Exploiting XML to Provide a Uniform Interface for Graphical Representation of Data

J. Barr von Oehsen, Richard C. Jenkins,
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

Abstract

In this paper, we describe a technique for representing the results of computational models by exploiting the power and expressivity of the Extensible Markup Language, XML. This uniform representation facilitates sharing of information on the Internet, as well as viewing by many tools, such as $Matlab^{©}$, $Maple^{©}$, $Tecplot^{©}$, or a web browser. We allow the user of the application to choose the desired file format. We then use XML, together with an XSLT style sheet for the particular application, to produce a document viewable by the tool. An advantage of using XML is that if there is an application that is not currently supported or if a new application is available, we simply create a new style sheet without having to modify the code that generates the data.

1 Introduction

The compelling attraction of the Internet is that it has made gigabytes of information widely available to people of all walks of life. Most of this information is packaged in a user-friendly, easy-to-view format. Furthermore, the Internet is a tremendous vehicle that permits mathematicians and engineers to share technical data. However, most mathematical and engineering software applications produce files in a proprietary format, consumable only by a particular tool, or a limited set of tools. In order for these applications to accommodate other file formats, the application must be modified. Moreover, users of different applications must either wait for the modification, or try to arrive at an interoperable solution on their own.

In this paper, we describe a technique for representing the results of computational models by exploiting the power and expressivity of the Extensible Markup Language, XML[3, 5]. This uniform representation facilitates sharing of information on the Internet, as well as viewing by many tools, such as $Matlab^{©}$, $Maple^{©}$, $Tecplot^{©}$, or a web browser. We allow the user of the application to choose the desired file format. We then use XML, together with an XSLT style sheet for the particular application, to produce a document viewable by the tool. An advantage of using XML is that if there is an application that is not currently supported or if a new application is available, we simply create a new style sheet without having to modify the code that generates the data.

We have applied our technique to a case study that computes the results of a system of linear equations related to the finite element solution of a viscoelastic flow problem; however, the technique can be applied to most computational models where data needs some form of visualization. The output of the computational model is an XML representation of the data that, 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 the next section, we present background about XML and finite element modeling of viscoelastic flow. In Section 3, we describe the design of our system. Finally, in Section 4 we draw conclusions and describe our ongoing efforts to exploit XML to facilitate visualization of the results of our model and to incorporate a database into our system.

2 Background

In this section, we provide background about XML and finite element modeling of viscoelastic flow. We include an example that illustrates our use of XML and XSLT style sheets.

2.1 XML

The extensible markup language, XML, has become one of the hottest concepts in computer science, web authoring and web programming. Like its HTML counterpart, XML is derived from the standard generalized markup language, SGML. However, 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.

Many applications for XML have been created for structuring data for various disciplines. In Mathematics, the Mathematical markup language, MathML, has been developed for describing mathematics notation. Previous language descriptions for mathematics notations have used software packages such as TeX and LaTeX; MathML enables techniques for describing mathematics notations for both databases and the web. Other example applications of XML include the chemical markup language, CML, the speech markup language, SpeechML, and the synchronized multimedia interface language, SMIL.

An XML document consists of a list of element types, together with their attributes. The relationships of these elements, to each other, can be specified by an optional document type definition (DTD). DTD's for a document describe the grammatical rules for a document. A DTD is not required for a document but is recommended for document conformity. By combining an XML document with its corresponding DTD, an XML parser can determine the content and structure of an XML document.

For web authoring, HTML has emerged as the technology of choice for describing the content of an HTML document; cascading style sheets, CSS, has 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 XML 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.

Figure 1 illustrates our use of XML to describe the modeling of a finite element mesh. The data structures in the model include XML elements to describe zones and nodes represented by `ZONE` and `NODE` elements on lines 5 and 7 in the figure. A zone is composed of nodes, which are composed of coordinate vectors, velocity vectors and pressure, illustrated on lines 10, 16 and 21 in the figure. We have abbreviated the XML representation but in the actual system these XML elements completely describe the data structures used in our finite element solution of a viscoelastic flow problem.

2.2 Finite Element Modeling of Viscoelastic Flow

The immediate application for the technique that we describe is an object-oriented software package being developed to numerically model viscoelastic flow occurring in polymer processing. This package will have widespread application in fibers and films industry and research, and will serve as a starting point for our future finite element system development. However, the approach that we describe in this paper can be applied

```
< ?xml version=”1.0” standalone=”no”? >
<!DOCTYPEFiniteElementSYSTEM
    ”finite_element.dtd” >
    < FiniteElement >
    < ZONE NUMBER_OF_NODES = ”1”
      HEIGHT = ”5”WIDTH = ”2” >
    < NODES >
    < NODE GLOBAL_NUMBER = ”3” >
    < COORDS >
     < VECTOR DIM = ”2” >
     < CRD > 0.05 < /CRD >
     < CRD > 0 < /CRD >
     < /VECTOR >
     < /COORDS >
     < VELOCITY >
     < VECTOR DIM = ”2” >
     < CRD > 0 < /CRD >
     < CRD > 0 < /CRD >
     < /VECTOR >
     < /VELOCITY >
     < PRESSURE >-104.249< /PRESSURE >
     < /NODE >
     < /ZONE >
     < /FiniteElement >
```

Figure 1. *XML Example*. **This figure illustrates our use of XML to describe the data structures that we use in our modeling of a viscoelastic flow problem. Our approach can be applied to most mathematical models where data needs some form of visualization.**

to any system where the output will be viewed in multiple formats or by multiple computer algebra systems or visualization tools.

In our system, the main components correspond to the algorithms needed for a finite element solution of partial differential equations. In particular, our system incorporates mesh generation, piecewise polynomial interpolation, quadrature, linear and nonlinear system solvers. 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.

3 Design of the System

We take the results of a system of linear equations related to a viscoelastic finite element problem and output them as an XML file defined by our finite element document type definition (DTD). We can pass this XML file along with an XSLT style sheet, to our translator. The translator then produces a file formatted for a particular visualization tool/computer algebra system such as Java 3D, Tecplot, MatLab, Maple, Mathematica or other data formats such as the Hierarchical Data Format (HDF).

3.1 Overview of previous approaches

Current designs, such as Deal.II, OFELI, and Fluent (Gambit, Polydata, and Polyflow), hard code the supported data formats within their system[4, 2, 9], so that adding an unsupported tool to these systems requires either the modification of existing code, or additional code to support the new tool. For example, with Tecplot, a new module must be created using the Tecplot Add-on Developers Kit (ADK)[1]. Thus, Tecplot places the burden of building the added module on the system user, so that adding an unsupported tool can be rather frustrating to the user.

```
( 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 2. *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.**

3.2 Design of our system

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), the Mozilla Organization (Rhino - JavaScript for Java), and the Simple API for XML (SAX) we have designed our system so that most decisions of this type are transparent to the user[6, 4, 7, 8]. 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 2). Another feature of these tools that we use is the ability to embed JavaScript within an XSLT style sheet (see Figure 3). Using JavaScript enabled us to write functions to aid in the translation process.

```
< lxslt:script lang="javascript" >
    function change_index(a,b) {
      var n = parseInt(a);
      var s = parseInt(b);
      var r = n % (2*s + 1);
      var p = parseInt(n/(2*s +1));
      if(r == 0) {
         r = 2*s + 1;
         p–;
      }
       return parseInt((r + p*(s + 1) + 1)/2);
   }
< /lxslt:script >
```

Figure 3. *Embedding JavaScript within XSLT.* **This figure summarizes how one embeds JavaScript within XSLT. We needed a function that allowed us to make changes to indexing on the fly during the translation process. Being able to embed JavaScript within the XSLT style sheet greatly simplified the task.**

Figure 4 represents an overview of our approach. The rectangle on the left of the figure represents the results of a numerical model or computation (in our case the results of a system of linear equations related to a viscoelastic finite element problem). The output of the computational model is an XML representation of the data that, together with an XSLT style sheet, becomes input to a translator, represented by the rectangle in the middle of Figure 4. 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 the figure, we illustrate four commonly used file formats for viewing data: Tecplot, Matlab, HTML and HDF.

4 Conclusions and Future Work

In this paper, we have described our approach to representing the results of computational models by exploiting the power and expressivity of the Extensible Markup Language, XML[3, 5]. This uniform representation facilitates sharing of information on the Internet, as well as viewing by many tools, such as $Matlab^{©}$, $Maple^{©}$, $Tecplot^{©}$, or a web browser. We allow the user of the application to choose the desired file format. Our approach provides easy extension of our modeling results to other viewing tools as well as easy incorporation of a database into our system.

Our approach has several advantages: (1) ease of maintenance and less code bloat (we do not have 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 not very difficult, (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.

Our ongoing work includes the elimination of the DTD aspect of XML, in favor of a schema. Schema's are more flexible, powerful and more precise. With a schema one can not only capture XML syntax of a document, but also the actual data type of each elements content. Moreover a schema captures the spirit of XML more fully. We plan to incorporate an XML database into the system to obviate recomputation of

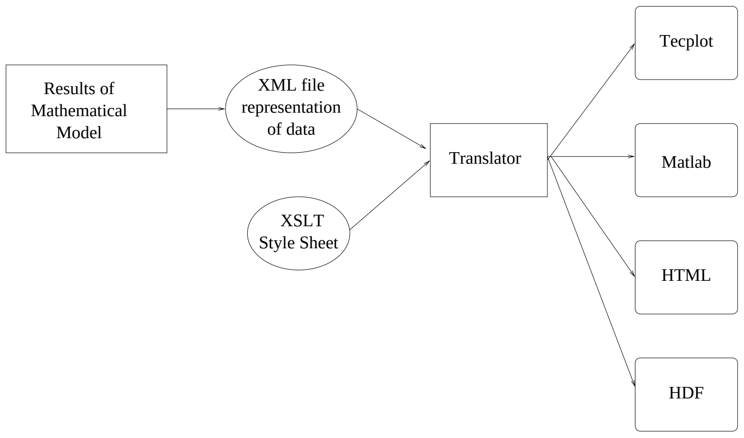
Results of Mathematical Model
XML file representation of data
XSLT Style Sheet
Translator
Tecplot
Matlab
HTML
HDF

Figure 4. *System Overview*. **This figure provides an overview of the application of our technique. The Mathematical model is illustrated on the left of the figure as a rectangle. Output from the model is an XML file which, together with an XSLT style sheet, is input to a translator. The translator produces a file readable by a number of tools; the tools listed in the figure are Tecplot, Matlab, HTML and HDF.**

certain equations and information. Finally, we plan to incorporate the XSLTC compiler into the system so that we can precompile the XSLT style sheets to make translation more efficient.

5 Acknowledgements

This work was supported in part by the ERC program of the National Science Foundation, under award number EEC-9731680.

References

[1] Amtec Engineering Incorporated. *Tecplot, Version 9.0*. http://www.amtec.com/, 2001.

[2] Deal.II. *deal.II: A Finite Element Differential Equations Analysis Library*. http://gaia.iwr.uni-heidelberg.de/ deal/, 2001.

[3] Deitel, Deltel, Nieto, Lin, and Sadhu. *XML How to Program*. Addison-Wesley, 2001.

[4] Fluent. *Fluent Flow Modeling Software*. http://www.fluent.com/, 2001.

[5] S. Holzner. *Inside XML*. New Riders Publishing, 2001.

[6] Oasis. *SAX developed collaboratively by the members of the XML-DEV mailing list*. http://www.oasis-open.org/, 2001.

[7] M. Organization. *Rhino: JavaScript for Java*. http://www.mozilla.org/, 2001.

[8] The XML Apache Organization. *The Apache XML Project*. http://xml.apache.org/, 2001.

[9] R. Touzani. *Oefeli Object Finite Element Library*. http://www.fluent.com/, 2001.