an XML file. The box to the left in the server box represents these results. The XML representation of the solution, together with an XSLT style sheet, becomes input to a translator. An XSLT style sheet is tailored to a particular viewing tool and is used to translate the XML representation into a file for that particular tool. Finally, the translated file is used as input to the tool and can be viewed by the user in a variety of file formats.

Using XML has several advantages: (1) ease of maintenance and less code bloat (we do not have

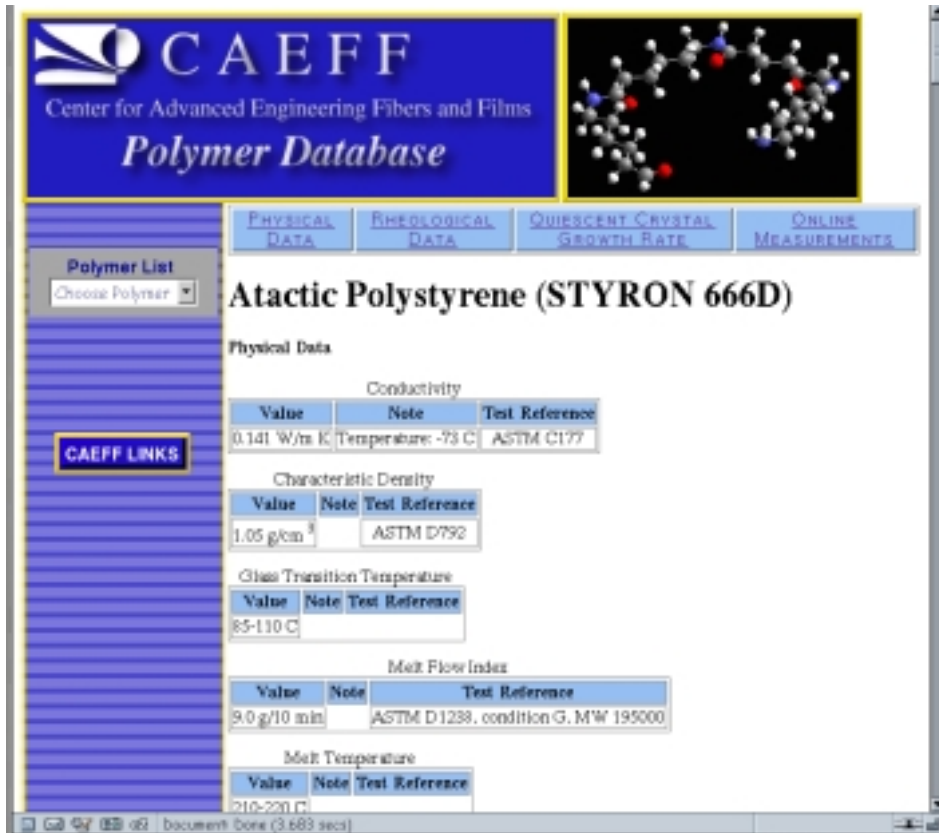


Figure 6. Database Snapshot. The polymer database can be used on the web or queried directly by the system.

to hard code different formats into our system), (2) the user does not need to wait for a software upgrade just so that he or she can use their favorite visualization/computer algebra system tool, (3) writing an XSLT style sheet is straightforward, (4) the user can save output in multiple formats, and (5) if the user decides later that he or she would rather use a different file format after the fact, then (as long as the user has the proper style sheet) getting a new translation can easily be managed without having to run the finite element code from the start.

5 Conclusions and Future Work

In this paper, we have described the development of an internet accessible software package that numerically models viscoelastic flow in polymer processing. The combination of the software tools XML, Java Servlets, and the Java Native Interface with a finite element solver, (either legacy code or the object-oriented code which we're writing) produces a powerful, user-friendly design package which polymer scientists can use to develop better fiber and film production facilities. The package is being written with sufficient generality so that extension to other physical systems of a similar nature will not be difficult.

Next steps in the effort include:

- enabling the finite element code to access material parameters directly from the polymer database,

```

(1) <?xml version="1.0"? >
(2) < FINITE_ELEMENT
(3)   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
(4)   xsi:noNamespaceSchemaLocation="finite_element.xsd" >
(5)   < NODES NUMBER_OF_NODES="1681" >
(6)     < NODE GLOBAL_NODE_NUMBER="0" >
(7)       < VECTOR DIM="2" >
(8)         < COORD > 0 </COORD >
(9)         < COORD > 0 </COORD >
(10)       </VECTOR >
(11)       < VELOCITY >
(12)       < VECTOR DIM="2" >
(13)         < CRD > 0 </CRD >
(14)         < CRD > 0 </CRD >
(15)       </VECTOR >
(16)       </VELOCITY >
(17)       < PRESSURE >-104.249</PRESSURE >
(18)     </NODE >
(19)   </NODES >
(20)   < ELEMENTS NUMBER_OF_ELEMENTS="800" >
(21)     < ELEMENT GEOMETRY="TRIANGLE_SIX_POINT"
(22)       GLOBAL_ELEMENT_NUMBER="0" >
(23)       < VECTOR DIM="6" >
(24)         < COORD > 0 </COORD >
(25)         < COORD > 2 </COORD >
(26)         < COORD > 84 </COORD >
(27)         < COORD > 1 </COORD >
(28)         < COORD > 43 </COORD >
(29)         < COORD > 42 </COORD >
(30)       </VECTOR >
(31)     </ELEMENT >
(32)   </ELEMENTS >
(33) </FINITE_ELEMENT >

```

Figure 7. XML Example. This figure illustrates our use of XML to describe the data structures that we use in our modeling of a viscous flow problem. Our approach can be applied to most mathematical models where data needs some form of visualization.

- building password protection into the user interface to allow industry users to privately model proprietary material processes, and
- a batch submission system which allows users to initiate CPU time-intensive simulations and be notified once the simulation is completed.
- incorporating mathematical control and optimization for determination of process conditions which produce film and fibers with desired final properties.

```
( 1) < xsl:when test="$param='velocity_pressure'" >
( 2) variables="x","y","u","v","p"
( 3) < /xsl:when >
( 4) < xsl:when test="$param='no_pressure'" >
( 5) variables="x","y","u","v"
( 6) < /xsl:when >
( 7) < xsl:when test="$param='no_velocity'" >
( 8) variables="x","y","p"
( 9) < /xsl:when >
```

Figure 8. Parameterized Conditional Statements. This figure summarizes how one uses parameterized conditional statements within an XSLT style sheet. If the variable "param" is set to "velocity_pressure," then only the text on line (2) will be used in the translation. Likewise for the other conditional statements.

References

- [1] R. BIRD, R. ARMSTRONG AND HASSAGER, *Dynamics of Polymeric Liquids, Volume One*, Wiley, 1987.
- [2] D.G. BAIRD AND D.I. COLLIAS, *Polymer Processing, Principles and Design*, Wiley, 1995.
- [3] J. B. VON OEHSEN AND R. C. JENKINS AND C. L. COX AND B. A. MALLOY, *Exploiting XML to Provide a Uniform Interface for Graphical Representation of Finite Element Analysis*, Proceedings of the International Conference on Computer and Information Systems", 181–185, Oct 2001.
- [4] FLUENT, *Fluent Flow Modeling Software*, 2001, <http://www.fluent.com/>.
- [5] C. JOHNSON, *Numerical Solution of Partial Differential Equations by the Finite Element Method*, Cambridge University Press, 1992.
- [6] MOZILLA ORGANIZATION, *Rhino: JavaScript for Java*, 2001, <http://www.mozilla.org/>.
- [7] OASIS, *SAX developed collaboratively by the members of the XML-DEV mailing list*, 2001, <http://www.oasis-open.org/>.
- [8] QMG, *2.0 mesh generation software*, Cornell University, www.cs.cornell.edu/home/vavasis/qmg2.0/qmg2.0_home.html, 1999.
- [9] THE XML APACHE ORGANIZATION, *The Apache XML Project*, Open Source Consortium, 2001, <http://xml.apache.org/>.