

# SVG: putting XML in the picture

## John McKeown

Trinity College Dublin, Dublin, Ireland  
john.mckeown@cs.tcd.ie  
<http://www.cs.tcd.ie/John.McKeown>

## Jane Grimson

Trinity College Dublin, Dublin, Ireland  
jane.grimson@cs.tcd.ie  
<http://www.cs.tcd.ie/Jane.Grimson>

### **Abstract:**

*The emergence of XML (Extensible Markup Language) has led to the development of powerful new technologies to overcome the limitations that exist with HTML (HyperText Markup Language), but it has also provided the framework for developing open document/data formats. The SVG (Scalable Vector Graphics) Specification describes one such data format for providing 2-dimensional vector graphics on the Web. Although SVG is not yet a standard, it has already attracted considerable attention from the Web design community and support from software vendors. A well-supported vector graphics standard would provide Web designers with a format for creating more flexible graphics to include in their Web pages. The fact that this vector graphics standard will also be an XML format makes it even more powerful. Apart from the obvious advantages of being non-proprietary and platform independent, it may be styled using style sheets, processed by any XML parser, and just like XML, it can be generated dynamically in a number of different ways.*

## Introduction

The traditional approach to presentation of information on the Web involves a combination of (simply) formatted text and raster (pixel-based) graphics (GIF and JPEG). Raster graphics are well suited to storing photographic images, however in Web presentation, they are often used to produce visual elements such as borders, graphical buttons, and logos. This approach can lead to a number of problems due to the static nature of these graphic formats. In particular raster graphics are not easy to maintain, and more important they do not scale well, which may impose restrictions on how the Web page is viewed. Raster graphics specify information about each pixel of an image so the size of the image is fixed. When a raster image is scaled, there is a distortion or loss in quality in the image. Ideally the graphics in a Web page should be able to scale to the size of the viewing area. This is possible with vector-based graphics. Vector Graphics offer a more flexible means of adding graphical elements to Web pages and can also provide a new method of presenting information.

Vector Graphics have been around since before the Web, however, they have not succeeded in gaining the widespread acceptance and usage on the Web enjoyed by the raster formats. A number of well-defined formats are available with Web capabilities, but these formats tend to be proprietary and require specific authoring and rendering tools. XML can be used to create text-based file formats that are platform independent and non-proprietary. XML has already been used to define a number of Vector Graphics formats submitted as notes to the W3C (World Wide Web Consortium). These include Vector Markup Language, and Precision Graphics Markup Language. As a result of these and other submissions, the W3C chartered a Scalable Vector Graphics working group that produced the Scalable Vector Graphics (SVG) 1.0 Specification

SVG is a powerful language for describing two-dimensional vector and mixed vector/raster graphics in XML. SVG was designed to integrate with other Web standards efforts like XLink, XML Namespaces, DOM (Document Object Model) , CSS (Cascading Style Sheets) and XSL (Extensible Stylesheet Language) . SVG images can use Stylesheets to control the look and feel of the image, and scripting to provide interactivity and animation. The specification also describes more complex functionality such as Filters Effects, Gradients, and Patterns. SVG has the potential to become the **de facto** standard for vector graphics.

Given that it is an XML format, SVG is platform independent and non-proprietary . It is also a text based format, and can therefore be edited by hand using a simple text editor rather than a special authoring tool, although this may not be acceptable for more complicated images. SVG graphics can be parsed using any XML compliant parser and may be transformed using XSLT (XSL Transformations ) stylesheets. SVG is used to markup up graphical data in the same way that XML and HTML are used to markup text. Therefore SVG can be generated dynamically using the same techniques for generating XML and HTML (CGI (Common Gateway Interface) Scripts, Java Servlets, Server Pages).

The ability to generate an SVG image dynamically enables Web designers to create customised graphics for their Web sites, thus enhancing the look and feel of their site. It also introduces the possibility of including external information into the image. This paper will present work carried out on a template-based XML Generator, which can be used to integrate SVG image templates with external information sources to produce an SVG image. The template approach will be examined as a means of providing dynamic graphics within a Web site. The SVG graphics produced by this Generator may be customised in different ways i.e. appearance, and content. Applications of this approach will be described to illustrate its usefulness.

## Motivation

The motivation for this work is based on experiences from the MAPPA Project , a project to develop a flexible platform allowing the general public to access complex information through multimedia service kiosks and Internet-enabled mobile technologies. To present complex information in a way that can be easily understood requires very fine integration of the data with the presentation environment. Although proprietary solutions may exist to allow this kind of integration of data and multimedia, an alternative was explored that could use emerging web technologies to provide a more open framework.

## Dynamic Web Content

Dynamic content has become a very important aspect in Web publishing. A Web site used to consist of a collection of static HTML pages. As the online service aspect of the Web emerged, the ability to generate HTML pages dynamically became very important, allowing a business to integrate up-to-the-minute data into their Web pages. Dynamic features added to Web sites can customise the content, layout, and simple formatting offered by HTML. However these features do not apply to the graphical content of the Web site.

The raster graphics formats most commonly used on the Web are static in both size and content. Vector graphics on the other hand are more dynamic, as the contents of an image are specified as a sequence of geometrical instructions that produce the image when evaluated. An open vector graphics format like SVG has the potential to deliver customised graphics to visitors of a Web site. Producing customised vector graphics with SVG involves the integration of external information with these geometrical statements to produce an image. With SVG this integration step is simply a matter of producing XML dynamically, something which is possible to do in a variety of ways. However, these techniques for producing XML dynamically may not suit the production of SVG.

To illustrate the shortcomings of some of the popular approaches to producing Dynamic Web content (including XML), consider a simple example that uses SVG to produce the graph in Figure 1. The graph illustrates the grades for four students in a class. Some of the current techniques for producing dynamic XML are examined below to show how they could be used to produce this graph.

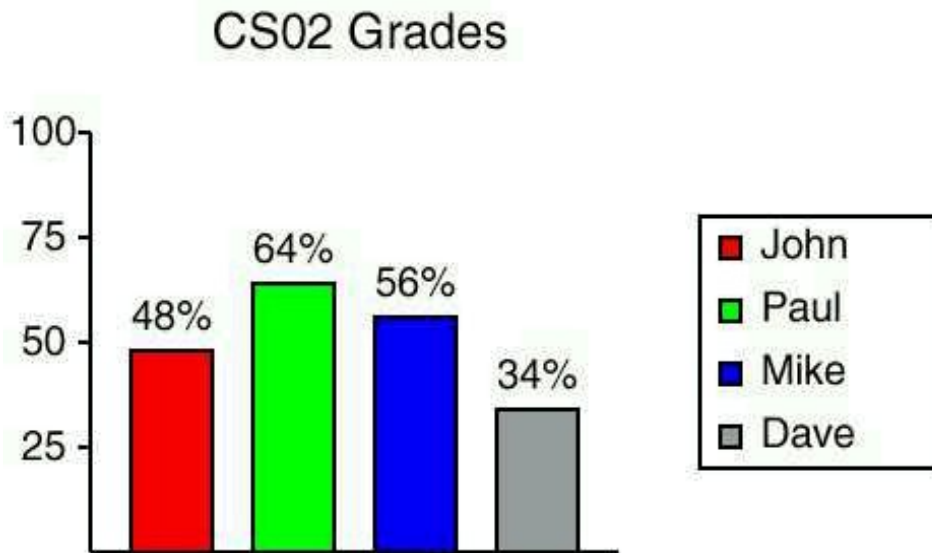


Figure 1. Graph of student grades

## Java Servlets and CGI scripts

Both Java Servlets and CGI scripts are invoked by a HTTP (HyperText transfer Protocol) request, and perform some processing to produce a suitable response. These technologies can be used to create HTML or XML on the fly. Both of these approaches are essentially a programmatic way of producing dynamic content. To change some aspect of the processing or modify the HTML/XML output in some way involves changing the program code directly.

Using either of these methods to produce SVG images dynamically makes the task of changing the image more difficult. For instance given a Java Servlet that produces the graph above, the task of modifying some part of the image involves modifying the Java code. In the example above this may seem trivial, but for a more complex image the task of editing the image is usually performed with the aid of a suitable graphics editing tool, instead of editing a text file, or program code. It is also a task more likely to be performed by a graphic designer than a programmer. The sample Java code below shows part of the doGet method of a Java servlet that produces an SVG image in response to a HTTP GET command. This code shows the principle problem with this approach that is the XML/SVG tags are contained within the program code and cannot be edited separately.

```
// Get Method of a Java Servlet that produces an SVG image containing
// a graph of student grades
public void doGet (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
{
    res.setContentType("image/svg");
    PrintWriter w = res.getWriter();
```

```

/* Print Prolog, Doctype, and Styles.....*/

w.println("<g>");
w.println("<g>");
w.println("<text x='40' y='87' class='text'>48%</text>");
w.println("<rect class='bar1' x='40' y='140' width='20' height='-48' />");
w.println("</g>");
w.println("<g>");
w.println("<text x='70' y='71' class='text'>64%</text>");
w.println("<rect class='bar2' x='70' y='140' width='20' height='-64' />");
w.println("</g>");
w.println("<g>");
w.println("<text x='100' y='79' class='text'>56%</text>");
w.println("<rect class='bar3' x='100' y='140' width='20' height='-56' />");
w.println("</g>");
w.println("<g>");
w.println("<text x='130' y='101' class='text'>34%</text>");
w.println("<rect class='bar4' x='130' y='140' width='20' height='-34' />");
w.println("</g>");
w.println("</g>");

/* Print the other objects contained in the image..... */

} // end doGet

```

## Server pages

The term Server Pages refers to ASP (Active Server Pages) , and JSP (Java Server Pages) . The server pages approach to dynamic Web pages embeds programming code within the actual Web page to produce a server page. When the server page is requested (i.e. by a Web browser), the code within the page is evaluated which provides the dynamic content. The finished page is then returned to the requesting clients.

Generating the graph above using this approach would also work, however, the server page file is not a well-formed SVG document and therefore cannot be edited using SVG software. The Java Server Pages specification has tried to address this issue by setting out a well-formed XML representation of a JSP file, however the use of this representation is not required. The sample code below is an excerpt of a Java Server Page showing how Java code is embedded in the page. The way in which the code is embedded in the Server Page is not XML 1.0 compliant and therefore the server page is not well-formed XML. There are other Server Pages initiatives attempting to make an XML compliant server page syntax, most notable is the Apache Cocoon Project's XSP (eXtensible Server Pages) .

```

<?xml version="1.0" standalone="yes"?>
  <svg width="4in" height="3in"
    xmlns = 'http://www.w3.org/2000/svg-20000303-stylable'>
    <rect class='bar1' x='40' y='40' width='80' height='60' />");
    <text x='70' y='71' class='text'>
      <% System.out.println(coursename + "Grades"); %>
    </text>
  </svg>

```

## XML and XSL transformations

A more recent approach to generating dynamic Web content involves the use of XSL Transformations , a part of the eXtensible Stylesheet Language (XSL). XSLT is a language for transforming XML documents

into other XML documents. The transformation is defined by a collection of rules/templates which indicate how the source document should be processed to produce the output document.

Given the XML source document below, an XSLT stylesheet could be developed to produce an SVG graph using the information in this source document. An example of this is given in the XSLT 1.0 Specification . XSLT may also be used to transform the content and structure of an SVG image to produce a new SVG image. It can also handle multiple XML source documents, however there is no defined way of using data from other sources like a database or text file. Although XSLT is a powerful transformation language, it is not intended to be a general purpose XML transformation mechanism.

```
<grades>
  <student>
    <name>John</name>
    <cs01>45</cs01>
    <cs02>48</cs02>
    <cs03>57</cs03>
  </student>

  <student>
    <name>Paul</name>
    <cs01>67</cs01>
    <cs02>64</cs02>
    <cs03>75</cs03>
  </student>

  <student>
    <name>Mike</name>
    <cs01>74</cs01>
    <cs02>56</cs02>
    <cs03>64</cs03>
  </student>

  <student>
    <name>Dave</name>
    <cs01>54</cs01>
    <cs02>34</cs02>
    <cs03>69</cs03>
  </student>
</grades>
```

## System design

The approaches mentioned above are all suitable for producing dynamic web content, however they all possess limitations, or impose restrictions on the production of dynamic SVG. SVG images are examples of Data-centric XML, as opposed to Document-centric . The distinction between the two can be summarised as fairly regular structure, fine-grained data, with very little or no mixed content. The XML contained in an SVG file determines the visual appearance of the image. SVG possesses some powerful graphics features, making the creation and editing of complex SVG images very difficult without the aid of suitable graphics software. Using the approaches mentioned earlier to generate SVG would change the image from XML into something else e.g. program code, making it illegible to any graphics software which understands SVG.

SVG is compatible with both the XML 1.0 Recommendation , and the Namespaces in XML Recommendations . The latter allows elements from foreign namespaces to appear within an SVG image. The SVG

specification states that foreign namespace elements should be included in the SVG DOM by user agents but can be otherwise ignored. This allows commands to be embedded in an SVG image using elements for a foreign namespace without affecting the integrity of the SVG image. The sample SVG document below shows the **barchart** and **bar** elements from the **graph** namespace included in an SVG image using a namespace declaration.

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in"
      xmlns = 'http://www.w3.org/2000/svg-20000303-stylable'>
  <defs>
    <graph:barchart xmlns:graph="http://mycompany/graphapp"
                    title="CS02 Grades">
      <graph:bar label="John" value="48"/>
      <graph:bar label="Paul" value="64"/>
      <graph:bar label="Mike" value="56"/>
      <graph:bar label="Dave" value="34"/>
    </graph:barchart>
  </defs>

  <desc>This chart includes private data in another namespace
  </desc>

  <!-- In here would be the actual graphics elements which
  draw the bar chart -->
</svg>
```

To address the limitations of the methods mentioned earlier, a new approach was devised using template files as the basis for producing dynamic content. The templates contain references to external information sources and instructions that indicate how these information sources should be processed for inclusion in the document. The templates themselves are well formed XML documents, and the additional instructions are embedded in the template by means of foreign namespace elements. These templates are processed by a **Generator** application that retrieves data from the referenced information sources, and processes it according to the instructions in the template. The result of this processing is another well formed XML document similar to the template but with dynamic content.

For example, the graph in Figure 2 shows a template SVG image for the graph example earlier. This template contains all the elements of the final image but lacks the actual data that determines how the graph will look. Data is required for the key names, the value for each grade, and title of the graph. The template contains references to the data required, and instructions to the Generator on how the data should be included. After the Generator has processed the template below, a graph similar to that in Figure 1 will be produced. This Generator processing could be carried out by some server-side process, producing dynamic graphics in response to requests from clients.

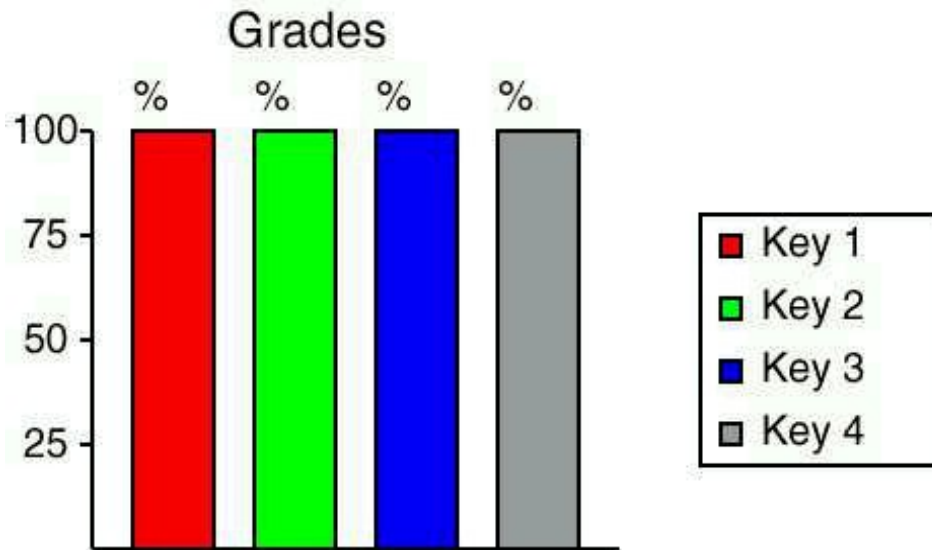


Figure 2. Template graph of student grades

## Design goals

The following goals were identified for the design of an application to generate dynamic XML which also suits the generation of SVG images.

1. The Template files used by the Generator must be well-formed XML (conforming to XML 1.0 and XML Namespaces). In the case of SVG the templates must be **Conforming SVG Stand-Alone Files** .
2. The method for referencing external information sources should be concise and simple.
3. The set of Generator instructions should be minimal.
4. The template files should Integrate with existing W3C recommendations and working drafts.
5. Support separation of roles for providing content on the web (content provider, graphic designer).

## Types of information sources

The template files processed by the Generator may reference different types of information sources. These references are interpreted by the Generator application and the data is retrieved for inclusion into the template file.

### Database

The template file may reference data stored in a Database. The reference can include a particular query to execute on the database, or some predefined view of the database. Database access is provided by the JDBC (Java DataBase Connectivity) API (Application Programming Interface) .

### Java class

To allow more flexible information sources, a Java classes may be used. This class requires a static method that the generator calls to retrieve data.

## URL

A URL can be specified as the information source. The protocol for the URL could be either HTTP, FTP, or FILE. Therefore the information could come from a text file, or in the case of HTTP, it could be returned by a servlet or cgi script.

## Information source formats

The data retrieved from an information source needs to be in a format understood by the Generator. The following formats are accepted:

### Java properties format

Data returned in the attribute/value pair format used in a Java properties file.

```
firstname=John
secondname=Doe
age=42
```

### Comma delimited text

The delimited text format offers a more tabular format for the data. The first row contains the field names separated by commas. Each row after that contains a result set with values for each of the fields, again separated by commas.

```
firstname, secondname, age
John, Doe, 42
```

### XML fragment

The information source may be an XML document fragment which can either be included in the generated document or further processed to extract certain data.

```
<data>
  <firstname>John</firstname>
  <secondname>Doe</secondname>
  <age>42</age>
</data>
```

## Template instructions

The set of instructions which can be embedded in the templates files should include instructions to insert or replace content within the document. Ideally the template should contain all the XML elements required, therefore the instructions in the template only change the attributes and content of these elements. Instructions to create new elements are not required important, but replicating elements of the DOM tree may be useful. The following set of instructions are supported within the template files.

### Importing information sources (processing instruction)

Information sources are imported into the template files using a Processing Instruction at the beginning of the file. The information source is referenced using the **src** attribute whose value is a URI (Uniform Resource Identifier) . As mentioned earlier there are three types of information source supported by the Generator (Database, Java Class, and URL). The URIs to reference a Database or Java class use two special

schemes (xjdbc:/// and xjava:///), which adhere to the URI guidelines . The code below shows the Processing Instructions required to import data from a Database and from a Java class. The other two attributes of the Processing Instruction specify the type of the information source (table, props, xml), and the prefix used to refer to this information source.

```

    <!-- Import a database -->
    <?dxml
    src="xjdbc:JDBCdriver?query=Select name, CS02 From Grades#JDBCDataSource"
    type="table"
    prefix="is1"
    ?>

    <!-- Import from a Java Class -->
    <?dxml
    src="xjava:MyJavaClass?course=CS02#getCourseData"
    type="table"
    prefix="is2"
    ?>

```

### Replace content (attribute)

The replace content attribute can be specified on any tag so that the contents of that tag are replaced with the specified information. The following code shows how this attribute was used in the graph template from earlier, to replace the name of a Key in the graph legend. This example uses data from the information source **is1**, which is in a tabular format. The attribute references the first row in the **Name** field of this information source. The text element shown in this example is taken from the legend of the graph template in Figure 2. The result of this instruction after it has been processed by the Generator is shown in Figure 1.

```

    <text x="195" y="70" class="text" is1:replace-content="Text(Name, 1)">
    Key 1</text>

```

### Insert content (element)

The insert content element can be used to insert data from the information source into the content of the element which contains it. This instruction will not affect any content already contained within the element. The code below inserts the title of the second field into the text element.

```

    <text x="60" y="20" class="text" style="fill-rule:nonzero;">
    <is1:insert-content value="Text(2,0)"/> Grades
    </text>

```

### Insert attribute (attribute)

This instruction inserts the specified attribute into the containing element. If the attribute already exists within the element then the attribute is replaced. The example below will replace the y attribute of the text element. In this case a numerical value is copied from the first row of the second field in the information source. Some simple expression is performed on this value to obtain the new value for y.

```

    <text x="40" y="35" is1:insert-attribute="y,Value(35+(Num(2,1)/2))">
    Some Text</text>

```

## Applications

A number of interesting applications of this dynamic SVG Generator are outlined and discussed below.

### **Graphical templates**

The use of SVG templates described here allows graphics to be created which present information in a particular way e.g. a graph. Once created, this template can be used to display other sets of information by just modifying the information sources referenced.

### **Dynamic images**

Dynamic images are not easily achieved with the existing technologies for Web presentation. When SVG becomes a full web standard it will provide Web page designers with an open format for inserting graphical content in their web-pages. As already mentioned in this paper these graphics can be generated dynamically because SVG is an XML format. Using the Generator application described here will allow information to be integrated into the image when it is requested, resulting in customised graphics.

### **Web applications**

Web applications are an alternative to the traditional Web presentation of formatted text and simple graphics. Web applications are generally graphically rich Web sites created with proprietary graphics software. An open vector graphics format such as SVG would support this type of web presentation, allowing a Web site to be created using only SVG images. Text can be included in these SVG images and formatted using CSS rules. SVG has the additional advantage of being scalable and can be viewed on different screen sizes without a loss in quality.

### **Conclusions**

This paper presented an alternative approach to generating dynamic XML content for the Web, one which is better suited to producing data centric XML than the existing approaches. The suitability of this approach was illustrated with examples of generating dynamic SVG images. Although the SVG examples presented were very simple they do help to highlight the limitations of current approaches, and illustrate the usefulness of this approach when applied to more complex SVG images. The templates used in this approach make it easier to present rapidly changing data. They also allow the same presentation to be applied to different data by simply changing the information source. The structure of the templates is such that they remain well-formed XML documents and can be interpreted by XML software. The Generator application described here is a prototype to demonstrate this approach and assess its effectiveness.

### **Acknowledgments**

The authors wish to thank the members of the Knowledge and Data Engineering Group, from the Computer Science Department in Trinity College Dublin. Special thanks also to the members of the MAPPA Project Consortium.

### **Bibliography**

- [BOURRET] R. Bourret, **XML and Databases**, ( <http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/XMLAndDatabases.htm> )
- [GR99] M. Gould, A. Ribalaygua, **SVG - A New Breed of Web-Enabled Graphics** , GeoWorld 12(3), pp. 46-48. 0000. ( <http://www.geoplace.com/gw/1999/0399/399svg.asp> )
- [JSP] E. Pelegrí-Llopart, L. Cable, **JavaServer Pages™ Specification Version 1.1**, Sun Microsystems 17<sup>th</sup> December 1999. ( <http://java.sun.com/products/jsp/> )
- [MAPPA] **MAPPA Project Web page**, ( <http://www.sics.se/mappa/> )
- [PGML] N. Al-Shamma, R. Ayers, R. Cohn, J. Ferraiolo, M. Newell, R. K. de Bry, K. McCluskey, J. Evans, **Precision Graphics Markup Language (PGML)** , W3C Note Submitted 10<sup>th</sup> April 1998, ( <http://www.w3.org/TR/1998/NOTE-PGML> )
- [RFC2396] T. Berners-Lee, R. Fielding, L. Masinter, **Uniform Resource Identifiers (URI): Generic Syntax**, <http://www.ietf.org/rfc/rfc2396.txt> , August 1998.
- [SVG] J. Ferraiolo, **Scalable Vector Graphics (SVG) 1.0 Specification**, W3C Working Draft, 3<sup>rd</sup> March 2000. ( <http://www.w3.org/TR/SVG/> )
- [VML] B. Mathews, D. Lee, B. Dister, J. Bowler, H. Cooperstein, A. Jindal, T. Nguyen, P. Wu, T. Sandal, **Vector Markup Language (VML)**, W3C Note Submitted 13<sup>th</sup> May 1998, ( <http://www.w3.org/TR/NOTE-VML> )
- [XML] T. Bray, J. Paoli, C. M. Sperberg-McQueen, **Extensible Markup Language (XML) 1.0**, W3C Recommendation, 10<sup>th</sup> February 1998. ( <http://www.w3.org/TR/REC-xml> )
- [XMLNAMES] T. Bray, D. Hollander, A. Layman, **Namespaces in XML**, W3C Recommendation, 14<sup>th</sup> January 1999. ( <http://www.w3.org/TR/REC-xml-names> )
- [XSLT] J. Clark, **XSL Transformations (XSLT) Version 1.0**, W3C Recommendation, 16<sup>th</sup> November 1999 ( <http://www.w3.org/TR/xslt> )
- [XSP] S. Mazzocchi, **eXtensible Server Pages (XSP) Layer 1**, Apache XML Project, Working Draft, ( <http://xml.apache.org/cocoon/> )

## Authors

### John McKeown

Research Student  
Trinity College Dublin  
Postal Address:  
Knowledge and Data Engineering Group  
Department of Computer Science  
2 Dublin  
Ireland  
Telephone: +353 1 608 2158  
Fax: +353 1 677 2204  
E-mail: john.mckeown@cs.tcd.ie  
Web: www.cs.tcd.ie/John.McKeown

**John McKeown** - John McKeown received a bachelors degree in Computer Science from Trinity College Dublin and is currently an MSc. Student in the Knowledge and Data Engineering Group at Trinity College Dublin. He has been involved in a number of projects to provide access to complex information using multimedia service kiosks. He is currently active on the MAPPA project. He is also a co-founder of the Trinity College XML working group.

### Jane Grimson

Associate Professor  
Trinity College Dublin  
Postal Address:  
Centre for Health Informatics  
Department of Computer Science  
2 Dublin  
Ireland  
Telephone: +353 1 608 1594  
Fax: +353 1 677 2204  
E-mail: jane.grimson@cs.tcd.ie  
Web: www.cs.tcd.ie/Jane.Grimson

**Jane Grimson** - Professor Jane Grimson obtained her bachelors degree in Engineering from Trinity College Dublin and her Masters and Doctorate in Computer Science from the Universities of Toronto and Edinburgh. She joined the Department of Computer Science in Trinity College Dublin as a lecturer in 1980 where she is now an Associate Professor. She was Dean of the Faculty of Engineering and Systems Sciences from 1996 to 1999. She has published over 80 papers in journals and the proceedings of international conferences, and has co-authored a textbook on Distributed Database Systems published by Addison-Wesley.

Her main research interests are in distributed information systems and health informatics. She has acted as principal investigator on several EU and national research projects. She established the Knowledge and Data Engineering Research Group in the Department of Computer Science and also, together with colleagues from the Faculty of Health Sciences, the Centre for Health Informatics in Trinity College. She is a Chartered Engineer and Fellow of the British Computer Society, Irish Computer Society, Institution of Engineers of Ireland, Irish Academy of Engineering, and the Royal Academy of Medicine in Ireland and a member of ACM and IEEE. She is a member of the Irish Council for Science, Technology and Innovation and chaired the Materials and Manufacturing Processes Panel of the Government's Technology Foresight Initiative. She is President of the Institution of Engineers of Ireland for 1999-2000.