

# Stirring XML: Visualisations in SVG

Benjamin **Jung** <benjamin.jung@cs.tcd.ie>  
Jamie **Brohan** <jamie@braynet.ie>  
Richard **Maher** <richard.maher@loox.com>  
Laura **O'Shea** <ljoshea50@hotmail.com>  
Vincent **Wade** <vincent.wade@cs.tcd.ie>

## Abstract

The idea of separating content from its presentation information was one of the key aspects in developing Extensible Markup Language (**XML**) and its supporting technologies such as e.g. Extensible Stylesheet Language (**XSL**). **XML** vocabularies are not much good for human viewing and navigating; they do not need to be and they (primarily) have not been designed with this purpose in mind. Instead, transformations from a (source) **XML** vocabulary into a (target) visualisation format, e.g. Hypertext Markup Language (**HTML**), have been used in order to present the information in an easy-to-understand layout. Nevertheless, most of today's **XML** vocabulary visualisations are intra-media transformations, e.g. from a text-based representation to a text-based representation. Future transformations will include inter-media types, e.g. from text-to-graphic or text-to-audio. What role does Scalable Vector Graphics (**SVG**) play in these scenarios?

This paper does not provide detailed programmatic explanations rather it demonstrates design and solutions for data and information visualisation using **SVG** images. It outlines the concepts of one inter-media transformations (e.g. from text to graphic) using different approaches. All work was commenced in the last two years at the Department of Computer Science, Trinity College Dublin (**TCD**) (Ireland) and subsequently submitted and published as **BSc** Final Year Thesis's by the mentioned authors. A PDF version of this paper is available ([StirringXML.pdf](#)).

## Table of Contents

1. Introduction .....	2
2. MusicML2SVG (Laura O'Shea, 2002) .....	2
2.1. Abstract .....	2
2.2. Introduction and Background .....	2
2.3. Source: MusicXML .....	3
2.4. Image Design .....	3
2.5. Design .....	4
2.6. Summary .....	5
3. BioML2SVG (Richard Maher, 2001) .....	6
3.1. Abstract .....	6
3.2. Introduction and Background .....	6
3.3. Source: BioML .....	6
3.4. Image Design .....	7
3.5. Implementation .....	8
3.6. Summary .....	9
4. CALS2SVG (Jamie Brohan, 2001) .....	9
4.1. Abstract .....	9
4.2. Introduction and Background .....	9
4.3. Source: CALS .....	10
4.4. Image Design .....	10
4.5. Implementation .....	12
4.6. Summary .....	12
5. ....	12
Bibliography .....	13

## 1. Introduction

An ancient proverb states that 'a picture says more than a thousand words'. Although this is true for most graphics, only experts in the images' content-domain may be able to extract and interpret almost each fragment of information an image holds. Graphics are not always self explanatory and 'a few words might say more than a thousand pictures'. Accompanying (e.g. textual) representation of the information are often essential in order to understand the content in its context. Unfortunately, this additional data is often not available because information is blended together with its presentations. Raster graphics are good examples of these (wrecking) techniques, where an image presents information in a certain style, but 'drilling' into the image does not reveal further relations between the various components of the image. Raster images are comparable to badly written [HTML](#) pages, where content and presentation information is melt together, impossible to distinguish and separate. A solution to this problem lies in the layout-free storage of information. A source schema (often called hub or core) stores the data in its most abstract and logical (raw) way. It is free of presentation information such as colours, scales and media-type. Various transformations techniques are available to facilitate the conversion from the core-schema into a user-preferred representation.

[XML](#) is an ideal syntax for the core-schema. Its integration into a large number of software applications lead to nearly ubiquitous availability of supporting tools. [XML](#) vocabularies exist or are being developed in every imaginable domain and level. As a range of standards exist to define these [XML](#) vocabularies (e.g. Document Type Definition ([DTD](#)), [XML](#) schema, Relax NG), similarly many methods are available to transform an [XML](#) document into another ([XML](#) or non-[XML](#) based) format. Two types of transformation-techniques are presented in this paper: firstly a pure Java application with all modification rules manually coded. This approach offers great flexibility in the creation of computationally complex conversions and storage in binary target formats. Nevertheless, it involves increased maintenance work and implementation time. The second method uses [XSL](#) transformations to convert from the source into a target vocabulary (representation format). In this case, the definition of transformation rules is easier and prototype transformation can quickly be achieved. Complex rearrangements based on e.g. composite mathematical computations are more difficult to implement and can be challenging.

## 2. MusicML2SVG (Laura O'Shea, 2002)

### 2.1. Abstract

The MusicML2SVG project investigated the first step, the use of [XSL](#) transformations in the process of converting files from a non-proprietary [XML](#) vocabulary for music notation (Music Markup Language ([MusicXML](#))) into sheet music visualisations ([SVG](#)). It turned out that the layout and physical placement of musical components (e.g. notes, measures and key-signatures) are very difficult and mathematically challenging. A small set of [XSL](#) transformations were developed to showcase results and feasibility.

### 2.2. Introduction and Background

Physically separating content from its presentation information is one often referenced advantage for multi-media visualisation support. This concept supports the idea of a centralised storage container for the content and a number of transformation files to create customised user- and/or application specific representations of the content. In the world of music, the content is held in a document containing descriptions of the composition (e.g. [MusicXML](#)), which could be visualised by sheet music (e.g. as [SVG](#)) or converted into an audio format (e.g. MPEG). The principal idea of this project is the creation of a prototype which transforms a piece of [MusicXML](#) sheet music into an [SVG](#) based graphical representation of the notation.

Common Western Music notation is a symbolic method of representing music for performers and listeners. It is used in publishing sheet music, musical scores and parts, and has been encoded in over 50 (published and unpublished) different computer formats over the past 30 years. Given the high costs of traditional music publication, many companies have recognised both size and profitability of the sheet music.

To date, the Internet sheet music market has been divided by it's reliance on a proliferation of proprietary binary formats [[Goo02](#)]. PDF, the most common of these, has no musical semantics and can only be displayed on screen

or paper printed. Others are proprietary to the publishing company and can only be viewed, printed or played by their specific software applications. This fragmentation of the online sheet music market is largely responsible for inhibiting its sales potential.

Although many music interchange formats have been developed over the last few decades, none besides Musical Instrument Digital Interface ([MIDI](#)) has met with any significance success.

- Notation Interchange File Format ([NIFF](#)) is based on the binary Resource Interchange File Format (RIFF), and is a standard digital format for the representation of musical notation [[Mou02](#)].
- The Standard Music Description Language ([SDML](#)) Standard Generalized Markup Language ([SGML](#)) vocabulary uses HyTime/Hypermedia and time-based document structuring facilities [[Cov02](#)].
- Silbelius is a state-of -the-art music notation program which helps write, play back and print sheet music [[Sib02](#)].
- The Humdrum toolkit provides a set of inter-related software tools intended to assist in music research. [[Hum02](#)]. It can transform, classify, coordinate, search, transfer, restructure, contextualise, compare and otherwise manipulate both pre- and user-defined information.
- MuseData is the primary encoding system used by the Centre for Computer Assisted Research in the Humanities ([CCARH](#)). Information is stored in plain American Standard Code for Information Interchange ([ASCII](#)) files which represent the logical content of scores in a software neutral fashion [[Hew02](#)].
- 4ML is an [XML](#)-based Music and Lyric Markup Language, recently defined with not a lot software support so far [[Mon02](#)]. FlowML, another [XML](#) vocabulary to store synthesis instruments as a graph of reusable blocks, differs from existing languages for audio systems, as it is not a programming language but a data format [[Sch02](#)].

### 2.3. Source: MusicXML

[MusicXML](#) represents common Western music notation from the 17th century onwards. Based on [XML](#), it serves as an interchange format for applications in music notation, music analysis, music information retrieval and musical performance. Thus, it augments existing specialised formats for individual applications without replacing them. [MusicXML](#) has become the most successful standard for music notation interchange since [MIDI](#). However, without customised products, there is no platform-independent method of graphically rendering a [MusicXML](#) file. The Recordare group [[Rec02](#)] is currently working on a prototype for a system that would convert from [MusicXML](#) to a graphical format such as [SVG](#). This transformation is complex and much harder than you would expect from a prototype [[Clo02](#)].

### 2.4. Image Design

At the beginning of the transformation in this prototype, a first line of musical staff is created, followed by clef as well as key and time signature. This adds an appropriate number of sharps and flats at the beginning of the musical staff and two values representing the time signature. The placement of the musical symbols themselves is decidedly more complicated. The y-coordinate of each note is decided by two things: the value of the step (i.e. the note name) and the value of the octave. The x-coordinate is computed by the note's order in the sequential line of the music. Depending on the note's type, the corresponding Unicode character would be created, possibly together with a mandatory ledger line. As each note above the middle line of the musical staff should have a downward stem direction, a symbol which is not provided by the Unicode character set, a rotation around the note's head centre point was applied. Using the [SVG](#) rotate attribute seemed to be problematic as the character rotates around the bottom left-hand corner of the imaginary frame of the character. An additional transform attribute fixed the problem, leaving the note's head at its original location. Start and end of measures are defined in the [MusicXML](#) document and drawn accordingly. Numerical challenges such as varying number of measures per musical staff, and aesthetical challenges such as justification of measures per musical staff and notes per measure had to be solved.

The complexity of the musical representation challenge is greatly increased by the provision of placement algorithms for multiple staves. The y-coordinates of key and time signatures and clefs are not longer fixed at one single

location, but have to be repeated at the beginning of each line. Musical symbols have to be arranged on a two- and three-dimensional space, e.g. page and set-of-pages respectively.



Figure 1. Three line music sheet ([MLscribe.svg](#))

## 2.5. Design

Music notation, by nature, has a complex and exact means of representing musical information. It has an elaborate structure and each element of a piece of music, be it a full length or a staccato dot, has a direct effect on the aesthetics of the music, both in sound when performed and meaning when read. Because of the complexities of pitch representation and layout, creating a graphical representation of musical data is a highly complex problem. Considerable structure, including detailed specification of vertical and horizontal relationships between graphical elements, is required to properly represent musical scores. Indication of pitch and layout of resulting musical structures involves not only specifications for the vertical relationship between musical notes, but in multiple staves between instrumental part and so forth.

In this project, a standalone [XSL](#) engine (e.g. SAXON [[Sax03](#)]) takes a [MusicXML](#) document together with an [XSL](#) style sheet to create a resulting [SVG](#) file. This can be viewed in any [SVG](#) aware application. For every musical element encountered in the [MusicXML](#) document (such as music staff, measures, clef, note), the [XSL](#) style sheet creates a corresponding graphics element in the [SVG](#) code. The necessary musical symbols are all defined in Unicode (code range x1D100 until x1D1DD) [[Uni03](#)] which prevented the author from manually construct them using basic geometric forms such as circles and lines. At the time of the project, it was difficult to find a font that actually would support this range of the Unicode character set, which lead to a very simplified subset of notes and replacement characters. Nevertheless, the principle of transforming [MusicXML](#) into an [SVG](#) representation was successfully demonstrated.

The following code fragment shows the beginning of a sample [MusicXML](#) document. Its two main pieces, the part attributes key-signature, time-signature and cleff as well as the first note (G4), can clearly be identified.

```
<?xml version="1.0"?>
<score-partwise>
  <part id="Scale_of_G">
    <attributes>
      <divisions>1</divisions>
      <key>
        <fifths>1</fifths>
      </key>
      <time symbol="common">
        <beats>4</beats>
        <beat-type>4</beat-type>
      </time>
      <clef>
```

```

    <sign>G</sign>
    <line>2</line>
  </clef>
</attributes>
<note id = "G">
  <name>G</name>
  <pitch>
    <step>G</step>
    <octave>4</octave>
  </pitch>
  <duration>1</duration>
  <type>quarter</type>
  <stem>up</stem>
</note>
[... ]
</part
</score-partwise>

```

The next code fragment shows the result from the [MusicXML](#) to [SVG](#) transformation of the previous section. It is important to mention that this transformation creates textual representations (using Unicode characters) of the various above mentioned musical elements. Depending on the fonts' Unicode support on the local machine, not all characters might display correctly.

```

<text x="1.25cm" y="2.25cm" class="key-signature"
  offset="0" class="text">&#x266F;</text>

<text x="2.3cm" y="2.7cm" class="time-signature"
  offset="0">4</text>
<text x="2.3cm" y="3.5cm" class="time-signature"
  offset="0">4</text>

<text x="0.1cm" y="3.75cm" class="cleff"
  offset="0">&#x0AA2;</text>

<text x="3.3cm" y="3.75cm" class="note"
  startOffset="0">&#x2669;</text>
<text x="4.3cm" y="3.5cm" class="note"
  startOffset="0" >&#x2669;</text>
<text x="5.3cm" y="3.25cm" class="note"
  startOffset="0">&#x2669;</text>

```

Figure 2 finally illustrates the result, rendered as an [SVG](#) image.



Figure 2. Scale of G ([ScaleOfG.svg](#))

## 2.6. Summary

A working model that creates reliable musical representation at a simple level was developed (prototype). It illustrates that the design and implementation of such a system is a possibility, although mathematically extremely challenging. Especially the transformation from a one-dimensional character stream ([MusicXML](#)) into two- and three-dimensional graphical spaces is complex if not impossible. The research undertaken shows that the theoretical concept of transforming [MusicXML](#) into [SVG](#) is not only feasible, moreover they have a wide range of future applications.

The Network for Inclusive Distance Education ([NIDE](#)) project [[UT02](#)] is examining the existing [MusicXML DTDs](#) to determine which will provide the best foundation for multimodal, accessible rendering of content, with the hope of developing an integrated group of utilities to seamlessly provide access to musical notation. The components being developed include tools for authoring and rendering music notation (from [MusicXML](#)), an accessible onscreen keyboard and an application for rendering notation in visual, audio and other alternative formats.

## 3. BioML2SVG (Richard Maher, 2001)

### 3.1. Abstract

The BioML2SVG [XSL](#) transformations generate (two-dimensional) [SVG](#) images from a Biopolymer Markup Language ([BioML](#)) data file. Using the accompanied Java application, the user can select a specific gene and Deoxyribonucleic Acid ([DNA](#)) fragment for conversion. Each image can be customised with additional information such as location numbers, secondary structure elements (e.g. alpha helix), amino acid properties and 3D structural elements (e.g. disulfide bonds).

### 3.2. Introduction and Background

Data from genetics projects such as the Human Genome Project [[NCBI03](#)] are publicly available from databases worldwide. Navigating the [DNA](#) sequences by means of the raw data files, which are offered in a number of [XML](#) vocabularies, is nearly impossible without the help of graphical representations and/or dedicated user interfaces. The use of [XML](#) and related technologies from the [XML](#) family imposes alternatives to proprietary GENOM viewers. This student project focuses on two aspects of data visualisation. Firstly, the creation of a Graphical User Interface ([GUI](#)) which a biochemist could use to browse through a database of genetics files to improve data accessibility. Secondly, and more importantly for inter-media transformations, it will demonstrate image representations of genetics data in order to aid the biochemist in examining and interpreting the information.

Genes are defined as nucleic acid [DNA](#), a large organic molecule composed of millions of sub-units called nucleotides. Each nucleotide consists of a sugar, a base and a phosphate group. The sugar and phosphate group perform structural roles whereas the bases carry the genetic information. This information is encoded in the order of the four different chemical bases, namely Adenine (A), Guanine (G), Thymine (T) and Cytosine (C). [DNA](#) exists as two long, paired strands spiralled into a double helix, which closely resembles a twisted rope ladder. The two strands which make up [DNA](#) are linked together by the base pairs. Each connection is made of exactly two bases with the condition that Adenine can only bond with Thymine and Guanine can only attach to Cytosine. [DNA](#) can create an exact copy of itself by separating its rope ladder formation between the bases. Free floating nucleotides then pair up with the corresponding bases, which results in two identical [DNA](#) structures and cell division can take place.

Sequences on the [DNA](#) strand specify how to build proteins that will be needed by the cell. Proteins are made from a repertoire of 20 amino acids. The linear sequence of these amino acid residues in a particular polypeptide chain is referred to as the "primary structure" of the polypeptide. The [DNA](#) template specifies this sequence. Due to bond angles, distances and hydrogen bonding of neighbouring amino acids, several repeating structures were found in proteins. These 'secondary structures' include the alpha helix, the beta pleated sheet and beta turn. Tertiary structures occur when certain physical attractions between alpha helices and pleated sheets create the proteins unique shape by twisting due to interactions with water and distant amino acids.

### 3.3. Source: BioML

Databases around the world store huge amounts of genetic data. Many allow users to query this information via the Internet and receive results in [XML](#) format. There are several [XML](#) formats in use, including the [BioML](#) [[Bio99](#)], Genome Annotation Markup Element (GAME) and Bioinformatic Sequence Markup Language (BSML).

The [XML](#) vocabularies BSML and GAME were developed in order to display genetic information in a specialised but proprietary browser. Both approaches include a number of elements and attributes to define formatting and display properties. This prevents the vocabulary from being flexible and independent. On the other hand, [BioML](#) describes bio- molecule information logically, excludes presentation information and therefore makes this language more applicable for this project.

The following simple example of a gene is composed of a very short oligonucleotide sequence. It was taken from the sequence of the *Drosophila melanogaster* gene for ubiquitin [Bio99].

```
<?xml version="1.0"?>
<bioml>
  <gene>
    <dna start="1" end="41">
      GCAGCGACGACC
      <dstart at="13">ATG</dstart>
      TCCGGCGCCACCGAG
      <dend at="30">TAG</dend>
      TCGGGCTC
    </dna>
  </gene>
</bioml>
```

### 3.4. Image Design

The secondary structure of DNA and Proteins is a two-dimensional (2D) visualisation of the element's three-dimensional (3D) structure. Each of the four bases Adenine, Thymine, Cytosine and Guanine is represented by a distinctively coloured and shaped icon. The pairing of Thymine and Adenine, as well as Cytosine and Guanine is described by complementary faces of the shapes. The image initially displays 75 sequence pairs; SVG viewer functionality such as panning and zooming lets the user access the hidden parts. BioML does not tag secondary structural information for DNA.

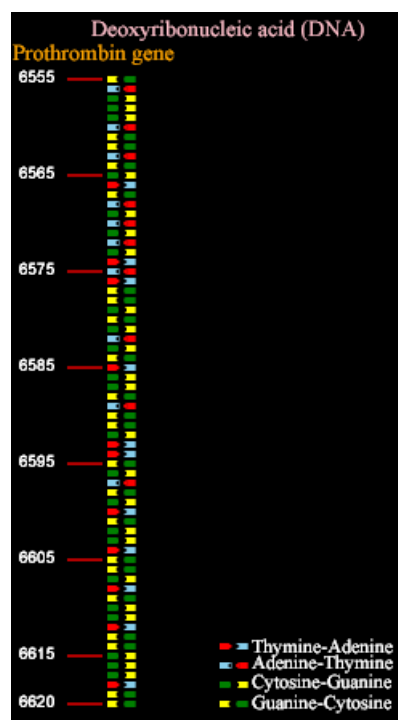


Figure 3. DNA Sequence (DNA.svg)

The protein's 3D structure is vital in producing its biological function, thus displaying its secondary structure, described by tags in the BioML file, is important to the biochemist. Peptides within the protein consist of a long linked chain of amino acids. 3D structural information is not requirement, but if contained within the BioML file, the application identifies the three main components alpha helix, beta leaded sheet and beta turn. Depending on the user preferences, these structural components are marked in the peptide SVG illustration with different background colours. Other important elements include cross links between chains or part of chains, so called disulfide bonds, which are shown in the image by connecting lines between the participating amino acids.

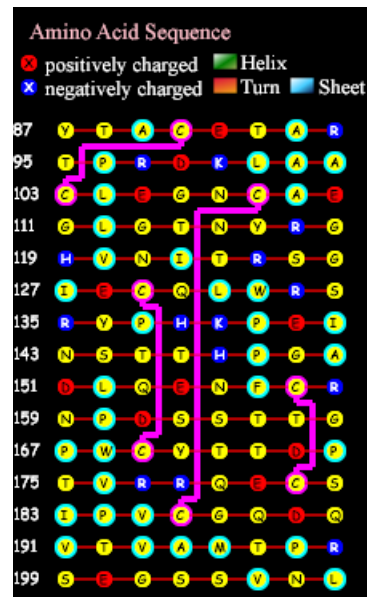


Figure 4. Amino Acid Sequence ([Peptide.svg](#))

A second view (genetic structure image) was chosen and implemented to provide the biochemist with a more abstract than detailed view of the gene. Instead of displaying each individual base or amino acid, key areas of interest such as introns, exons and domains are highlighted.

### 3.5. Implementation

A central requirement of the project was the design and implementation of a [GUI](#) that would allow opening a [BioML](#) file and comfortably accessing all its structural and semantic characteristics. Apart from the primary genetic data in textual format, the [BioML](#) file contains descriptive text and links to related websites, which should be incorporated into the [GUI](#). Sections of the textual genetic data should be easily selectable by the user and transformed into images, preferably [SVG](#). CSIRO's [\[Csi01\] SVG](#) toolkit was integrated into the final application to display the [SVG](#) images.

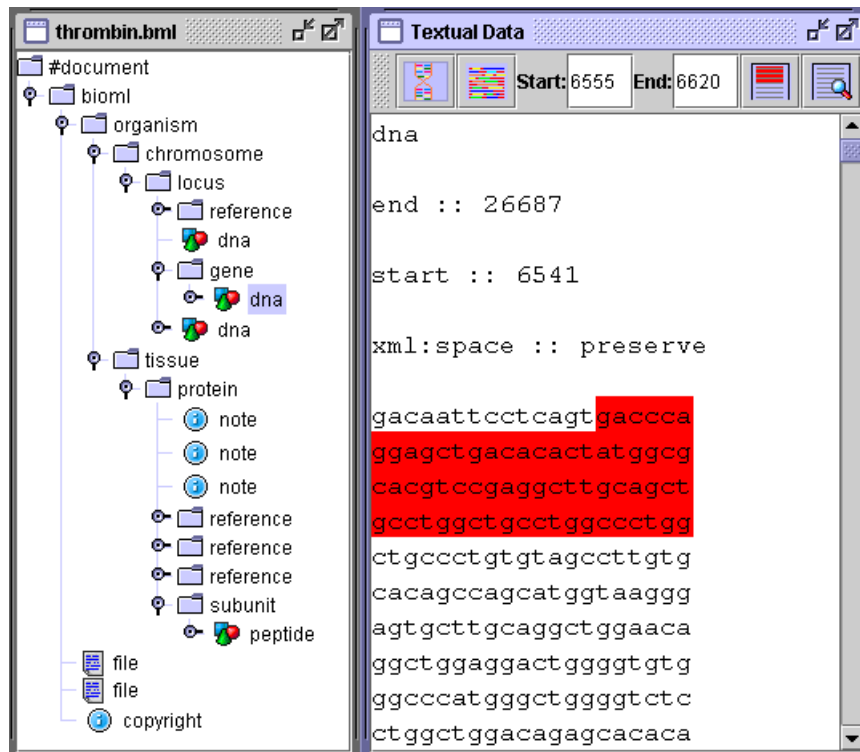


Figure 5. DNA selection

### 3.6. Summary

The [SVG](#) image created by the [BioML](#) application addresses two main shortfalls of a non-graphical browser. Firstly, the sequence data represented by a long list of letters is difficult to scan, especially in term of searching and pattern recognition. Secondly, the genetic structure image overcomes the inadequacy of navigating through hundreds of structured elements in a treeview. It provides a graphical overview of the genetic structure, including parent-child relationships. 'There is a great advantage in visualising the genome as it allows the human eye and brain to interpret patterns' [GuJa00] and different colours and shapes improve the ability to spot patterns while panning though the picture. As for the amino acid sequence picture, the user defines the chain area and properties to be displayed allowing for cluster searches such as positive amino acids or alpha helices.

## 4. CALS2SVG (Jamie Brohan, 2001)

### 4.1. Abstract

The CALS2SVG application creates instant [SVG](#) images from tables using the Computer Aided Logistic Support, Continuous Acquisition & Life-Cycle Support ([CALS](#)) vocabulary. A number of user-specific properties can be set before the conversion process. Due to the complexity of generating charts from a [CALS](#) tabular file format, the author decided in favour of a (proprietary) Java Transformation Engine ([JTE](#)) rather than a "plain" [XSL](#) transformation.

### 4.2. Introduction and Background

The availability of highly functional text-processors and spreadsheets offers dynamic and on-the-fly conversion of tabular data into diagrams 'at your fingertips'. Changes within the table are reflected immediately in the charts; Chart-types, -orientation and additional information such as axes and data captions can be altered by a click. But, tabular data is often kept in a proprietary format, which makes data exchange only possible by export and import of the table structure, formulas and values. The open and widely accepted [SGML/XML](#) vocabulary [CALS](#) has been in use for a long time, firstly in combination with [SGML](#) and more recently with [XML](#) documents. Nevertheless, a graphical representation of tabular data in [CALS](#) format is not known of.

### 4.3. Source: CALS

**CALS** tables were first defined by the US Department of Defence (**DoD**) for the interchange of documentation between the **DoD** and its subcontractors, in the procurement, production and support of new weapons systems. Since then, the **CALS** table model found its way into many document formats, e.g. docbook (**SGML**) and docbookx (**XML**) [Oas03]. In the past, the costs of managing paper (e.g. technical manuals) within the military have been substantial. The following example code shows a fragment of a **CALS** formatted table, representing the first two rows, including a header row and a first row of actual data.

```
<?xml version="1.0"?>
<table>
  <tgroup cols="9">
    <thead>
      <row>
        <entry colname="colFish" />
        <entry colname="colMon">Monday</entry>
        <entry colname="colTue">Tuesday</entry>
        <entry colname="colWed">Wednesday</entry>
        <entry colname="colThu">Thursday</entry>
        <entry colname="colFri">Friday</entry>
        <entry colname="colSat">Saturday</entry>
        <entry colname="colSun">Sunday</entry>
      </row>
    </thead>
    <tbody valign="top">
      <row>
        <entry colname="colFish">Plaice</entry>
        <entry colname="colMon">12.4</entry>
        <entry colname="colTue">15.9</entry>
        <entry colname="colWed">21.7</entry>
        <entry colname="colThu">39.45</entry>
        <entry colname="colFri">27.3</entry>
        <entry colname="colSat">12.1</entry>
        <entry colname="colSun">8.325</entry>
      </row>
      [...]
    </tbody>
  </tgroup>
</table>
```

According to the **CALS** specification, a table can contain header and footer rows as well as information about the leftmost and rightmost columns to cater for all possible table layouts and designs. This flexibility seems to be an advantage from a table designer's perspective; nevertheless, it makes linear table transformations (i.e. column by column and row by row) very complicated.

### 4.4. Image Design

This application was developed to create four typical diagrams, namely Histograms (Figure 6), Line Graphs (Figure 8), Scatter Plots and Pie Charts (Figure 9), from data described in **CALS** tables. The types can be divided into two subcategories, namely block- /line-charts and pie-charts. Before the image creation process, the user is asked to select properties such as chart type, orientation and the number of rows/columns to include.

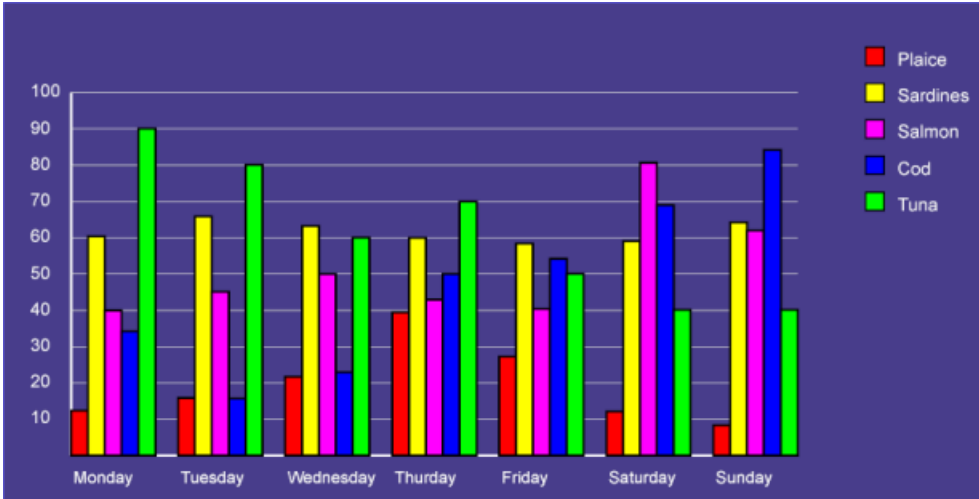


Figure 6. Barchart (weekdays horizontal) ([barchart1.svg](#))

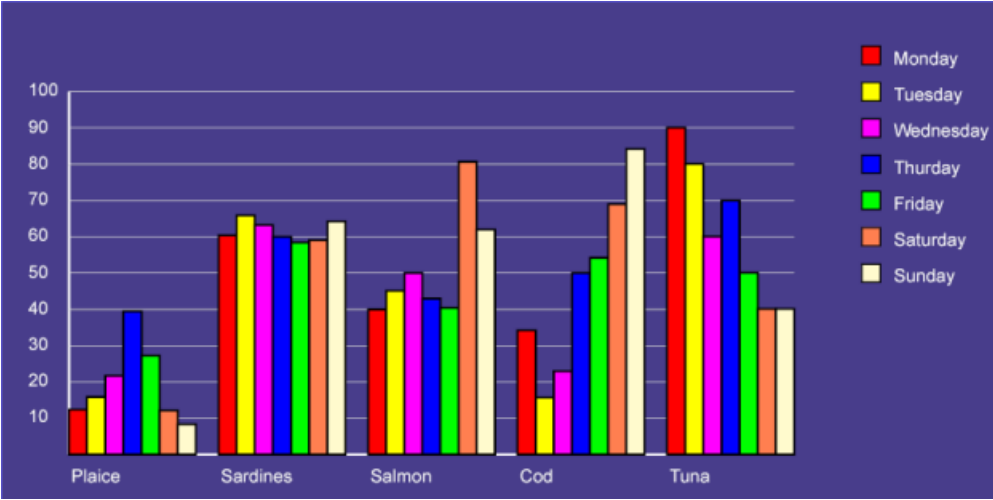


Figure 7. Barchart (weekdays as third dimension) ([barchart2.svg](#))

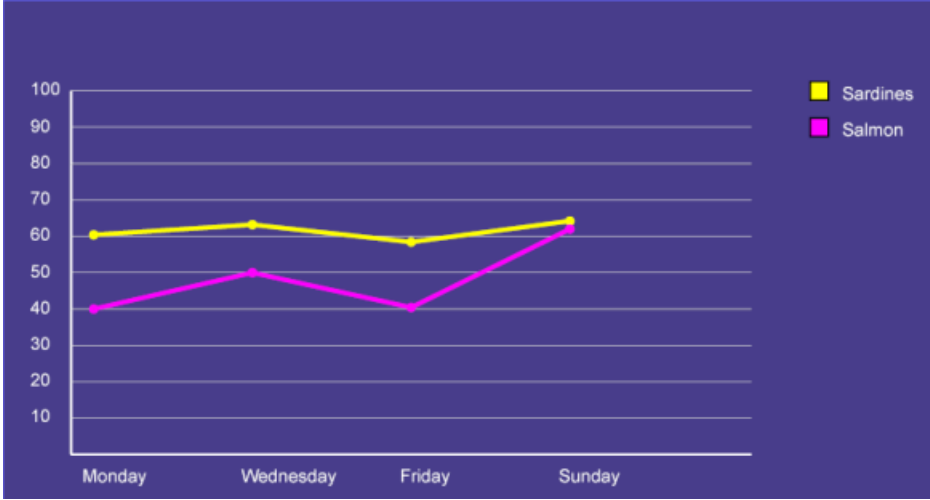
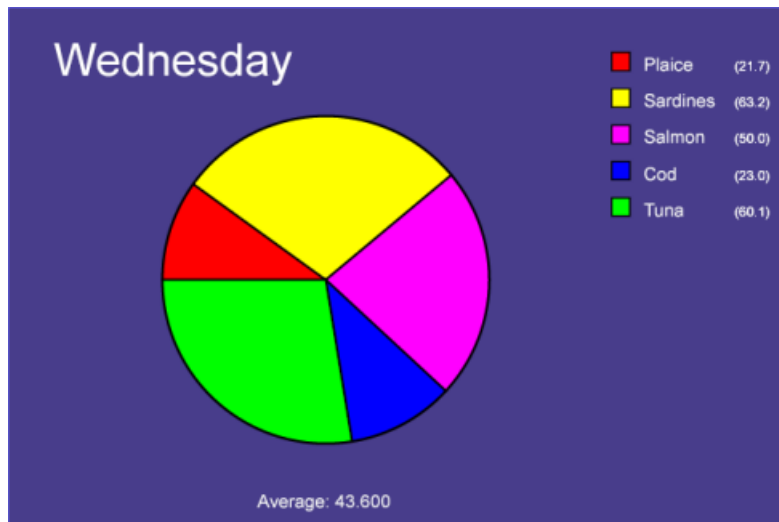


Figure 8. Linechart (Selection of Columns and Rows) ([linechart.svg](#))



**Figure 9. Piechart (Wednesday) ([piechart.svg](#))**

## 4.5. Implementation

Due to the (above mentioned) flexible nature of [CALs](#) tables including row and column spanning, the Document Object Model ([DOM](#)) parser with an in-memory and tree-structured representation of the data was the preferred choice. Using a linear Simple API for XML ([SAX](#)) parser (or [XSL](#) transformations) would have added additional complexity to the project, which would have been almost impossible to handle. With the [DOM](#) Application Programming Interface ([API](#)) available, random access to the data is possible. The assumption that header rows can only appear at the top and leftmost column resulted in a normalisation and simplification of table descriptions.

This project is implemented as a Java standalone application. It reads and parses the source [XML](#) document containing the [CALs](#) data. The application creates an [SVG](#) image file with a diagram specified by the user in terms of the type (line-/pie-chart) and source (selection of row and column numbers). The integrated [SVG](#) viewer from CSIRO [[Csi01](#)] is used to display the resulting image within the application.

## 4.6. Summary

Summarising can be said that a [CALs](#) to [SVG](#) transformation is an easy and straight-forward process. Nevertheless, [CALs](#) table allow for a flexible and dynamic representation of tabular data, often not following linearly the expected table layout, which makes transformations using [SAX](#) or [XSL](#) more complex than initially expected. Using a [DOM](#) parser and associated techniques overcomes this difficulty.

Data visualisation using images has always been a great benefactor in presenting complex information models and understanding data relations. Although [XML](#) is pure syntax and declared 'human readable', understanding or navigating an [XML](#) document is a different matter. Generic [XML](#) document visualisations include hierarchical trees and 'tagged' text, using different shapes and colours. However, depending on the semantic, more specialised data visualisations are necessary for adequate comprehension. As shown in this paper, this includes common chart images for tabular data, sheet music notation for musical data and proprietary [2D](#) representations for genetic data.

Summarising can be said that [XML](#) based vocabularies in general provide an ideal base for data visualisation. Nearly all of today's software products support [XML](#) processing in one or another way and visualising [XML](#) vocabularies by means of transforming it into 'easy-to-view' end-user formats, such as Extensible Hypertext Markup Language ([XHTML](#)) for text or [SVG](#) for images can be achieved in a number of ways. One option is the use of basic [XSL](#) transformations. Although the easiest and quickest method, complex 'non-linear' conversions such as e.g. mathematical transformations might be more comfortably solved using a [DOM](#) or [SAX](#) interface from Java or by adding proprietary ??? extension functions.

# Bibliography

- [Bio99] The Biopolymer Language BioML, Working Draft, in: Introduction to Bioinformatics, <http://www.bio-ml.com/BIOML/>, 2001.
- [Clo02] Close, N.J., Recordare: Internet Music Publishing, Recordare LLC, <http://www.recordare.com/>, 2002.
- [Cov02] Robin, C., The XML Cover Pages: XML and Music, <http://www.oasis-open.org/cover/xmlMusic.html/>, 2002.
- [Csi01] CSIRO SVG viewer, <http://www.cmis.csiro.au/svg/>, 2001.
- [Goo02] Good, M., MusicXML: An Internet-friendly format for sheet music, <http://www.idealliance.org/papers/xml2001/papers/html/03-40-05.html>, December 2002.
- [GuJa00] Guerrinia, V.H., Jackson, D., Bioinformatics and Extended Markup Language (XML), in: Online Journal of Bioinformatics, Volume 1 (1): 12-21, 2000. <http://www.cpb.uokhsc.edu/ojvr/ojbspecials2001/sub1abs.htm>.
- [Hew02] Hewlett, W.B., Beyond MIDI, The handbook of musical codes, The MuseData representation of musical information, <http://www.ccarh.org/publications/books/beyondmidi/online/musedata/>, 2002.
- [Hum02] The Humdrum Toolkit, <http://www.lib.virginia.edu/dmmc/Music/Humdrum/>, 2002.
- [Mon02] Montgomery, L., <http://www.4ml.org/>, 2002.
- [Mou02] Mounce, S., NIFF Homepage, <http://www.student.brad.ac.uk/srmounce/niff.html>, 2002.
- [NCBI03] National Center for Biotechnology Information, The Human Genome Resources, <http://www.ncbi.nlm.nih.gov/genome/guide/human/>, 2003.
- [Oas03] Oasis Open, Docbook SGML/XML, <http://www.oasis-open.org/committees/docbook/>.
- [Rec02] Good, M., Recordare: MusicXML, <http://www.musicxml.org/>, 2002.
- [Sax03] The Saxon XSLT processor, <http://saxon.sourceforge.net/>, 2003.
- [Sch02] Schiettecatte, B., A format for virtual orchestras, Vrije Universiteit Brussel, Version 0.5, Draft, <http://www.qorchestra.sourceforge.net/FlowML.pdf>, 2002.
- [Sib02] Sibelius, <http://www.sibeliusmusic.com/>, 2002.
- [Uni03] Unicode Musical Symbols, Range: 1D100-1D1FF, <http://www.unicode.org/charts/PDF/U1D100.pdf>, 2003.
- [UT02] University of Toronto, Music Notation Project, Network for Inclusive Distance Education, <http://nide.snow.utoronto.ca/music/Musicindex.html>, 2002.

# Glossary

2D	two-dimensional
3D	three-dimensional
API	Application Programming Interface

ASCII	American Standard Code for Information Interchange
BioML	Biopolymer Markup Language
BSc	Bachelor of Science
CALS	Computer Aided Logistic Support, Continuous Acquisition & Life-Cycle Support
CCARH	Centre for Computer Assisted Research in the Humanities
DNA	Deoxyribonucleic Acid
DoD	US Department of Defence
DOM	Document Object Model
DTD	Document Type Definition
GUI	Graphical User Interface
HTML	Hypertext Markup Language
JTE	Java Transformation Engine
MIDI	Musical Instrument Digital Interface
MusicXML	Music Markup Language
NIDE	Network for Inclusive Distance Education
NIFF	Notation Interchange File Format
SAX	Simple API for XML
SDML	Standard Music Description Language
SGML	Standard Generalized Markup Language
SVG	Scalable Vector Graphics
TCD	Trinity College Dublin
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

## Biography

### Benjamin Jung

Trinity College Dublin  
Department of Computer Science  
Dublin  
Ireland  
benjamin.jung@cs.tcd.ie

Benjamin Jung is a lecturer in Computer Science and Medical Informatics at [TCD](#). He is a regular speaker at [XML](#) conferences in Europe and the US. His main research interest are data visualisation, document architectures and [XML](#) technologies in the domains of medical informatics and electronic publishing. Since 1997, Benjamin has presented papers and chaired sessions at various Computer Science and Medical conferences.

He developed full-day [XML](#) tutorials and workshops that were given at conferences in Europe and the US. In 2000, Benjamin co-founded deepX Ltd (<http://www.deepx.com/>), where he holds positions of director and consultant.

**Jamie Brohan**

Trinity College Dublin  
Department of Computer Science  
Dublin  
Ireland  
jamie@braynet.ie

Jamie received a Bachelor of Science ([BSc](#)) in Computer Science from [TCD](#) in 2001.

**Richard Maher**

Trinity College Dublin  
Department of Computer Science  
Dublin  
Ireland  
richard.maher@loox.com

Richard received a [BSc](#) in Computer Science from [TCD](#) in 2001.

**Laura O'Shea**

Trinity College Dublin  
Department of Computer Science  
Dublin  
Ireland  
ljoshea50@hotmail.com

Laura received a [BSc](#) in Computer Science from [TCD](#) in 2002.

**Vincent Wade**

Trinity College Dublin  
Department of Computer Science  
Dublin  
Ireland  
vincent.wade@cs.tcd.ie

Vincent Wade is a lecturer in the Computer Science Department in [TCD](#). He is head of the Knowledge and Data Engineering Research group and director of the Centre for Learning Technology, [TCD](#). He received his BSc from University College Dublin, Ireland in 1987 and a MSc from [TCD](#), Ireland in 1991. Vincent leads a team of four academic and 20 research associates and postgraduates. He is active in the research of educational hypermedia, adaptive hypermedia and instructional systems development. He is chairman of [TCD](#)'s Learning and Technologies Strategy Committee and currently leads several European Union and industrial research projects in the area. He is author of over sixty technical papers in international conference and research journals. He is guest editor of 'Network Interoperability Journal', published by Baltzer scientific publications. He also holds the position of editor for the highly acclaimed IS&N conferences series (Communications section) for the last three years. He is also editor of two European Union Guidelines on the development of based Telecommunications Multimedia Information Systems. Successful European research project in e-Learning have included GESTALT (distributed learning resource retrieval and management system) and EASEL (adaptive hypermedia for personalised learning).