The Benefits and Practicalities of Using Extensible Markup Language (XML) for the Interfacing and Control of Object-Oriented Simulations

Good J.

Lincoln Ventures Limited, Lincoln, New Zealand. Email: goodj@lincoln.ac.nz

Keywords: *XML; object-oriented; simulation; XSDL.*

EXTENDED ABSTRACT

Traditionally simulation software has been custom built for its particular purpose with input and output file formats as well as control interfaces being unique to the simulation. This leads to considerable difficulties in preparing simulation data, utilising output data in other applications and using the simulation as a component of larger simulations.

This paper presents an alternative method for the interfacing and control of simulations. This method uses an industry standard language for the implementation of interfaces and the representation of data and metadata. The use of this technique facilitates the interoperability of simulation components and provides flexibility for the development of user interfaces. It also greatly simplifies connection to external data sources such as database applications and GIS systems.

With the advent of the Extensible Markup Language (XML) standard for data interchange, web services and the advent of grid computing significant opportunities exist to overcome these problems. XML is a self-describing data interchange language that not only contains data but also contains information about the structure and nature of the data. The vast majority of desktop applications are now able to view and manipulate XML data with the XML standard being consistent across languages and operating systems.

The XML Schema Definition Language (XSDL) also provides a platform independent method for ensuring that XML data is appropriately structured and valid. The benefits of these technologies include easier production and validation of input data, flexible analysis of output data, enabling the reuse of components, and the remote or parallelised deployment of components on a variety of platforms. XSDL schemas enable the precise structure of XML documents to be specified. They have considerable advantages over the earlier technology of Document Type Definitions (DTDs).

This paper illustrates the benefits of using XML and XSDL using the example of the FarmSim application produced as part of the Integrated Research for Aquifer Protection (IRAP) programme. This has provided great flexibility in handling the large quantities of data associated with the simulation.

The implementation of this technology has enabled an external company to develop a user interface for the paddock-level component that specifically addresses their operational requirements without any knowledge of the internal operation of simulation component. Should the user simulation component change, the user interface automatically incorporates the changes by utilising the data in the XML and XSDL structures.

The reading of XML data documents and the conversion of this data into a structure of simulation entity objects in memory usually involves the writing of a complex data reading and object instantiation component. However, the System.Xml namespace within the Microsoft .NET framework enables the simulation entities themselves to carry out this operation using features of the XML Document Object Model (DOM) and XPath queries. This paper describes the implementation of this method in Microsoft Visual C#.NET. This technique enables the simulation classes themselves to accept XML documents and use the data contained in these documents to set their parameters and state variables. Where the objects are composites the appropriate XML nodes are extracted using XPath queries and directed to the constructors of the component objects.

1 INTRODUCTION

The development of XML and its adoption as a formal recommendation of the World Wide Web Consortium (W3C 2004a) provides significant opportunities to improve data exchange and interoperability for modelling and simulation software projects (Argent 2004; Fishwick 2002; Jolma and Rizzoli 2003; Kokkonen *et al.* 2003).

There are many advantages of utilising XML in simulations. XML is a language that is able to represent not only data itself, but the nature and structure of that data. This combination of data and metadata in the one file means XML has considerable advantages over traditional data formats. As XML files are plain text and comply with a globally accepted standard, the files are very useful in almost any application where interoperability is required. XML is a cross-platform format that is not exclusive to any particular operating system or development platform. For example, common desktop software tools such as the Microsoft Office and OpenOffice.org (OpenOffice.org 2005) produce and utilise XML files. Almost all commercial databases and GIS products include XML support as a standard feature.

XML is fundamental to many internet technologies such and web services and grid computing. Software development platforms such as Sun Microsystems J2SE/J2EE framework and the Microsoft .NET framework all include extensive support for XML. The examples in this paper are presented using Microsoft Visual C# running on the Microsoft .NET framework. The System.Xml namespace in this framework provides considerable functionality for reading, creating and manipulating data in XML format.

The use of XML for data representation and interfacing within simulations greatly assists in the interoperability of components. The work by Kokkonen *et al.* (2003) illustrates how XML can be used to create an interface between a simulation and meteorological data.

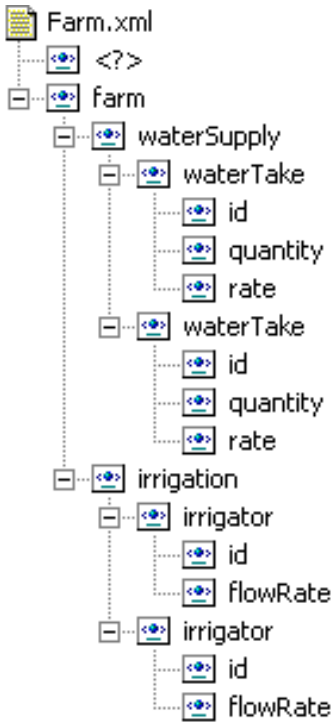
The present paper describes how XML data can be supplied to an object-oriented simulation component running in the Microsoft .NET framework. The method for validating the content of input XML using XML schemas is explained. XML schemas are a recent addition to the W3C XML recommendations (W3C 2004b) and provide considerable advantages over the earlier method of using Document Type Definitions (DTDs). A method for using the XML data to cause simulation entities to instantiate themselves is also described.

2 DATA IN XML FORM

XML offers a rich set of tools for the representation and exchange of data within modelling and simulation projects. Figure 1 shows a simplified XML data file and associated tree-view visualisation. The file contains data that can be used to construct the simplified object-oriented simulation shown in Figure 2. This Unified Modelling Language (UML) diagram shows a greatly simplified object structure based on that

Figure 1 – Farm XML File and Tree-view Visualisation

```
<?xml version="1.0" encoding="utf-8" ?>
<farm>
   <waterSupply>
      <waterTake>
         <id>A</id>
         <quantity>28000</quantity>
         <rate>100</rate>
      </waterTake>
      <waterTake>
         <id>B</id>
         <quantity>30000</quantity>
         <rate>100</rate>
      </waterTake>
   </waterSupply>
   <irrigation>
      <irrigator>
         <id>A</id>
         <flowRate>40</flowRate>
      </irrigator>
      <irrigator>
         <id>B</id>
         <flowRate>39</flowRate>
      </irrigator>
   </irrigation>
</farm>
```



used in the IRAP FarmSim simulation.

FarmSim simulates the operation of cropping and pasture grazing (sheep and dairy) farms under differing management regimes and is primarily concerned with determining time-varying effect of these land uses on drainage flux and nitrate leaching from the vadose zone into the underlying groundwater. FarmSim is described in more detail in a companion paper at MODSIM05 (Good and Bright 2005). While completing the initial design for this application, it was identified that the application needed interoperability and interfacing functionality that was outside the capability available from traditional methods of data representation such as delimited text files. XML was chosen as it provides the necessary functionality and has become a widely adopted standard for data representation and exchange.

The tree-view visualisation of the XML file shown in Figure 1 clearly shows the ability of XML to represent a complicated nested data structure in a simple text format. In the example, the "farm" element is comprised of a "waterSupply" and "irrigation" element. The "waterSupply" element contains two "waterTake" elements and they have three elements that record data about the waterTakes.

In this example, the "waterSupply" represents the total water available to the whole farm, while the "waterTake" is an individual right to take a certain volume of water from a particular source over a defined irrigation season. Likewise, the "irrigation" element represents the irrigation system for the whole farm. This contains of two "irrigator" elements that record the properties of the individual machines that carry out the irrigation. Both these examples have been considerably simplified for this illustration.

The use of XML within the FarmSim project has enabled the decoupling of the simulation controller from the user interfaces that are developed for the project. The simulation operates as an independent component receiving data and simulation parameters in XML form and returning the results of the simulation as XML data. This has enabled a separate research agency (Aqualinc) to develop an independent user interface for the paddock-scale version of FarmSim. To achieve this, all that was required was for the Aqualinc application to produce appropriate input XML data and provide a means of browsing and visualising the output XML data.

The paddock-scale version of FarmSim was created to enable the development and testing of the paddock model that would later be the fundamental building block of the farm-scale simulation (Good and Bright 2005). The up-scaling of this generic user interface to work with farm-scale FarmSim should be simple, due to the use of XML schemas and the generic design of the interface.

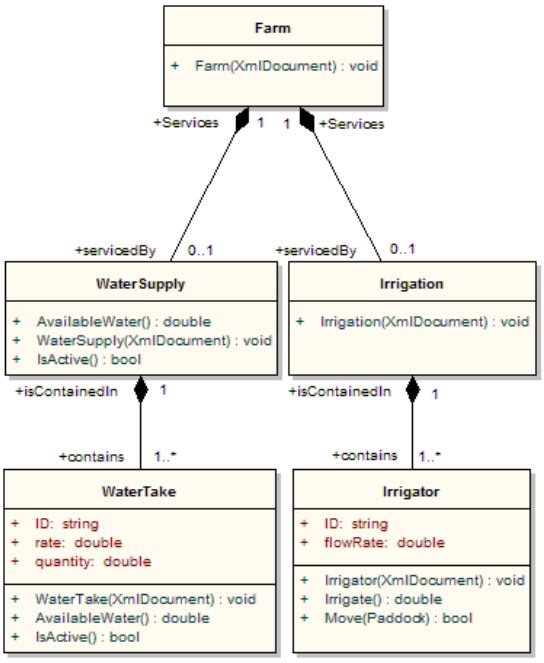
3 USE OF XML SCHEMAS

XML files have considerable advantages due to their ability to contain both data and information about the data and provide an extensible and adaptable data format. However, this extensibility means that there is no control over the structure other than that it comply with the rules for well-formed XML (W3C 2004a).

Initially a technique using DTDs was developed to ensure XML data was in the correct structure. The use of this technique is described by Kokkonen *et al*. (2003) The recently introduced technique using XML Schemas written using XSDL (W3C 2004b) have considerable advantages over using DTDs (Fraser and Livingstone 2002). They are written using XML rather than a separate language, they provide extensive support for data types, allow namespaces to be used and are fully extensible into structures that are more complex. Using this technique XML documents can be validated using schemas called XML Schema Definitions (XSDs). The use of a validation against an XSD not only ensures that the document is well formed XML, it also ensure that the structure and content of the data is valid for its purpose.

Figure 3 is an example of a simple XSD document that could be used to validate the simple XML

Figure 2 – Farm UML Diagram



Farm data shown in Figure 1 (line numbers are used for ease of description and are not part of the schema). The root (or parent) element required to be in the XML by line 04 of the XSD is "farm", all other elements must be contained within this node. Line 07 requires a "waterSupply" element and Line 38 requires an "irrigation" element (the specification for the irrigation node has been omitted to conserve space). Lines 10 & 11 require one or more "waterTake" elements and these must be made up of an "id" element (line 14), a "quantity" element (line 16) and a rate element (line 24). The data type used for each element can be defined, in the case of the "id" element, it is required to be a string by line 14 and line 15 requires that it cannot be an empty string. The simpleType structure used for "quantity" and "rate" allows an element to be restricted to both a data type (in this case double precision floating-point types in lines 18 and 26) and a value range limits. Using the "quantity" element as an example line 19 provides a minimum value restriction and line 20 a maximum restriction.

This XSD can easily be used by any application using a framework component designed to carry

```
01<?xml version="1.0" encoding="utf-8" ?>
02<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
03     xmlns:xs="http://www.w3.org/2001/XMLSchema">
04 <xs:element name="farm">
05     <xs:complexType>
06        <xs:sequence>
07           <xs:element name="waterSupply">
08              <xs:complexType>
09                 <xs:sequence>
10                    <xs:element name="waterTake" minOccurs="1"
11                           maxOccurs="unbounded" nillable="false">
12                       <xs:complexType>
13                          <xs:sequence>
14                             <xs:element name="id" type="xs:string"
15                                nillable="false" />
16                             <xs:element name="quantity">
17                                <xs:simpleType>
18                                   <xs:restriction base="xs:double">
19                                      <xs:minInclusive value="0" />
20                                      <xs:maxInclusive value="100000" />
21                                   </xs:restriction>
22                                </xs:simpleType>
23                             </xs:element>
24                             <xs:element name="rate">
25                                <xs:simpleType>
26                                   <xs:restriction base="xs:double">
27                                      <xs:minInclusive value="0" />
28                                      <xs:maxInclusive value="1000" />
29                                   </xs:restriction>
30                                </xs:simpleType>
31                             </xs:element>
32                          </xs:sequence>
33                       </xs:complexType>
34                    </xs:element>
35                 </xs:sequence>
36              </xs:complexType>
37           </xs:element>
38           <xs:element name="irrigation">
39              Irrigation node specification here…
40           </xs:element>
41        </xs:sequence>
42     </xs:complexType>
43 </xs:element>
44</xs:schema>
```

Figure 3 – Farm XSD Document

out validation. One appropriate class in the Microsoft .NET Framework is the XmlValidatingReader class in the System.Xml namespace. The program carrying out the validation can be written to return considerable detail about any validation errors found in the document.

The XSD example in Figure 3 is very simple and uses a small subset of features available in the XML schema standard (W3C 2004b). However, the example does demonstrate the power of XML Schemas to require that an XML document conform to a particular structure. Traditionally simulation applications have been supplied with data and control parameters from complex structures developed using delimited-text files. This type of structure contains little or no information about the nature of the data that it contains and the process of identifying errors in the data is difficult.

The production of XSD documents for large data structures is quite complicated and time consuming. However, products such as Microsoft Visual Studio 2003 and 2005 as well as Altova's XMLSpy 2005 (Altova 2005) greatly simply the production of XSD documents, providing tools for the automatic generation of the schemas and graphical interfaces for the visualisation and editing of their contents.

XML Schemas have considerable potential for use in the inoperability of models and the accessibility of information stored in database systems. If participants in a collaborative modelling project are able to agree on the XML schemas for component interfaces, the details of how the individual components are implemented become unimportant. They can also be used in a broker-style component (similar to that described by Kokkonen *et al.* (2003)) to reveal a consistent interface to similar data stored in a number of different database systems.

The widespread practical implementation of these techniques will require all users to agree on the naming and details of elements used in the schemas for component interfaces. The co-ordinated development of shared ontologies (Jolma and Rizzoli 2003) is required to facilitate this.

4 CONSTRUCTION OF SIMULATION ENTITIES

The first task before commencing an object-oriented simulation is to construct an object structure in memory that represents the objects being simulated in their initial state. This step usually involves data being read from text files, a database or other data structure. The controlling program then creates required objects and the appropriate state variables and parameters are set.

XML data files and the System.Xml namespace in the Microsoft .NET framework provide the opportunity to simplify this process. The top-level simulation entity class is able to accept an XML document representing the object structure for all the simulation entities. Using the Document Object Model (DOM) structure and simple XPath queries the top-level class is able to use the XML data to construct itself and supply the appropriate XML nodes to the constructors of its component classes.

This construction process is illustrated by the simplified FarmSim simulation example in Figure 2. In the example, the Farm may have a WaterSupply, which in turn is made up of one or more WaterTakes. The Farm class has a constructor method that takes as an argument an XML document (an object of the .NET framework class XmlDocment which is derived from the XmlNode class) for which a sample XML document is shown in Figure 1. This document contains a Farm node that, in this example, is comprised of a WaterSupply node and an Irrigation node. The only step the controlling program needs to carry out is to call the constructor for the Farm class and supply it with the XML document in Figure 1. (It is assumed that prior to this step the controlling program has used an XSDL schema to ensure that the XML document is in a valid format for use by the Farm class constructor).

The Farm constructor uses simple XPath queries to select the nodes for the various objects that make up the farm. Figure 4 is a C# .NET code sample that returns the WaterSupply node (using the XPath query contained within the double quotation marks) and then supplies the extracted node to the constructor of the WaterSupply class:

```
XmlNode wsNode = farmDoc.SelectSingleNode(
   "/farm/WaterSupply");

waterSupply = new WaterSupply(wsNode);
```

Figure 4 – Farm Constructor Code

The constructor of the WaterSupply class now has only the XML node that relates itself and in this case, it is made up of two WaterTake nodes. The Farm constructor will also contain the code to deal with the remaining nodes in the input XML (in the Figure 1 example, code to extract and construct the irrigation node will be required).

Figure 5 is a code sample from the WaterSupply constructor:

```
XmlNodeList nodes = wsNode.ChildNodes;

waterTakes = new WaterTake[wtNodes.Count];

unit w=0;

foreach (XmlNode take in wtNodes)
{
   waterTakes[w] = new WaterTake(take);
   w++;
}
```

Figure 5 – WaterSupply Constructor Code

This code creates an XmlNodeList (another .NET framework class that is an ordered collection of XML Nodes) from the child nodes that contains all of the WaterTakes. The array that contains the WaterTakes is then created using the number of elements in the WaterTakes node list (using the public Count property of the list). The C# foreach looping operation and a counter is then used to work through the nodes in the XmlNodeList, creating a new WaterTake (by calling its constructor with the appropriate node) and placing each one in the WaterTakes array. The following code sample (Figure 6) is from the WaterTake constructor:

```
foreach (XmlNode node in take)
{
   switch (node.Name)
   {
      case "ID":
         ID = node.InnerText;
         break;
      case "rate":
         rate = XmlConvert.ToDouble
            (node.InnerText);
         break;
      case "ID":
         quantity = XmlConvert.ToDouble
            (node.InnerText);
         break;
   }
}
```

Figure 6 – WaterTake Constructor Code

This constructor again uses a foreach loop to visit each node in the input XML. In this situation, it is used in conjunction with a switch statement that examines the name of each node and specifies the appropriate actions to be taken. The .NET framework specifies a number of specific XML conversions to translate XML string data into variables of particular types. In this instance, the conversion used is to a double-precision floating-point variable using the XmlConvert.ToDouble method (as the ID field is a string it can be directly be assigned the InnerText of the relevant XmlNode).

The parent document supplied to the Farm constructor has already been validated against the appropriate XSDL schema. This means that there is no need to carry out extensive checking and validation at the constructor level, as the structure and content of the XML has already been validated. However, there are some limitations to the extent of validation that can be carried out using XSDL schemas. There is always likely a need for some level of validation of the data once the XML document has been read. For example, a situation where the parameters set in one part of the model determine the settings that should be used in another.

While the example chosen is deliberately trivial, this method is very robust and capable of handling huge object structures so long as the design of the input XML has been carefully considered. One significant advantage of this approach is that the logic required to construct the object structure is contained within the classes that make up the structure. This simplifies the on-going extension and customisation of the structure. Using the example provided, the addition of a field to the WaterTake class is simply a matter of including an additional case statement and conversion/assignment in its constructor. The XSDL schema used to validate the Farm XML will also require an addition to allow/require the inclusion of the field and require it to be of a specific type. This approach promotes the loose coupling of the simulation entities to the simulation controller and user interfaces. All these components need to know about the Farm is the XSDL schema for the XML data that they require.

5 CONCLUSIONS

The use of XML data is considered to have considerable potential for the representation of simulation data and to provide interoperability between simulation components. XML provides effective cross-platform data representation. When combined with XML schemas, XML provides a method of documenting data interfaces that is also able to provide effective validation of incoming documents.

The Microsoft .NET framework provides extensive XML functionality including the processing of documents using the DOM. This means that simulation classes written in .NET languages (such as Microsoft C#) can take advantage of the ability of XML to represent complex data structures. Such classes can include constructors that are able to accept XML documents and process then directly. Where the objects are composed of other objects these portions of the document can be extracted using XPath queries and the relevant

XML nodes can then be passed to the constructors of those classes.

The experience with the initial development of the IRAP FarmSim project is that the use of XML has improved the adaptability of the framework and the usefulness of the output data. It is expected that these benefits will increase as the project continues when the interoperability of the system with other components will be more significantly tested.

6 REFERENCES

Altova, (2005), *Altova XMLSpy 2005*, http://www.altova.com/products_ide.html Last Accessed August 8, 2005.

Argent, R. M. (2004), An overview of model integration for environmental applications - components, frameworks and semantics, *Environmental Modelling and Software,* 19(3), 219-234.

Fishwick, P. A. (2002), Using XML for simulation modeling*, paper presented at the Winter Simulation Conference*, San Deigo, USA, December 8-11.

Fraser, S., and Livingstone, S. (2002). *Beginning C# XML - Essential XML Skills for C# Programmers*. Birmingham, UK: Wrox Press.

Good, J. M., and Bright, J. (2005), An Object-Oriented Software Framework for the Farm-Scale Simulation of Nitrate Leaching from Agricultural Land Uses - IRAP FarmSim*, paper presented at the International Congress on Modelling and Simulation - MODSIM05*, Melbourne, Australia, December 12-15.

Jolma, A., and Rizzoli, A. (2003), A review of interoperability techniques for models, data, and knowledge in environmental software*, paper presented at the International Congress on Modelling and Simulation - MODSIM03*, Townsville, Australia, July 14-17.

Kokkonen, T., Jolma, A., and Koivusalo, H. (2003), Interfacing environmental simulation models and databases using XML, *Environmental Modelling and Software,* 18(5), 436-471.

OpenOffice.org, (2005), *OpenOffice.org - free office suite*, http://www.openoffice.org/ Last Accessed August 8, 2005.

W3C, World Wide Web Consortium (2004a), *Extensible Markup Language (XML) 1.0 (Third Edition) - W3C Recommendation 04 February 2004*, http://www.w3.org/TR/2004/REC-xml-20040204/ Last Accessed June 28, 2005.

W3C, World Wide Web Consortium (2004b), *XML Schema Part 0: Primer Second Edition - W3C Recommendation 28 October 2004*, http://www.w3.org/TR/xmlschema-0/ Last Accessed June 28, 2005.