Development of Chemical Markup Language (CML) as a System for Handling Complex Chemical Content

Peter Murray-Rust[1], Henry S. Rzepa[2] and Michael Wright[2]

(1) School of Pharmaceutical Sciences, University of Nottingham, UK - (2) Department of Chemistry, Imperial College of Science, Technology and Medicine, UK - ***August 25, 2000***

We report the first fully operational system for managing complex chemical content entirely in interoperating XML-based markup languages. This involves the application of version 1.0 of Chemical Markup Language (CML 1.0) and the development of mechanisms allowing the display of CML marked up molecules within a standard web browser (Internet Explorer 5). We demonstrate how an extension to including spectra and reactions could be achieved. Integrating these techniques with existing XML compliant languages (e.g. XHTML and SVG) results in electronic documents with the significant advantages of data retrieval and flexibility over existing HTML/plugin solutions. These documents can be optimised for a variety of purposes (e.g. screen display or printing) by single XSL stylesheet transformations. An XML schema has been developed from the CML 1.0 DTD to allow document validation and the use of data links. A working online demonstration of these concepts termed ChiMeraL, containing a range of online demonstrations, examples and CML resources such as the CML DTD and Schema has been associated with this article via the supplemental (supplemental.xml) material.

Introduction

The World-Wide-Web was originally conceived as a collaborative tool for scientists, allowing the rapid distribution and publication of results and greatly improved online communication. It has become increasingly common for academic papers, databases and knowledge archives to be made available in the form of *hypertext markup language* (HTML) pages, this being an efficient way of making them widely and globally available. [1] However, the absence of mechanisms in HTML for directly handling chemical information such as e.g. molecular structures and hence the difficulties in automatically recognising and extracting chemical data from HTML pages limit their potential. [2 3 4] Our solution to this problem involved the development and extension of a *chemical markup language* (CML) [5] and techniques to allow the display of molecules, spectra and reactions within a web browser. [6] These chemical objects can be seamlessly integrated into existing textual markup to create complete electronic documents.

Existing Solutions - HTML and plugins

The text formatting language HTML and its interface, the web browser are familiar tools to many scientists. HTML was originally designed to display plain text on a computer screen, with markup **'tags'** (formally known as **elements**) primarily supporting the formatting of textual objects (paragraphs, tables, headings, lists etc.). Inter-document linking is achieved by **hyperlinking** (via the **anchor**, **applet**, **embed** or **object** elements) which also allows content in the form of bitmapped images (normally *gif, jpg* and *png*), and traditional chemical data files to be transcluded.

HTML documents rely significantly on human processing for chemical content interpretation and error detection since the language syntax is too flexible to be reliably interpreted by machines,

especially the early versions of hypertext markup language (1.0, 2.0, 3.2, 4.0 and their commonly used non standard derivatives, *collectively referred to in this article as 'HTML'*). No mechanisms exist in these versions of HTML to allow them to be validated with respect to the document integrity and there is very little markup of context (to indicate the meaning and relevance of each information component). Non textual data normally requires the use of browser **'plugins'** (small, platform specific helper programs) or Java **'applets'** (non platform specific programs, that run on a 'virtual machine' on top of the web browser). As an example, a present day online chemical paper might consist of HTML text, static bit-map images and diagrams and molecular structures transcluded from an external **'legacy'** data file such as e.g. MDL *Molfile* [7] or the Brookhaven *Protein Databank (PDB)* formats [8] which are intended to be visualised using an appropriate plugin (supplementary figure 1) . The syntax of these external files is heavily implied and requires specialist knowledge to interpret. This severely limits the re-usability of the rapidly growing amount of high-quality chemical information available on Web pages and a significant amount of time and more importantly information can be wasted converting between these files.

Since HTML [9] was not originally designed to support non textual data, there is no built in mechanism to allow molecular structures to be shared between a plugin and the parent document. As a result, the external data files become isolated from the text and from each other. This reduces inter-operability and makes it significantly harder to identify information within these files, while at the same time retaining their context. Browsing through HTML pages using a conventional search engine requires significant human interaction in order to identity useful chemical information. If the information was labelled by context, far more could be handled automatically, e.g. filtering and ignoring identical data from different sources. The main problems with existing HTML/file transclusion solutions include:

- The lack of a clear separation of content from the formatting or presentation in the HTML document. For example, HTML defines a superscript element - `<sup>` - which can ambiguously be used to either define a cited reference or footnote, or in a molecular formula as an isotope, charge, unpaired electron, oxidation state, state symmetry or multiplicity. Or it can be used simply for stylistic effect. Similarly, chemical content and meaning can often be expressed using Greek characters, which in HTML are considered a stylistic attribute best expressed using a formal stylesheet declaration. This means that a molecular formula or constitution marked up using HTML is effectively lost as far as machine readability is concerned.
- The use of legacy formats with implied syntax and the resulting multiplicity of standards. A single, human readable format is required, combining both textual and non textual information within a single document.

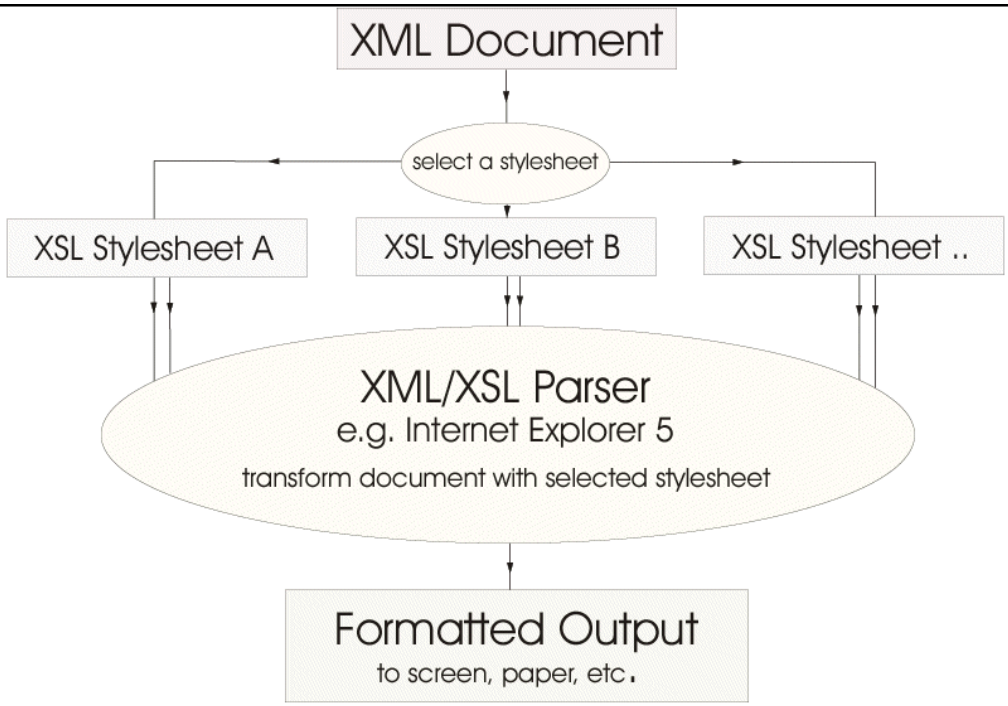
Additional considerations include scalability, validation, platform independence (particularly relevant online), portability and flexibility. [10] With the present cost of computer hardware and improvements in communication bandwidths, file size is significantly less of a concern, but many legacy formats are were designed, in the era of expensive computer memory, to be extremely terse and hence have a large amount of implied and hence essentially unavailable syntax.

Extensible Markup Language (XML)

In February 1998, the *World Wide Web Consortium* (W3C) published their recommendations for *extensible markup language* (XML) [9] as a successor to HTML. XML is not a markup language in its own right, instead it is a series of rules and conventions defining how to build discipline-specific markup languages (a so called 'meta-markup language'). XML is designed to allow the description of any structured data by the use of an user defined set of tags, or **elements** in XML

nomenclature, support precise declarations of the content of a document **and** context using explicit syntax. A defined set of elements is called an XML language and these languages are far more flexible than HTML, since they can be tuned to a particular subject.

Figure 1: *Data flow in an XML/XSL transform - an XML document can be transformed by one of many stylesheets depending on the required output.*



All XML compliant languages use the same syntax and can be manipulated using any XML compliant tools or applications. Information marked up using different languages can be easily combined into a single document and manipulated at the same time. These different languages; e.g MathML, CML, [11] SVG (*Scalable Vector Graphics*) are identified within the document by the use of discrete **namespaces** (vide infra) and are often concatenated using XHTML (HTML following XML rules). An online XML article might consist of XHTML text combined with e.g. SVG diagrams and CML structures or spectra.

Unlike HTML, XML elements do not describe the formatting of a document and hence a generic browser can only by default display the document and element tree structure. Instead the parser built into the browser uses an external **stylesheet**, which contains formatting instructions for each element in the XML source. This stylesheet is supplied by either the author or the reader and effectively maps the contents of each element to an output. This would normally be an XHTML page or some other text file but can also be to e.g. an Acrobat file. [12]

The older *cascading stylesheets* (CSS) is a standard commonly used for formatting HTML, and provides a mechanism for defining the formatting and layout instructions for an entire HTML document collection in a single stylesheet file. CSS works by allowing the author to redefine how an HTML tag will be presented, e.g. `<h1>` can be defined using a CSS stylesheet as meaning present this heading with e.g. size 11 font and red colour. This is useful when the style of an entire document collection needs to be changed, since it only requires editing one stylesheet rather than each HTML document. Much more sophisticated stylesheets can be written using a powerful

formatting, searching and scripting language **XSL** (*extensible stylesheet language*). In this article, we will describe the design and implementation of a range of chemically useful stylesheets.

Since any number of stylesheets can be applied to a single XML document, it can be displayed in a large variety of ways (Figure 1). The searching and scripting capabilities of XSL mean that element(s) can be picked out and displayed and/or manipulated according to their contents. Other information can be ignored if required, making data searches from large XML documents trivial. An article containing a mixture of text and chemical information can be scanned for e.g. spectra or numerical information using such a stylesheet and this information transformed and displayed using e.g. a suitable applet. Alternatively, a different stylesheet might be used to identify molecular structures and potentially calculate their properties.

Stylesheet transformations are performed using software called an XML/XSL **parser**, whichmay be a dedicated program or a component built into an existing web browser (as with Internet Explorer version 5 and higher). [13] Stylesheet selections can be defined by a URL (*uniform resource locator* from within the document or selected from a local collection using the parser. This allows a reader to use their own stylesheets which invokes their own preferences or resources available to them. This flexibility makes stylesheets very powerful and is exemplified by the ChiMeraL demonstration [6] where sample CML data files containing molecule and spectra can be transformed with a range of XSL stylesheets.

The use of a unified XML format reduces the time wasted in processing legacy files and avoids the risk of serious information loss due to poor semantics. This "plug-compatible" XML approach guarantees a document to be searchable, sortable, mergeable, and printable with minimal extra cost. Ideally all future document and publishing systems will be transferred to using XML languages.

Chemical Markup Language (CML 1.0)

The definition of **CML version 1.0** [5] was developed to carry molecules, crystallographic data and reactions using an XML language and for the first time offers a universal, platform and application independent format for storing and exchanging chemical information. As the first generation of CML, it outlines a variety of general purpose 'data-holder' elements and a smaller number of more specifically chemical elements (e.g. `<molecule>`, `<reaction>`, `<crystal>`) used to indicate chemical 'objects'. For example, a `<molecule>` will contain a `<list>` of `<atom>`s, which in turn have three `<float>`s specifying Cartesian coordinates for each atom.

This framework is flexible but leaves many areas open for later evolution. In particular, CML provides no default conventions for labelling data elements and puts few restrictions on element ordering. This is intentional as CML is designed to be generic and contains minimal preconceptions as to the type of chemical information that will be stored using it. Standardisation and conventions are the concern of the community and will be co-ordinated by publications and projects such as this one.

In the work presented in this article, we have developed rigorous markup procedures for small and medium molecule structures. We have included spectral information using an additional element `spectrum>` which is not part of CML 1.0. Exploratory work has been carried out into `<reaction>` markup and **data linking** within a CML/XHTML document. Simply defining such procedures is of

limited use and a variety of XSL template 'fragments' have been developed. These allow molecules, spectra and reactions to be displayed using a variety of existing Java applets and can be used for the construction of stylesheets able to format mixed XML documents containing chemical information. These stylesheet fragments and examples are offered as online resources in the form of a fully operational demonstration of CML parsing within a web browser known as ChiMeraL. This is available via the supplemental information associated with this article, and in more extended form at the http://www.xml-cml.org/chimeral/ (http://www.xml-cml.org/chimeral/) Web Site.

Results and Discussion

The markup syntax used for XML languages is superficially similar to that used for HTML, but significantly stricter. The markup is expressed using **ASCII** text and consist of data units contained within nested tag pairs (identifiable as an open `<tag>` and a close `</tag>`). In HTML, these tags are often used to describe formatting properties, e.g. `<font size="+1" color="#FF0000">here is some <b>bold</b> text</font>`. XML languages strictly separate out formatting from semantic markup, and describe the meaning and context of data as opposed to its formatting e.g. `<float title="melting point" units="degC">238</float>`). In XML nomenclature, tag pairs are called 'elements' (not to be confused with chemical elements) and the content of an element consists of any attributes of that element, their values, and any text or sub-elements nested within the tag pair. Text or sub elements within a *parent* element are described as *children* of that element and *siblings* of each other.

XML Rules

The most important rules of XML syntax are as follows. A document that follows these is described as '*well formed*'.

1. All tags must be closed to form an element e.g. `<formula>C8 H10 N4 O2</formula>` and must be correctly nested. This is in contrast to HTML, where incorrect nesting is normally ignored.
2. 'Empty' elements with no children may be written using the shorthand `<link/>` = `<link></link>`
3. All attribute values must be within quotes (") e.g. `<string title="CAS">` (in this article, we use the convention; `@title` = an attribute named 'title')
4. Element and attribute names are case sensitive and are normally written using lower case to distinguish them from non XML compliant HTML.
5. Code comments are of the form `<!-- comment -->`and are ignored by the parser.

Figure 2: *Simple XML document*

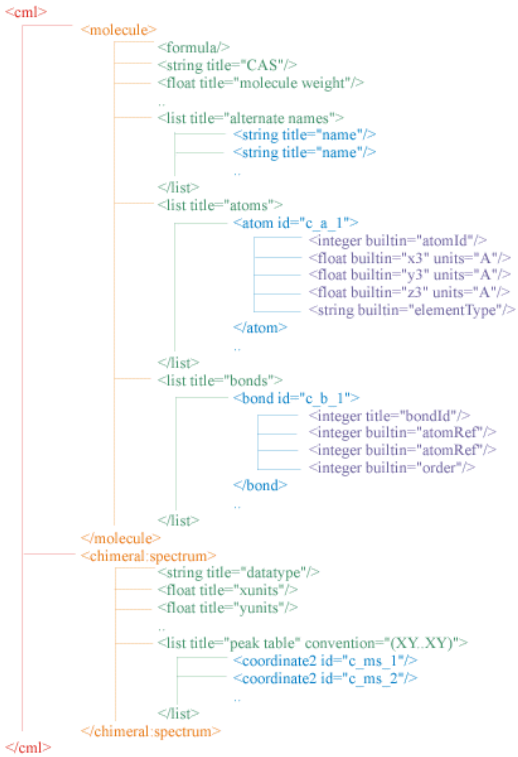
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
        href="http://www.ch.ic.ac.uk/chimeral/publications/document.xsl" ?>

<document title="Simple example of an XML document" id="xmldoc_simpleEx"
        xmlns:xhtml="http://www.w3.org/1999/xhtml"
        xmlns:cml="x-schema:http://www.xml-cml.org/chimeral/cml_schema_ie_02.xml">

<xhtml:p>This is a block of text formatted using XHTML (HTML following
         XML rules). Normal <xhtml:i>italic</xhtml:i> and <xhtml:b>bold</xhtml:b>
         elements can be used as can more complex <xhtml:font face="Helvetica"
         color="#000080">fonts and colours.</xhtml:font> Following this is a block
         of molecular information marked up using CML. Additional blocks using
         other XML languages could be added at will.<xhtml:/p>

<cml:cml title="Properties of Caffeine" id="cml_caffeine">
           <cml:molecule title="caffeine" id="mol_caffeine">
              <cml:formula>C8 H10 N4 O2</cml:formula>
              <cml:string title="CAS">58-08-2</cml:string>
              <cml:float title="molecule weight">194.19</cml:float>
              <cml:float title="melting point" units="degC">238</cml:float>
             <cml:string title="comments">White powder or white glistening needles
                 usually melted together. LIGHT SENSITIVE</cml:string>
                  <cml:list title="alternate names">
                        <cml:string title="name">1,3,7-Trimethylxanthine</cml:string>
                        <cml:string title="name">1,3,7-Trimethyl-2,6-dioxopurine</cml:string>
                        <cml:string title="name">7-Methyltheophylline</cml:string>
                   </cml:list>
        </cml:molecule>
</cml:cml>
</document>
```

An example of a short XML document is given in Figure 2 which contains both XHTML and CML marked up information. Those familiar with HTML will recognise the syntax but will find the element names unfamiliar. Notice how information can be stored both as an element child and also as the value of an attribute.

The structure of this document is best visualised as a branched tree, similar to the directory structure on a hard-drive. It has a single 'root' (in this case `<document>`) with a number of sub-elements (`<p>` and `<cml>`, ignoring the prefix) and sub-sub-elements, defining the logical structure of the document. Each branch in the tree may contain information either as a text child or as attribute values. Each component of information is therefore uniquely described by a list of its ancestors and this provides a convenient method for stylesheets to navigate the XML tree (called *pattern matching*).

From Figure 2 we can see that this document contains a chemical formula, CAS number, molecular weight, melting point and three alternate names for caffeine. In addition, there are two **processing instruction** tags (line 1 and 2) that use the syntax `<?tag_name_here?>`. The first, `?xml version="1.0"?>` indicates that the document has been written to follow XML rules. The second, `<?xml-stylesheet .. ?>` gives the URL of the default stylesheet. Unless given over-riding instructions (e.g. by a reader who wishes to use their own stylesheet) the parser will automatically obtain this stylesheet and use it to **transform** (format) the XML document. This transformation can be carried out locally (i.e. using the Web browser), remotely on a web server or using a combination of the two.

Figure 3: *The CML element 'tree'. The molecule contains three lists, one of alternate names, one of atoms with their coordinates and one of bonds and their atom references (the atoms defining the bond)*



When writing an XML document, it is usual to collect data of similar types and organise this into a suitably logical structure. For example; the formula for caffeine is stored in branch \document\cml\molecule\formula. Figure 3 shows a simplified CML document tree.

Namespacing

The attribute:

`xmlns="x-schema:http://www.xml-cml.org/chimeral/cml_schema_ie_02.xml"`

of `<document>` in Figure 2 is called an XML **namespace declaration**. [9] XML rules allow for the mixing of XML languages within the same document and there is no mechanism to prevent potentially conflicting element names being used in two or more different languages. Indeed such a mechanism would greatly restrict development of XML. Instead, the author assigns each language a different **namespace prefix**, using the syntax: `<namespace:element>` and `@namespace:attribute`. The choice of prefix for any particular language is left to the author and these need only be unique over the document in question. A namespace declaration (normally found on the root element) is then used to index each prefix to a globally unique URL (often the language development site). Alternatively, it can point to a file containing a description of the language, as it does for the CML namespace.

There are two types of namespace declaration. The simplest is of the form `@xmlns:mynamespace="http://www.uniqueURL.com"` and declares any elements with a mynamespace: prefix as belonging to a language defined at 'http://www.uniqueURL.com'. It is assumed that attributes share the namespace of their element unless declared otherwise. A default namespace declaration; `@xmlns="http://www.anotherURL.com"`, results in the element containing the declaration and all ancestors of that element, belonging to "http://www.anotherURL.com" unless otherwise stated. This avoids having to use a prefix in front of a large number of similar elements.

Figure 4: *Namespaces used in an XHTML/CML document*

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="document.xsl" ?>
<mrw:document
        xmlns="http://www.w3.org/1999/xhtml"
        xmlns:mrw="x-schema:http://www.xml-cml.org/chimeral/mrw_schema_01.xml"
        xmlns:chimeral="x-schema:http://www.xml-cml.org/chimeral/spectrum_schema_ie_01.xml">

<mrw:abstract>We report the ..</mrw:abstract>

<mrw:chapter id="chap_introduction" title="Introduction">
<mrw:index>1</mrw:index>

<p>The <b>World-Wide-Web</b> was originally developed as a collaborative tool
for scientists, allowing for rapid distribution and publication of results and
greatly improved communication by email. It has become increasingly common for
academic papers and results to be posted online, this being an extremely ..</p>

</mrw:chapter>

<cml title="cml data block" id="cml_moldata"
        xmlns="x-schema:http://www.xml-cml.org/chimeral/cml_schema_ie_02.xml">
           <molecule title="caffeine" id="mol_caffeine">
        ..
        </molecule>
        <chimeral:spectrum title="Furan, tetrahydro-" id="spect_furantetrah_ir_1">
        ..
        </chimeral:spectrum>
</cml>

</mrw:document>
```

Figure 4 shows a document mixing XHTML, published CML 1.0, [5] ChiMeraL [6] CML (which also includes `<spectrum>`) and a simple document structure language. The four namespace declarations are as follows:

- **xmlns="http://www.w3.org/1999/xhtml"**
  All elements default to the XHTML namespace (defined by the W3C) unless otherwise stated. There is no need to write stylesheet entries to transform this language, instead it can be designed to pass these unrecognised elements directly to the browser (which intrinsically implements default instructions on how to format XHTML). This 'bypass' avoids unnecessary stylesheet templates and makes stylesheet development significantly quicker.
- **xmlns:mrw="x-schema:http://www.xml-cml.org/chimeral/mrw_schema_01.xml"**
  This language describes the logical structure of the document e.g. `<abstract>`, `<chapter>`, `appendix>`
- **xmlns="x-schema:http://www.xml-cml.org/chimeral/cml_schema_ie_02.xml"**
  This declaration is found on `<cml>` and changes the default namespace for that element and its children, anything within `<cml>` is CML unless declared otherwise
- **xmlns:chimeral="x-schema:http://www.xml-cml.org/chimeral/spectrum_schema_ie_01.xml"**
  One element in `<cml>` is not in the default `cml:` namespace, this is `<spectrum>` and requires a namespace of its own - `chimeral:`. As with the two previous declarations, the URL points to a XML schema for this language.

Schema and DTD

The URL in a namespace declaration may point to a file describing the XML language. This can be a simple HTML page but a more powerful method is to point to a file defining the language using a machine readable syntax. This allows the parser to automatically **validate** (check) the document against the language(s) it uses, before transforming it with a stylesheet. The parser can also be informed if elements belong to a particular data-type (string, integer, etc.) or of 'special' attributes exist (e.g. unique @id or @href links) and validate these also. If a document is correct XML (well formed) but does not comply to the definitions of the languages it uses, it is described as non-valid and can not be parsed. For example, `<float type="CAS">58-08-2</float>` is well formed XML but is not valid CML since `<float>` may only contain a single floating point number.

There are two alternative standards for writing machine readable descriptions. In both cases, this comprises a list of all valid elements and attributes for the language and declarations of restrictions on element content and child ordering. The older standard is called **DTD** (*document type definition*) and uses a specialised machine readable (but not particularly human readable) syntax. DTDs have wide support amongst XML tools and parsers but are being superseded by XML **schemas**. These are written using an XML language defined by the W3C and can therefore be parsed with the same tools as XML documents and XSL stylesheets. Schemas allow more sophisticated descriptions of valid element trees and they are significantly more human readable. Parsers can recognise a schema referenced in a namespace declaration and can automatically validate the document against it. [13] If a document consists of a mixture of XML languages, the use of a schema for each allows more complex manipulations (for example the linking of data between languages). Since the modular structure of XML allows (and indeed expects) the addition and extraction of information 'objects' using many languages, this flexibility is very important.

The CML schema (version 0.2) was developed from the published DTD as part of this project. Two versions have been made available. The first is a generic XML schema. The second is optimised for Internet Explorer 5 and includes additional data-type and linking declarations. A full

description of this schema can be found in the supplemental information for this article as Appendix C. Since `<spectrum>` is not included in the CML 1.0 DTD, an additional namespace and schema have been constructed for it. It is expected that additional elements will be required as further areas of chemical markup are developed and it is suggested that this approach be maintained. Once a particular addition has been reviewed and accepted by the community, it can then be incorporated into a future version of published CML and included in the schema. The CML schema is available at http://www.xml-cml.org (http://www.xml-cml.org) [11]

Platform Choice

There are a wide variety of XML parsers presently under development, [14] many written using the programming language **Java**. [15] Java programs are run within a software 'virtual machine' and not directly at the operating system level. Since this software has been written for most operating systems (Windows, MacOS, Linux, Unix, etc.), Java programs can be ported between these platforms with minimal effort. It was the aim of this project to build a fully working online demonstration of real time CML parsing and transforming using the ubiquitous and familiar interface of a web browser. This restricts functionality (Java parsers are significantly more advanced) but allows CML examples to be run using software already installed in a large number of computers and the requires minimal knowledge of XML from the user.

The choice of browser is currently (September 2000) restricted. Later versions of Windows Internet Explorer 4.x have some limited XML support and this is much improved in version 5 onwards. It now includes full **DOM** (*Document Object Model*), XSL and XQL (*extensible query language*) support. Netscape 6 (based on the open source Mozilla project) supports XML but is currently only able to display it using CSS stylesheets. It is hoped that support for XSL will be included in the next release. No other common browsers yet support XML to any reasonable degree. This being the case, Internet Explorer 5.x is presently the only suitable client for online XML applications.

Chemical Markup Language

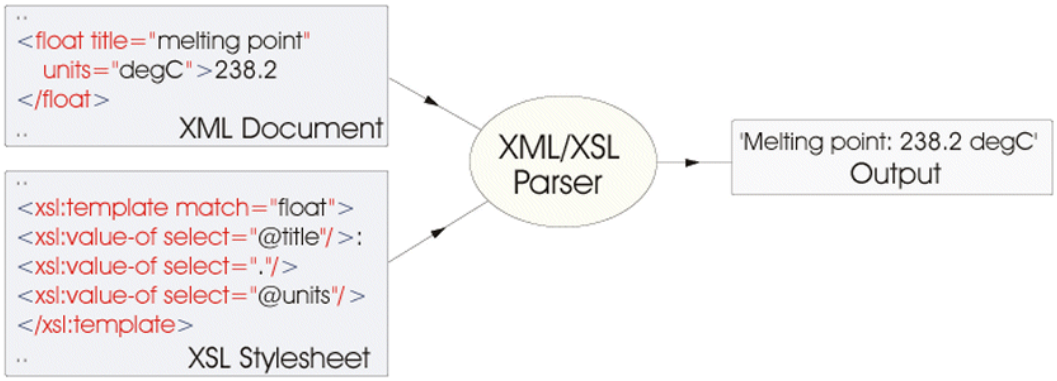
In keeping with the XML philosophy, CML is designed to be extensible and will be published as a series of drafts. The first draft (CML 1.0) [5] contains the core components of this new language and defines two main types of markup: **data holders** and **chemical components**.

Data Holders

These are generic elements designed to contain the standard data types required by chemistry. These include strings, integers, floating point numbers and a series of array and matrix elements. These do not usually contain sub-elements, only text, and their context is described by the values of their attribute (particularly `@title`); e.g. `<float title="melting point"` `238.2</float>` contains the melting point data: 238.2 degrees Centigrade.

The values of these attributes are not defined in the DTD or schema so any data can be contained simply by choosing them appropriately. This avoids the need to define large numbers of specific elements within CML. To include a melting point, an additional element could be defined `meltPoint>` but this would not be recognised and formatted by a generic CML stylesheet. It is possible to build new stylesheets including the new element, but this reduces their compatibility. The better approach is to use existing `<float>` and add `@title="melting point"` to give it a label. A stylesheet can then display this as 'melting point: ...' without having to recognise the attribute value directly (Figure 5).

Figure 5: *Simple stylesheet transformation*



Chemical Components

Chemical components represent objects of chemical interest such as molecules, atoms, bonds etc. and these elements normally contain a number of data holders. CML 1.0 defined the basic components for molecular structure and crystallographic markup but the extension of CML to other areas of chemistry requires the addition of additional elements. The number of these should be minimised as much as possible and new elements must be described as an extension to CML with their own DTD or schema. In most cases each chemical component should also be supplied with a unique identity `@id`. A molecule containing chemical property information is shown in Figure 6 In order to simplify the figures, processor instructions, `<document>` tags and namespace declarations have not been included in the following examples.

Figure 6: *A simple CML block containing property information*

```
<cml title="tetrahydrofuran" id="cml_THF">
        <molecule title="Furan, tetrahydro-" id="mol_thf_1">
                <formula>C4 H8 O</formula>
                <string title="CAS">109-99-9</string>
                <float title="molecular weight">72.1066</float>
                <string title="ACX">I1001473</string>
                <string title="DOT">UN 2056</string>
                <string title="RTECS">LU5950000</string>
                <float title="melting point" units="degC">-108.3</float>
                <float title="boiling point" units="degC">65.0</float>
                <float title="specific gravity">0.886</float>
                <float title="water solubility" convention="g/100 mL at 23 degC">30.0</float>
                <string title="comments">
                Colorless liquid with an ether-like odor detectable at 2 to 50 ppm.
                HYGROSCOPIC</string>
                <list title="alternate names">
                        <string title="name">THF</string>
                        <string title="name">1,4-Epoxybutane</string>
                        <string title="name">Butylene oxide</string>
                        <string title="name">Cyclotetramethylene</string>
                        <string title="name">tetramethylene oxide</string>
                        <!-- etc -->
                </list>
        </molecule>
</cml>
```

Two other important elements used in CML are `<list>`, which allows for collections of similar data holders and `<link>` which supports linking between data in the same, or different documents. Common attributes are `@title`, `@convention`, `@unit` and `@builtin`. The latter uses a series of fixed values predefined in the DTD (e.g. xyzFract, elementType, atomId) but values for the rest depend on convention and published examples. Attempts should be made to use values compatible with those used by others in the field and it is expected that published examples and online projects will help coordinate this. In a similar manner while the DTD defines the element names that should be used when marking up a chemical object, the tree structure is left to convention. It is likely that many of these conventions will be based on existing file formats (e.g. the Molfile convention), simply because this makes conversions to and from CML easier. At a later stage we would expect CML to become a standard export format for chemical software packages.

Displaying CML Using a Stylesheet

The CML block in Figure 6 contains only trivial CML property information. A more complex example might contain molecular structures (2D and 3D), spectra (ir, uv/vis, nmr or ms) and possibly a reaction scheme. Techniques for marking up and displaying these more complex components are covered later, but to illustrate how a XSL stylesheet is written a simple example will be explained in full (Figure 7).

XSL follows XML rules and therefore requires a namespace declaration in the same way as any other XML document. Internet Explorer recognises `http://www.w3.org/tr/WD-xsl` [13] as a XSL namespace declaration and the ability to understand this language is built into the parser. XSL uses a limited set of '**commands**' (approximately 15) which can be recognised by their xsl: prefix. All text and all elements not part of the XSL namespace are passed directly to the output (in this case, the browser). Two exceptions are, comments (`<!-- comment -->`) which the parser ignores and the contents of the `<xsl:eval>` element which are used for scripting and calculations within the stylesheet.

The stylesheet contains a series of `<xsl:template>`s which in turn consist of a mixture of XSL commands and template text/tags. At the simplest level, each template refers to the particular element in the XML source that matches the value of the template `@match`. Therefore, `xsl:template match="molecule">` marks the start of a template for the `<molecule>` element.

The parser navigates iteratively through the XML document tree, starting at the document root and selecting matching templates at each branching. Navigation through the tree is controlled by the `<xsl:apply-templates select="` value and branches can be parsed or ignored as required. Matching and selection of elements is handled by a powerful pattern matching language which refers to the document tree using a Unix like syntax (a full description of pattern matching can be found on the W3C [9] and MSDN [13] web-sites).

Figure 7: *Simple XSL stylesheet (XHTML table) transforming contents of Figure 6*

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/tr/WD-xsl">
<!-- root -->
<xsl:template match="/">
<xsl:apply-templates select="*"/>
</xsl:template>

<!-- match <cml> and build XHTML page -->
<xsl:template match="cml">
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 strict//EN">
<html>
        <head><title>
        <xsl:value-of select="@title"/> - <xsl:value-of select="@id"/></title></head>
        <body>
        <xsl:apply-templates select="molecule"/>
        </body>
</html>
</xsl:template>

<!-- build xhtml table -->
<xsl:template match="molecule">
        <!-- Pull out @id="" etc -->
        <table><tr>
                <td>Molecule ID:</td>
                <td>Formula:</td>
                <td>CAS:</td>
                <!-- etc. -->
                </tr><tr>
                <td><xsl:value-of select="@id"/></td>
                <td><xsl:value-of select="formula"/></td>
                <td><xsl:value-of select="*[@title = 'CAS']"/></td>
                <!-- etc. -->
                </td>
                </tr><tr>
                <td>Alternate Names:</td>
                <td colspan="6">
                <xsl:for-each select="list[@title = 'alternate names']/string[@title='name']">
                <xsl:value-of select="text()"/>, </xsl:for-each>
                </td>
                </tr>
                <!-- etc -->
        </table>
</xsl:template>
</xsl:stylesheet>
```

The example shown here (Figure 7) contains three templates. The first template (`@match="/"` - the document root) starts the parsing process at the first level element in the document (`<cml>` in this example). A similar template is required for all XSL stylesheets. The second template `@match="cml"`) builds the start of a skeleton XHTML page and put the values of `@title` and `@id` within the TITLE tags. `<xsl:apply-templates select="molecule"/>` instructs the parser to select child element(s) `<molecule>` and to find a template for it. Once that branch has been completed, the parser returns to the second template and completes the XHTML page:

```
<html><head><title> Furan,tetrahydro- mol_thf_1 </title></head><body><!--results of transforming <molecule> here --></body></html>
```

The third template recognises the chemical properties within the `<molecule>` and formats their values to a simple XHTML table. Note how the alternate names are handled; the contents of `xsl:for-each select="list[@title = 'alternate names']/string[@title='name']">` is repeated each time the pattern value of `@select` matches an element in the XML document. The output from the complete stylesheet is shown in Figure 8

Figure 8: *Displaying CML block as a simple XHTML table*

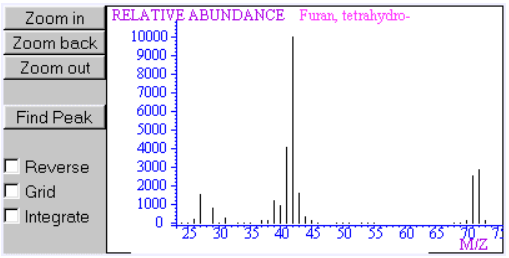
Molecule ID:	Formula:	CAS:	ACX:	DOT:	RTECS:	MW:
mol_tetrahydrofuran_1.mol	C4H8O	109-99-9	I1001473	UN 2056	LU5950000	72.1066
MP: -108.3 degC	**BP:** 65 degC	**Spec. Gravity:** 0.886		**Water Sol.:** 30 g/100 mL at 23 degC		
Alternate Names:	THF, 1,4-Epoxybutane, Butylene oxide, Cyclotetramethylene, tetramethylene oxide,					
Comments:	Colorless liquid with an ether-like odor detectable at 2 to 50 ppm. HYGROSCOPIC					

Display of CML using Applets

More complex chemical data cannot be easily displayed using text and a more sophisticated solution is required. A hypothetical CML-aware parser might have this functionality built in, being able to recognise chemical components and display them appropriately. Using existing parsers, the problem is analogous to that encountered with normal HTML pages. Browser plugins and Java applets are used to provide additional display functionality and a wide range of of these add-on programs have been developed, some of them very advanced. An efficient solution is to use a combination of stylesheets and applets to display the CML components contained within a mixed XML document. Using existing software is preferable as this allows faster development of a working application and is in the spirit of XML's reusability.

Figure 9: *Applet code produced by the stylesheet and its display*

```
<applet name="jSpec" code="Visua.class"
        width="400" height="250">
<param name="SOURCE" value="
        ##TITLE=Furan, tetrahydro-
        ##JCAMP-DX=4.24
        ##DATA type=MASS SPECTRUM
        ##CAS REGISTRY NO=109-99-9
        ##EPA MASS SPEC NO=61352
        ##MOLFORM=C4H8O
        etc.">
</applet>
```



Both plugins and Java applets are primarily intended to read external data files, normally in a legacy format (.mol, .pdb, .dx etc.) and to embed a graphical display of this data within an HTML page. In both cases, special tags are required:-

```
<embed name="Plugin" src="datafile.mol" width="300" height="300">
```

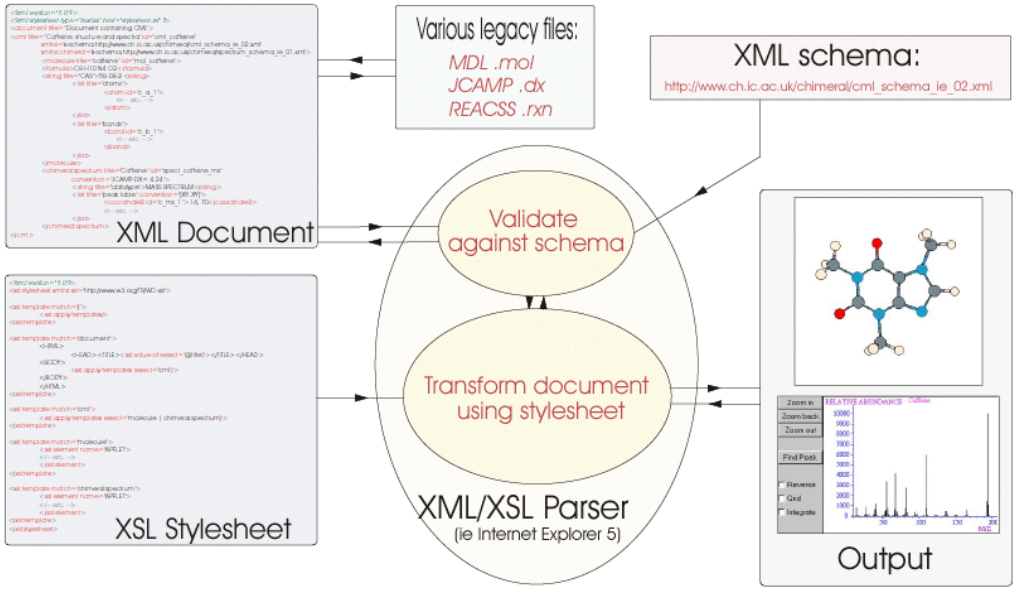
```
<applet name="Applet" code="applet.class" width="300" height="300">
```

CML is normally embedded into a mixed XML document, and therefore it is not easily accessible by a plugin. Applets are more flexible and can receive data in a wider variety of ways, which makes them significantly more useful for XML display. Stylesheets can be written to recognise CML components and build appropriate `<applet>` tags to display them. Since all applets are designed to read legacy formats, the data within the components must also be transformed (called a CML to legacy conversion). The transformed data string can then be passed to the applet in two different ways (depending on which type the applet supports). The easiest technique is to place the string within the `<param>` declaration for an applet.

An alternative technique is to place the transformed string within a hidden `<input>` element. These are normally used within the (X)HTML `<form>` element and cannot be seen by the reader. Here it becomes a convenient holder for the transformed string and JavaScript is then used to call a public function on the applet and pass it the contents of the `<input>` element. The JavaScript

can be triggered by an button or when the page has completely loaded into the browser. This is less direct than using <param>s but allows more powerful interactions between the page and the applet since other functions can also be made accessible to JavaScript. This approach is used for the JME applet described in the next section. [16]

Figure 10: *Data flow in an XML/XSL system*



A selection of applets have been chosen depending on their suitability for XML integration. It is desirable that the applets be small (faster downloads), flexible, easy to use and open source since this allows changes to be made to the applet code if required. XSL 'template fragments' have been built for each of these applets and for several common CML to legacy conversions (available from http://www.xml-cml.org/chimeral (http://www.xml-cml.org/chimeral)). By combining these stylesheets with applets and the schema, complete CML applications can be built (Figure 10).

Molecular Structures

The most commonly used file formats for molecular structures are the MDL Molfile [7] and the Brookhaven protein database format. As its name suggests, the Brookhaven format is normally used for large molecules or proteins, but its use for expressing small and medium sized molecules (< 500 atoms) along with the Molfile format is increasingly common.(supplementary figure 3) .

MDL Molfile

The Molfile uses a strict **new-line delimited** structure with a heavily implied syntax. As a result it is very sensitive to the amount of white (empty) space on each line. This can cause serious problems when converting to and from this format from CML. The format has the advantage of being extremely terse, relevant when it was developed but much less so now. Other file formats tend to use rather similar syntaxes (e.g. the XYZ format lacks the additional atom columns and the bond data).

CML: molecule

CML molecular structures are reminiscent of the Molfile format but with the data being fully marked up and explicitly defined. Implied syntax is reduced to a minimum and conventions are declared and not assumed. The aim is to make each chemical component (coordinate, atom, bond etc.) completely separable from the rest of the document, allowing components to be easily added or removed without destroying the document tree. Since an XML document might contain a very large number of molecules (examples containing over 600 molecules have been built), each component requires a unique @id. A large collection of Molfiles files can be easily converted to CML and then concatenated to a single XML document. A stylesheet can then pick out a single molecule and display its structure by searching against `@id`.

Figure 11: *Example of small molecule markup*

```
..
<molecule title="ethanol" id="mol_ethanol">
        <formula>C2 H6 O</formula>
        <string title="CAS">64-17-5</string>
        <!-- etc -->
        <list title="atoms">
                <atom id="ethanol_a_1">
                        <integer builtin="atomId">1</integer>
                        <float builtin="x3" units="A">1.0303</float>
                        <float builtin="y3" units="A">0.8847</float>
                        <float builtin="z3" units="A">0.9763</float>
                        <string builtin="elementType">C</string>
                </atom>
                <!-- eight further atoms -->
        </list>
        <list title="bonds">
                <bond id="ethanol_b_1">
                        <integer title="bondId">1</integer>
                        <integer builtin="atomRef">1</integer>
                        <integer builtin="atomRef">2</integer>
                        <integer builtin="order" convention="MDL">1</integer>
                </bond>
                <!-- seven further bonds -->
        </list>
</molecule>
..
```

As with the Molfile, lists of atoms and bonds are used. Atoms contain either 3 (x3, y3, z3) or 2 (x2, y2) Cartesian coordinates, an atom type and an 'atomID'. This is a non-unique label referenced by the 'atomRef' values of the bond and are **not** the same as the atom's unique @id (hence `integer builtin="atomRef">1</integer>` refers to an atom of `<integer`

`1</integer>` not `@id="1"`). The same applies to 'bondId'. While not optimal, this greatly eases the conversion of CML to and from legacy formats. Coordinate units are no longer implied and since this molecule was converted from a Molfile file, the bond order convention is 'MDL'.

Additional information (formula, CAS number) is not extracted from the Molfile file, since although such information might be supplied using comments, it is not automatically identifiable. In our case it was acquired from various online databases such as ChemFinder. [17] The author may decide to include any information they wish in this way, simply by adding additional elements at appropriate places in the CML document.

Java Molecular Editor (JME)

The **JME** (*Java Molecular Editor*) applet is a simple and small (31K) 2D editor, [16] incorporating a SMILES calculator (a widely used standard for producing text descriptions of molecular structure) [18] . Rather than including filters for legacy file formats at the expense of increasing the applet size, JME reads a simple coordinate and bond string. The syntax used is unique to JME and can be a problem for XHTML solutions since there is no way to dynamically calculate it from a standard data file. Using CML, this problem is solved with a trivial stylesheet transformation. This makes the applet excellently suited for our purposes (supplementary figure 4) .

Figure 12: *JME stylesheet fragment - reformatted for clarity*

```
..
<!-- match <cml> and build HTML page -->
<xsl:template match="cml">
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 strict//EN">
<html>
        <head><title>
        <xsl:value-of select="@title"/> - <xsl:value-of select="@id"/></title></head>
        <body onLoad="document.JME.readMolecule(jmeoutput.value)">
        <xsl:apply-templates select="molecule"/>
        </body>
</html>
</xsl:template>
..
<!-- match molecule and display using JME -->
<xsl:template match="molecule">
        <applet code="JME.class" name="JME" archive="JME.jar" width="400" height="300">
        You have to enable Java and JavaScript on your machine !
        </applet>
        <!-- hidden form element contains the JME data string -->
        <xsl:element name="input">
                <xsl:attribute name="name">jmeoutput</xsl:attribute>
                <xsl:attribute name="type">hidden</xsl:attribute>
                <xsl:attribute name="value">
                        <!-- select last atom and calculate its number -->
                        <xsl:for-each select="list/atom[end()]" xml:space="preserve">
                                <xsl:eval>formatIndex(childNumber(this), "1")</xsl:eval>
                        </xsl:for-each>
                        <!-- select last bond and calculate its number -->
                        <xsl:for-each select="list/bond[end()]" xml:space="preserve">
                                <xsl:eval>formatIndex(childNumber(this), "1")</xsl:eval>
                        </xsl:for-each>
                        <!-- select each atom and extract coordinates and atom type -->
                        <xsl:for-each select="list/atom" xml:space="preserve">
                                <xsl:value-of select="*[@builtin = 'elementType']"/>
                                <xsl:value-of select="*[@builtin = 'x3']"/>
                                <xsl:value-of select="*[@builtin = 'y3']"/>
                        </xsl:for-each>
                        <!-- select each bond and extract atom refs and bond order -->
                        <xsl:for-each select="list/bond" xml:space="preserve">
                                        <xsl:value-of select="*[@builtin = 'atomRef']"/>
                                <xsl:value-of select="*[@builtin = 'order']"/>
                        </xsl:for-each>
                </xsl:attribute>
        </xsl:element>
</xsl:template>
..
```

The string read by JME uses the syntax;

n_atoms n_bonds ***{****atomic_symbol x_coord y_coord****}per atom {****atom1 atom2 ***}per bond***

This string is built by the stylesheet and held in hidden `<input>` labelled with the name 'jmeoutput'. A Java Script 'onLoad' function is also added to the XHTML `<body>` tag and this is called when the page has been completely loaded into the browser. The script takes the content of 'jmeoutput' and passes it to a public 'readMolecule' function in the applet which proceeds to display it. Stereoisomers and simple reactions can also be displayed using JME but have not yet been incorporated into ChiMeraL.

Some additional XSL commands should be explained at this point; `<xsl:element>` and `xsl:attribute>` create new XHTML/XML tags in the output and are used as an alternative to writing the tags directly. For example, `<xsl:element name="input"><xsl:attribute name="name">jmeoutput</xsl:attribute></xsl:element>` is equivalent to `<input name="jmeoutput"/>`. They are used to build complex tags that might otherwise cause the stylesheet to become invalid XML.

The `<xsl:eval>formatIndex(childNumber(this), "1")</xsl:eval>` command uses Internet Explorer specific functions to calculate the position of a element with respect to its siblings and return this as a formatted number. It is used in almost all the stylesheet fragments, normally to count the total numbers of atoms and bonds in the molecule. Non Internet Explorer stylesheets would use different functions depending on the parser they are designed for.

JMol 3D Viewer

Jmol [19] is a stand alone Java application (not an applet) being developed by the Open Science project. It is designed to read and display molecules using a variety of legacy formats. Its functionality is similar to the Chime plugin but it is open source and platform independent. An experimental applet has been released and this has been adapted for use in this project. A stylesheet converts the CML `<molecule>` to a *% delimited .xyz string*, which is then stored and transferred to the applet in a manner similar to JME. The '%' delimiters must be added because the converted string can not replicate the line breaks required in the .xyz format (similar to Molfile). The applet has been slightly rewritten to recognise % as equivalent to a new line.

Problems also occur with white space; both the XYZ and Molfile formats require strict space handling to ensure their data remains in straight columns. Both formats use *right* aligned text, in contrast to XML/HTML standard *left* alignment:

```
right: C  -1.7560   0.0000   0.3080%         left: C   -1.7560   0.0000   0.3080%
        C -10.3600   3.0760   5.2880%          C   -10.3600   3.0760   5.2880%
        H   8.1400 -11.1520   7.5080%          H   8.1400   -11.1520   7.5080%
```

In both cases, the stylesheet must check each coordinate and add a varying amount of white space depending on its size and sign (+ or -). As with JME, the converted string is contained within a named `<INPUT>` and then passed by `onLoad="document.JMolApplet.setModelToRenderFromXYZString(jmoloutput.value,'T')"` to the applet.(supplementary figure 5)

Figure 13: *Jmol stylesheet fragment.*

```
..
<xsl:template match="molecule">
        <!-- build applet -->
        <applet code="org.openscience.miniJmol.JmolApplet.class"
                name="JMolApplet" archive="JmolApplet.jar"
                width="300" height="300">
        You have to enable Java and JavaScript on your machine !
        </applet>
        <xsl:for-each select="id(@idref)">
        <!-- build INPUT containing the xyz source -->
        <xsl:element name="input">
                <xsl:attribute name="name">jmoloutput</xsl:attribute>
                <xsl:attribute name="type">hidden</xsl:attribute>
                <xsl:attribute name="value">
                <!-- start convertion to .xyz -->
                <!-- select last atom and add its number -->
                <xsl:for-each select="list/atom[end()]"xml:space="preserve">
                <xsl:eval>formatIndex(childNumber(this), "1")</xsl:eval>
                </xsl:for-each>%
                <!-- add title -->
                <xsl:value-of select="@title"/> from CML%
                <xsl:for-each select="list/atom" xml:space="preserve">
                        <xsl:for-each select="string[@builtin='elementType']">
                                <xsl:value-of select="."/>
                                <!-- single letter elements with require an additional space  -->
                                <xsl:if test=".[. = 'H']"> </xsl:if>
                                <xsl:if test=".[. = 'B']"> </xsl:if>
                                <xsl:if test=".[. = 'C']"> </xsl:if>
                                <xsl:if test=".[. = 'N']"> </xsl:if>
                                <xsl:if test=".[. = 'O']"> </xsl:if>
                                <!-- etc -->
                        </xsl:for-each>
                <!-- for each atom, extract coord adding white space as required -->
                <xsl:if test="float[@builtin='x3' and . $lt$ 10]"> </xsl:if>
                <xsl:if test="float[@builtin='x3' and . >= 0]"> </xsl:if>
                <xsl:value-of select="float[@builtin='x3']"/>
                <!-- repeat for y3 and z3 -->
                %
                </xsl:for-each>
                <!-- finish convertion to .xyz -->
                </xsl:attribute>
        </xsl:element>
</xsl:template>
..
```

Marvin and Structure Drawing Applet (SDA)

Marvin is an commercial applet produced by Chemaxon [20] and is free to academic users. It is a wire frame 3D viewer able to accept Molfile data using external files, as a `<param>` or via JavaScript. The applet is configurable and includes a 2D editing function. **SDA** (*Structure Drawing Applet*) is a freeware editor developed by ACD Labs and also accepts Molfile data via `<param>`. [21] (supplementary figure 6)

Figure 14: *Marvin stylesheet fragment - reformatted for clarity*

```
..
<xsl:template match="molecule">
        <applet code="MView" archive="marvin.jar" width="300" height="300">
                <xsl:element name="param">
                        <xsl:attribute name="name">mol</xsl:attribute>
                        <xsl:attribute name="value">
<!-- start conversion to .mol -->
<xsl:value-of select="@title"/>\
  from CML\
\
<!-- select last atom, calculate its number and required white space -->
<xsl:for-each select="list/atom[end()]/integer[@builtin='atomId']" xml:space="preserve">
        <xsl:if test=".[text() $lt$ 100]"> </xsl:if>
        <xsl:if test=".[text() $lt$ 10]"> </xsl:if>
        <xsl:for-each select="../../atom[end()]">
                <xsl:eval>formatIndex(childNumber(this), "1")</xsl:eval>
        </xsl:for-each>
</xsl:for-each>
<!-- select last bond, calculate its number and required white space -->
<xsl:for-each select="list/bond[end()]/integer[@title='bondId']" xml:space="preserve">
        <xsl:if test=".[text() $lt$ 100]"> </xsl:if>
        <xsl:if test=".[text() $lt$ 10]"> </xsl:if>
        <xsl:for-each select="../../bond[end()]">
                <xsl:eval>formatIndex(childNumber(this), "1")</xsl:eval>
        </xsl:for-each>
</xsl:for-each>\
<!-- for each atoms, extract coord and type, with required white space -->
<xsl:for-each select="list/atom" xml:space="preserve">
         <xsl:if test="float[@builtin='x3' and . $lt$ 10]"> </xsl:if>
        <xsl:if test="float[@builtin='x3' and . >= 0]"> </xsl:if>
        <xsl:value-of select="float[@builtin='x3']"/>
        <!-- repeat for y3 and z3 -->
        <xsl:for-each select="string[@builtin='elementType']">
                <xsl:value-of select="."/>
                <!-- elements with single letter names require an additional space -->
                <xsl:if test=".[. = 'H']"> </xsl:if>
                <xsl:if test=".[. = 'B']"> </xsl:if>
                <xsl:if test=".[. = 'C']"> </xsl:if>
                <xsl:if test=".[. = 'N']"> </xsl:if>
                <xsl:if test=".[. = 'O']"> </xsl:if>
                <!-- etc. -->  0  0  0  0  0  0  0  0  0  0  0  0\
        </xsl:for-each>
</xsl:for-each>
<!-- for each bond, extract atomRefs and order, with required white space -->
<xsl:for-each select="list/bond" xml:space="preserve">
        <xsl:for-each select="integer[@title='atomRef']">
                <xsl:if test=".[. $lt$ 100]"> </xsl:if>
                <xsl:if test=".[. $lt$ 10]"> </xsl:if>
                <xsl:value-of select="."/>
        </xsl:for-each>
        <xsl:value-of select="integer[@builtin='order']"/>  0  0  0  0\
</xsl:for-each>
M  END
<!-- finished convertion to mol -->
                        </xsl:attribute>
                </xsl:element>
        You must have Java turned on!
        </xsl:element>
</xsl:template>
..
```

The stylesheets for these applets are very similar and use the <param> mechanism. As with Jmol, line delimitation is required for the Molfile and is built into both applets (Marvin uses "\" and SDA "|"). As with XYZ, handling white space is complex, since Marvin in particular is intolerant of errors. The following XSL fragment works for most small molecules. Experiments have shown that the CML and Marvin applet scales well and can display over 300 molecules on a modern computer.

Spectra

Spectra are regularly stored and exchanged using an electronic format and modern spectrometers can save their data directly to disk. Electronically stored spectra can be easily indexed, searched against or otherwise manipulated and large databases of spectra files exist online (e.g. the NIST Webbook). [22] As with molecular structures, a number of legacy formats are in use, the most common being JCAMP-DX [23] and SPC. [24] In order to investigate the X (extensibility) of XML, we investigated mechanisms for extending CML to incorporate spectral markup and to develop techniques for displaying it.

JCAMP-DX/CS

The JCAMP-DX format(supplementary figure 7) defines a series of conventions for the storage and exchange of spectral data. Files using this format are written using ASCII text and contain a series of *labelled data records* (**LDR**). Each LDR is delimited by a new line and a double hash:

```
##LDRname= value
```

While having a much simpler structure than XML, these LDRs can be considered as a form of markup with the LDR name (*data label*) being equivalent to the element name. This is unsurprising since the two formats share a number of important design criteria, such as extensibility, human/machine readable, flexibility and platform independent. Approximately 25 *reserved data labels* were originally defined for JCAMP-DX. These are intended to contain:

- Metadata - (e.g. ##ORIGIN, ##OWNER, ##DATE)
- Header information - (e.g. ##TITLE, ##DATA TYPE, ##BLOCKS, ##END)
- Required spectral parameters - (e.g. ##XUNITS, ##FIRSTX, ##MAXX)
- Optional spectral parameters - (e.g. ##RESOLUTION, ##DELTAX)
- Spectral data - (e.g. ##XYDATA, ##XYPOINTS, ##PEAK TABLE, ##PEAK ASSIGNMENTS)
- Equipment - (e.g. ##INSTRUMENT PARAMETERS)
- Sample information - (e.g. ##CAS NAME, ##MOLFORM, ##MP)

Important LDRs are #DATA TYPE which describes the type of spectrum and ##PEAK TABLE that contains the data points. A *variable list* label following ##PEAK TABLE describes how the data is tabulated. Data is grouped into XY or XYZ/XYM coordinates and separated by commas within the group, while groups are separated by semicolons or spaces. Any additional white space is ignored. Common variable lists are;

- **(XY..XY)**
  Grouped XY pairs.
  `23,102 19,89 15,87 ..`
- **(XYZ..XYZ) or (XYM)**
  Grouped XYZ or XYM (can be used to give multiplicity).
  `23,102,S 19,89,S 15,87,D ..`
- **(X++(Y..Y))**
  More complex. Each line starts with an X value, then a series of Y value at equal X spacings, so -
  `12 52 48 46 43 42`
  `22 43 38 36 35 31`
  is equivalent to -
  `12,52 14,48 16,46 18,43 20,42 22,43 24,38 26,36 28,35 30,31`

JCAMP-DX also allows the use of *user defined data labels*. These are distinguished by a $ prefix and are specific to a particular instrument, location, user etc. This effectively means that although

JCAMP is an extensible language, it does not include any procedure for systematically describing the meaning of an LDR. This seriously limits the use of user defined data labels beyond the original author. A number of JCAMP 'sub-languages' have been defined in an attempt to solve this issue. [25] [26] These contain lists of LDRs intended for particular areas of spectroscopy (e.g. UV/Vis, H1 NMR, MS). Particularly interesting is JCAMP-CS [27] which allows the addition of limited structural information. These JCAMP files are split into a number of separate **blocks**, each containing a different type of information (e.g. molecular structure, peak table and peak assignments). This information can be extracted and marked up as CML, and a JCAMP-CS to CML converter has been written for this purpose.

CML: spectrum

Since a well established (if non-XML based) markup language already exists, many technical difficulties have already been solved. Rather than develop new conventions for spectral markup, we have chosen to incorporate those used by JCAMP. It is expected that most users will wish to convert CML to and from JCAMP so this needs to be as easy as possible. Once converted, spectra can incorporated into XML documents and displayed using techniques similar to those used for molecules.

Comparing JCAMP with Figure 15 illustrates the approach used. A new element `<spectrum>` is defined as an extension to CML, and this is given a new namespace `chimeral:`. The contents of the JCAMP labelled data records are then mapped to a `<string>` or `<float>` with an appropriate @title value. Reserved LDRs (##TITLE, ##XUNIT etc.) are directly recognised and given lower case values to comply with XML conventions. User-defined or unrecognised LDRs can either be ignored or, more correctly, given `@title="LDRname"`. This avoids any information loss on converting JCAMP to CML.

Figure 15: *Example of spectral markup*

```
..
<chimeral:spectrum title="Furan, tetrahydro-" id="spect_furantetrah_ir_1"
        convention="JCAMP-DX= 4.24">
        <string title="datatype">MASS SPECTRUM</string>
        <string title="EPA">61352</string>
        <string title="origin">D.HENNEBERG, MAX-PLANCK INSTITUTE, MULHEIM, WEST GERMANY</string>
        <string title="owner">NIST Mass Spectrometry Data Center</string>
        <float title="xunits">M/Z</float>
        <float title="yunits">RELATIVE ABUNDANCE</float>
        <float title="firstx" convention="M/Z">24</float>
        <float title="lastx" convention="M/Z">73</float>
        <float title="xfactor">1</float>
        <float title="firsty" convention="RELATIVE ABUNDANCE">4</float>
        <float title="miny" convention="RELATIVE ABUNDANCE">3</float>
        <float title="maxy" convention="RELATIVE ABUNDANCE">9999</float>
        <float title="yfactor">1</float>
        <float title="npoints">32</float>
        <list title="peak table" convention="(XY..XY)">
                <coordinate2 id="furantetrah_ir_c_1">24, 4</coordinate2>
                <coordinate2 id="furantetrah_ir_c_2">25, 30</coordinate2>
                <coordinate2 id="furantetrah_ir_c_3">26, 171</coordinate2>
                <coordinate2 id="furantetrah_ir_c_4">27, 1545</coordinate2>
                <!-- 28 more coordinate2s -->
        </list>
</chimeral:spectrum>
..
```

The delimiters between data groups and between data *within* a group are only loosely defined in the JCAMP specifications. As a result, the syntax used for ##XYDATA and ##PEAKTABLE is very variable. In particular, while the (X++(Y..Y)) convention is common, it can be confusing for the non specialist and is difficult to store using XML elements. It is suggested that CML spectra should only use the (XY..XY) or (XYM)/(XYZ) conventions and with each pair or triplet be marked up

using <coordinate2> and <coordinate3> respectively. Data within these elements is separated by the use of a comma and a white space (', '). More complex conventions are deconvoluted during the JCAMP to CML conversion process, e.g. (X++(Y..Y)) is expanded to (XY..XY).

Although a large number of LDRs have been defined, we have chosen to include only a small number of generic ones here. If an author wishes to use additional LDRs, these need to be included into the converter and stylesheets.

JSpec - JCAMP and SPC display applet

Jspec is a small open source applet [28] designed to display JCAMP and SPC spectra files. It also incorporates a number of useful features, including zooming, peak finding and integration.

The stylesheet is similar to that used for the Marvin applet. CML is converted to a normalised JCAMP text string and incorporated into the applet's <param> declaration. Line delimitation is not required since JCAMP already has a ## separator, but the applet was slightly adapted to accept comma-separated ##PEAKTABLE and ##XYDATA groups. (supplementary figure 8)

Figure 16: *Jspec stylesheet fragment - reformatted for clarity*

```
..
<!-- you must include the namespace in the template match -->
<xsl:template match="chimeral:spectrum">
<!-- build applet -->
        <applet code="Visua.class" width="400" height="250">
        <xsl:element name="param">
                <xsl:attribute name="name">SOURCE</xsl:attribute>
                <xsl:attribute name="value" xml:space="preserve">
<!-- start conversion to JCAMP -->
<!-- * means 'any' within a pattern match -->
##TITLE= <xsl:value-of select="@title"/>
##<xsl:value-of select="@convention"/>
##DATA type= <xsl:value-of select="*[@title='datatype']"/>
##XUNITS= <xsl:value-of select="*[@title='xunits']"/>
##YUNITS= <xsl:value-of select="*[@title='yunits']"/>
##FIRSTX= <xsl:value-of select="*[@title='firstx']"/>
##LASTX= <xsl:value-of select="*[@title='lastx']"/>
##FIRSTY= <xsl:value-of select="*[@title='firsty']"/>
##MINY= <xsl:value-of select="*[@title='miny']"/>
##MAXY= <xsl:value-of select="*[@title='miny']"/>
##NPOINTS= <xsl:value-of select="*[@title='npoints']"/>
<xsl:for-each select="*[@title='peak table' and @convention='(XY..XY)']">
##PEAK TABLE= (XY..XY)
<xsl:for-each select="coordinate2">
<xsl:value-of/>,
</xsl:for-each>
</xsl:for-each>
<!-- repeat for 'xypairs (XY..XY)' and 'peak table (XYM)' -->
##END=
<!-- end conversion' -->
                </xsl:attribute>
        </xsl:element>
        <!-- Reverses IR spectra -->
        <xsl:if test="*[@title='datatype' and .='INFRARED SPECTRUM']">
                <param name="WAY" value="REVERSE"/>
        </xsl:if>
</applet>
</xsl:template>
..
```

Scalable Vector Graphics (SVG)

Chemistry, in common with disciplines such as mathematics, conveys information through formal notation (often machine parsable) and more general graphical objects. In chemistry these include full and dotted lines, straight and curly arrows with various types of arrowheads, links, braces, containers, specialised glyphs and pictorial objects (e.g for surfaces, solids, etc.). Chemical schemes and diagrams are common and often consist of a mixture of graphics objects and formal notation. These enable great creativity of expression and several editing tools pay great attention to providing these. They are therefore difficult to capture accurately for machine processing using

a markup language. The only current method, line art or pixel maps, loses much of the machine readable semantics.

XML provides a very powerful graphics language, **Scalable Vector Graphics (SVG)**. Most Web-based images use bitmapped formats (.gif, .jpg, .png), which use a grid of coloured pixels to form the picture. In contrast, vector images use Cartesian descriptions of drawing objects (lines, polygons, fills, text) and overlays them onto a blank canvas. Vector formats are most efficient for simple line images and diagrams, where the files they produce will be significantly smaller than the equivalent bitmap. Since the drawing objects are described mathematically, vector images can also be zoomed and resized without pixellating. SVG uses a range of drawing 'primitives' (line, circle, path, etc. with fill, stroke, pattern etc.) as elements. Being XML it allows these to contain attributes and other element children, and SVG specifically anticipates that elements from other namespaces will be interspersed with its own information.

The natural use of SVG is for transmission of line art and other semantically rich graphics (2D only) over the Web. For example, most of the diagrams in this manuscript are created directly in SVG and can be viewed with widely available tools. SVG allows local and global rescaling, extraction of sub-components, and searches for text strings. Filters allowing files to be exported as SVG are available for Corel Draw [29] and Adobe Illustrator, and Adobe has released an SVG plugin [30] for Internet Explorer and Netscape Navigator. SVG can also be exported to various outputs, for example Acrobat or Postscript files

A common use of graphics is to display instrumental output such as spectra. SVG is a natural medium for this as the data are not corrupted and accurate values can be extracted from the spectrum if required. Moreover the spectrum can have XML-based links to other elements such as molecules (e.g. in a chromatogram) or atoms (as in molecular spectra).

We can therefore confidently use SVG as a semantically rich tool to co-exist with CML. A typical example is a Scheme with several molecules, linked with lines, and surrounded by containers; this could denote reactions, interrelationships, systematisations, etc. The basic framework consists of an SVG element which contain `<cml:molecule>` elements or, even better, links to a `cml:molecule>` elements. The graphic elements can contain metadata stating their function (e.g. a "curly arrow" denotes a 2-electron transfer from its tail to head). If the chemical community can converge on conventions for such metadata, it would allow searches of the documents for constructs such as that in Figure 17:

Figure 17: *Example of combined SVG and CML*

```
..
<svg:svg>
  <defs>
    <cml:molecule id="mol1">...</cml:molecule>
    <cml:molecule id="mol2">...</cml:molecule>
  </defs>
  <use href="#mol1" x="100" y="0"/>
  <svg:path d="M 100 0 l 100 0 100 100 0 100 z">
    <string type="metadata">Conformational change</string>
    <cml:float name="temperature" units="Celsius">80</cml:float>
  </svg:path>
  <use href="#mol2" x="0" y="100"/>
</svg:svg>
..
```

This portray the conformational interconversion of Molecule 1 to Molecule 2 at specific points in the Scheme and using a curved line to link them.

CML containing 2- or 3-D coordinates can be transformed to SVG with XSLT stylesheets. Since bonds contain references to atoms (atomRefs) it is possible to evaluate where the bonds should be drawn. Atom properties (e.g. calculated charge) can be simulated by spheres or other primitives, so XML/SVG-aware browsers already contain components for a simple molecular renderer.

The examples given in the supplementary materials (supplementary figure 9) were produced from CML using the XT parser. SVG promises to be an excellent alternative to applets for the display of static molecules, reactions and spectra. [31] [32]

Reactions

The `<reaction>` element was included in the CML DTD but not elaborated further. The techniques presented here are proofs-of-concept, and further work is required to develop robust and flexible reaction markup.

Reaction markup could potentially get extremely complicated, particularly if *atom mapping* or *functional groups* are included. For our initial experiments, we have concentrated on using CML to display simple reaction schemes within the IE browser. This can be considered as an extension of `<molecule>` markup and is approached in a similar way.

Linking via @id

A reaction is considered as a series of molecular species; potentially including reactants, products, reagents, intermediates, transition states etc. These species are then grouped into reaction steps with suitable information given at each step such as reaction conditions, yield, reaction name. If the structure of each species is expressed as a CML `<molecule>` and supplied with a unique @id value, then a reaction step need only contain references to the @id of each species (via `<@href="idname">`) and need not contain the entire molecule. This keeps the CML succinct, enhancing human readability. It also enhances the reusability of chemical components; a library of reactions using the same reactive species needs only contain one copy of its structure. A example of a simple CML reaction is given in Figure 18 with the molecular structures omitted. A full description of our reaction markup can be found in the supplementary information.

The attribute names @id and @href are those chosen as linkers for CML. Other languages might use different attributes and the parser needs a definition of which these are. IE uses three *data type* declarations in the schema for this purpose. An attribute declared as `dt:type="id"` contains an element's unique identity, `dt:type="idref"` contains a reference and `dt:type="idrefs"` may contain one or more references (space separated). The parser is required to ensure all identities contain unique strings and that all references refer to an existing identity.

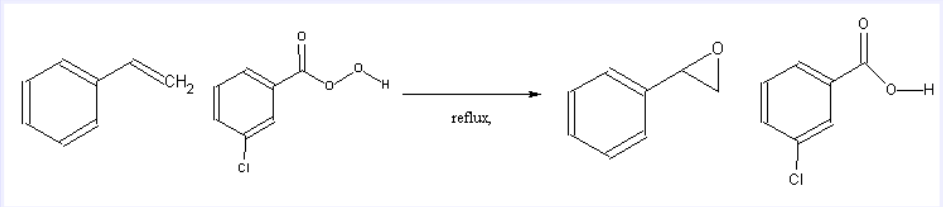
Figure 18: *Simplified 'stepwise' reaction illustrating a Diels Alder Cycloaddition expressed using CML*

```
..
<cml title="Simple Reaction" id="cml_simple">
        ..
        <reaction title="Diels-Alder cycloaddition" id="simple_rxn_1" convention="stepwise">
                <!-- overall information -->
                <string title="description">Simple example of a A + B -> C reaction.</string>
                <float title="yield" units="%">88</float>
                <string title="notes">taken from Vollhardt and Schore</string>
                <list title="reactionStep" id="simple_s_1">
                        <!-- reaction step information -->
                        <string title="description">cycloaddition</string>
                        <string title="notes">one step</string>
                        <!-- series of links to each reaction species -->
                        <link title="reactant" href="simple_mol_reactant1"/>
                        <link title="reactant" href="simple_mol_reactant2"/>
                        <!-- reagent could contain: a reagent name,
                                link to a structure or as here, a list of
                                conditions. multiple reagents can be included
                                if needed -->
                        <link title="reagent">
                                <integer title="index">1</integer>
                                <string title="solvent">Acetonitrile</string>
                                <string title="temperature" convention="degC">100</string>
                                <string title="duration" convention="hours">3</string>
                                <string title="notes">reflux</string>
                        </link>
                        <link title="reagent">
                                <integer title="index">2</integer>
                                <string title="notes">workup</string>
                        </link>
                        <link title="product" href="simple_mol_product"/>
                        <!-- could also include catalyst, intermediate,
                                transition state etc. -->
                </list>
        </reaction>
        ..
        <!-- the actual molecules can be anywhere, even in a different cml block -->
        <molecule title="2,3-Dimethyl-1,3-butadiene" id="simple_mol_reactant1">
        <!-- etc. -->
        </molecule>
        ..
        <molecule title="Propenal" id="simple_mol_reactant2">
        <!-- etc. -->
        </molecule>
        ..
        <molecule title="Diels-Alder adduct" id="simple_mol_product">
        <!-- etc. -->
        </molecule>
        ..
</cml>
..
```

Links of this sort are particularly useful for comparing or indexing two sets of data against each other. Further chemical examples might be atom mapping in a reaction or peak assignment for a spectrum. Significantly more advanced linking techniques called **XLink** and **XPointer** are being developed by the W3C [9] . These allow for one or two way linking between documents and complex *one to many* or *many to one* links. Using these techniques, a `<reaction>` in a XML document could reference data-sheets in a chemical archive. By clicking on a reaction species, the reader would then have access to full chemical data for that molecule.

Figure 19: *Stepwise epoxidation reaction displayed in a browser using the Marvin applet (CML directly converted from a rxn file)*

Stepwise Reaction: Epoxidation of Styrene	**id**: epoxidation_rxn_1
Description: Example of a simple A + B -> C + D reaction.	**Yield**: 75 %
Notes: taken from Warren	



Two stylesheets have been developed to display CML reactions. Both use the Marvin applet (in 2D mode) to display the reaction species and an XHTML table to format the applets into a

reaction schema, using gif images to represent reaction arrows. Additional information such as reaction conditions etc. is displayed above each reaction step or below the arrow, as appropriate. The first stylesheet (Figure 20) is able to display multiple reaction steps of the form A + B + .. -> X + Y + ..

Figure 20: *Stepwise reaction stylesheet fragment - reformatted for clarity*

```
 ..
<!-- use this template for a stepwise reaction -->
<xsl:template match="reaction[@convention='stepwise']">
        <!-- build HTML table -->
        <table><tr>
            <td colspan="2"><B>Stepwise Reaction:</B> <xsl:value-of select="@title"/></td>
            <td><B>id:</B> <xsl:value-of select="@id"/></td>
        </tr>
        <!-- etc. for 'description', 'yield' and 'notes' -->
        <!-- build a reaction scheme for each step -->
        <xsl:for-each select="list[@title='reactionStep']">
        <tr><td colspan="3">
                <table><tr>
                   <td>
                <!-- select each link to a reactant, then find and select the elements
                with matching @id (this is what id(@href) does) -->
                <xsl:for-each select="link[@title='reactant']">
                        <xsl:for-each select="id(@href)">
                                <!-- call a template that builds Marvin applet -->
                                <xsl:apply-templates select="list[./atom/float/@builtin='x3']"/>
                        </xsl:for-each>
                </xsl:for-each>
                </td><TD>
                <!-- the arrow is a normal gifs -->
                <img src="writeupdata/arrow.gif" /><br/>
                <!-- select each reagent in turn -->
                <xsl:for-each select="link[@title='reagent']">
                        <xsl:if test="*[@title='index']">
                                <!-- if an index number is given, include it -->
                                <xsl:value-of select="integer[@title='index']"/>)
                        </xsl:if>
                        <xsl:for-each select="id(@href)">
                                <!-- if a structure is given, include it -->
                                <xsl:apply-templates select="list[./atom/float/@builtin='x3']"/>,
                        </xsl:for-each>
                        <xsl:for-each select="string[@title='solvent']">
                                <!-- if a solvent is given, include it -->
                                <xsl:value-of select="text()"/>,
                        </xsl:for-each>
                        <!-- etc. for temperature, duration and notes -->
                </xsl:for-each>
                </td><td>
                <!-- select each link to a product -->
                <xsl:for-each select="link[@title='product']">
                        <xsl:for-each select="id(@href)">
                                <!-- call a template that builds the Marvin applet -->
                                <xsl:apply-templates select="list[./atom/float/@builtin='x3']"/>
                        </xsl:for-each>
                </xsl:for-each>
                </td>
                </tr></table>
        </td></tr>
        </xsl:for-each>
        </table>
</xsl:template>
 ..
```

The code to convert the molecular information to a Molfile and to build the invocation of the Marvin applet is almost identical to that described previously and has been omitted. The stylesheet is significantly more structured than previous examples, and contains larger amounts of XHTML. In particular the requirement for `<table>` tags restricts the flexibility of the stylesheet. More complex reactions would require dedicated stylesheets written specially for them.(supplementary figure 10) Alternative approaches would be to use the JME applet which can handle limited reactions, or SVG as noted above

Converting Legacy Formats to CML

Manually creating CML using simple text editors is laborious and so alternatives are needed. It is hoped that future chemistry software will include filters saving data directly to CML, but until this occurs the best solution is to convert existing legacy formats to CML. This has the advantage of normalising the existing range of diverse chemical files to a single format. The extensibility of XML (and hence CML) and the use of generic data holders means that any ASCII-text based legacy file can be converted to CML without information loss.

CML has been designed to make these conversions as easy as possible. In most cases (e.g. Molfile or JCAMP) this involves identifying blocks of data and wrapping them in the appropriate CML elements. This is equivalent to a complex text search and replace operation. Perl (a widely used text manipulation and report language) is an excellent tool for carrying out exactly this sort of manipulation. Perl scripts are plain text files and are normally run from the command line. With slight alterations, they can also be be run as CGI (*common gateway interface*) programs on a web server. This allows text written into an online `<form>` to be sent to the server, converted and returned to the user as an XHTML page. (supplementary figure 11)

A range of perl converters have been written and are available for use from the ChiMeraL site. Perl cannot easily access binary compressed data and since both Molfile and JCAMP files sometimes use compressed formats, they must be expanded before conversion. Available converters include;

- MDL .mol to CML
- MDL .mol to the JME input string
- .sd (archive format) to CML
- .xyz to CML
- JCAMP .dx and .cs to CML (JCAMP-CS files produce both a `<molecule>` and a `<spectrum>`)
- REACSS .rxn (reaction format based on .mol) to CML

Writing Complete XML Documents

Pure CML documents are best suited to chemical data-sheets or archived data. A more flexible approach is to combine chemistry with text and diagrams to from a report or scientific article. Namespacing allows different XML languages to be intimately mixed and an excellent text formatting language already exists in XHTML. A sophisticated XML scientific document might therefore consist of XHTML formatted text, chemistry using CML, symbolic theoretical expressions in MathML, and diagrams either as bitmaps (gif or jpg) or better, expressed as SVG. An XML document language is also needed to describe the document structure and to wrap these various components. A simple 'docuML' language has been developed to illustrate these concepts and used to write several documents, including this article.

Figure 21: *Simplified XML document - outlines the document structure*

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="document.xsl" ?>
<mrw:document title="Skeleton" id="xmldoc_skele"
        xmlns="http://www.w3.org/1999/xhtml"
        xmlns:mrw="x-schema:http://www.xml-cml.org/chimeral/mrw_schema_01.xml"
        xmlns:chimeral="x-schema:http://www.xml-cml.org/chimeral/spectrum_schema_ie_01.xml">

<!-- ********************** metadata ********************** -->
<mrw:metadata>
        <mrw:date builtin="date:creation">June 12, 2000</mrw:date>
        <mrw:author idref="insti_ic" id="author_mw" email="karne@innocent.com"
         href="http://www.ch.ic.ac.uk/chimeral">Michael Wright</mrw:author>
        <mrw:institution id="insti_ic" href="http://www.ch.ic.ac.uk">
        Department of Chemistry, Imperial College of Science, Technology and Medicine, UK
        </mrw:institution>
</mrw:metadata>

<!-- ********************** abstract ********************** -->
<mrw:abstract>Demonstrates the key elements of an XML document.</mrw:abstract>

<!-- ********************** keywords ********************** -->
<mrw:keywords>ChiMeraL, XML, CML, chemical markup language, chemistry</mrw:keywords>

<!-- ********************** chapter ********************** -->
<mrw:chapter id="chap_introduction" title="Introduction">
        <mrw:index/>
        <p>A document is made up of nested chapters and subsection.</p>

<!-- ********************** subsection ********************** -->
<mrw:subsection id="sub_xhtml" title="XHTML">
        <mrw:index/>
        <p>Elements with no prefix are assumed to be XHTML by the stylesheet and
passed directly to the output. This allows the author to use familiar <i>'HTML
like'</i> formatting without disrupting other XML blocks. Example code needs to
be escaped;
<div class="code"><docml:lt/>formula<docml:gt/>C4 H8 O<docml:lt/>/formula<docml:gt/></div>
</p>
</mrw:subsection>
</mrw:chapter>

<!-- ********************** bibliography ********************** -->
<mrw:bibliography title="References">
        <mrw:ref id="ref_CML1.0">For a formal description of CML version 1.0, see
        <mrw:index>1</mrw:index>
                <mrw:author href="http://www.xml-cml.org/">
                        P. Murray-Rust and H. S. Rzepa</mrw:author>
                <mrw:publication>J. Chem. Inf. Comp. Sci.</mrw:publication>
                <mrw:date>1999</mrw:date>
                <mrw:volume>39</mrw:volume>
                <mrw:pages>928</mrw:pages>
        </mrw:ref>
</mrw:bibliography>

</mrw:document>
```

An XSL stylesheet has been written combining the CML template fragments with a number of templates designed to format the various document components. The stylesheet is generic and able to display any document written using this docuML language. In order to allow the author to use familiar XHTML for text formatting, the stylesheet has been designed leave unrecognised elements unchanged. These pass through to the output, are recognised by the browser as

XHTML and formatted appropriately. By setting the document default namespace to XHTML, no element prefixes are required and the document can be written using standard HTML editing tools. A CSS stylesheet is also used, allowing changes to the appearance of the transformed document without having to edit the significantly more complex XSL.

Both CML and DocuML components have `@id` attributes, allowing sophisticated linking behaviour. The stylesheet can display components in any order and is not restricted by the physical order of the code in the XML source. CML molecules, spectra and blocks of sample code can be extremely long and would normally interrupt the chapter/subsection structure of the document. This makes it much harder for a human to read or edit. The preferred approach is to separate out CML data and sample code and move them to the end of the document. Links (`<docml:link builtin="molecule" display="jmol" idref="mol_thf_1"/>`) can then be used within text or figures, to reference these external blocks as they are required.

Figure 22: *Part of an XML document showing linking*

```
..
<!-- ********************** subsection ********************** -->
<mrw:subsection id="sub_idlinks" title="Linking">
<mrw:index/>
        <p>Links between @id and @idref are used to automatically build references
to the bibliography<mrw:link builtin="ref" idref="ref_CML1.0"/> or
<mrw:link builtin="figure" idref="fig_Jmol"/>. The reference will include the
correct index number, hyperlinks and mouse-over effects, even if the target is
moved or completely rewritten. This helps avoid the notorious HTML problem -
"broken links". Links are also used to build the document's index.</p>

<!-- figure title -->
<mrw:figure id="fig_Jmol" builtin="showLabel">Tetrahydrofuran using Jmol</mrw:figure>

<!-- this tells the stylesheet to render the CML object with @id="mol_thf_1"
using Jmol, changing @display changes the applet used -->
<mrw:link builtin="molecule" display="jmol" idref="mol_thf_1"/>
</mrw:subsection>
..
<!-- ********************** CML data block ********************** -->
<cml title="cml data block" id="cml_moldata" xmlns="x-schema:cml_schema_ie_02.xml">
        <molecule title="Furan, tetrahydro-" id="mol_thf_1">
                ..
        </molecule>
        <chimeral:spectrum title="Caffeine" id="spect_caffeine_ms_1" convention="JCAMP-DX= 4.24">
                ..
        </chimeral:spectrum>
</cml>
..
```

Links can also be used to automatically build and maintain literature or figure references. For example, the stylesheet can search for the start of each chapter, sub-section and figure, then use this information to automatically create an index of titles complete with hyperlinks. Document components can be reordered or even completely rewritten, and these links will be automatically updated. Figure 22 shows references to a figure and the bibliography (lines 6 and 7)and a link to a CML molecule (line 17). Although the last instructs the stylesheet to use the Jmol applet to render the molecule, a different applet can be chosen simply by changing the value of `@display`.

Printing XML - Formatting Object and FOP

Formatting Objects (FO) is a sub-language of XSL, intended to describe the layout of document components on a printed page. It comprises formatting elements rather similar in concept to XHTML but describing exact positions and font sizes in contrast to relative positioning in XHTML. FOP is a Java application being developed by Apache. [33] When it is combined with a Java XML/XSL parser (e.g. Xalan) it acts as a formatting object parser and can convert an FO file to an Adobe Acrobat .pdf file. Acrobat is a very widely used for exchanging 'ready for print' electronic documents and being a read-only format has important legal implications.

Using multiple stylesheets, an XML document can be transformed for browser display or converted to a FO file for printing. Since the stylesheets can optimise the document for each purpose, the results are much superior to printing directly from the browser. A printable version of this article was produced in this way. It is expected that further converters will become available as this technology matures.

Reusable Journals

Journals and books are currently created for humans to read. Some contain machine readable supplemental material but the formats are not usually standardised. Articles in XML/CML have the potential of being entirely machine processible, which means it would be easy to extract all the `cml:molecule>` elements in an article or, indeed, in an entire book, journal, journal collection or publisher's output.

The reader can then ask sophisticated questions of this data, such as e.g.

- what molecules contain a given functional group?
- where do molecules and their spectra co-occur?
- identify all molecules which are described as taking part in reactions

Note that XML/CML allows flexible queries so that the author of the article(s) need not necessarily anticipate the use that the publication will be put to. For example, given the relative cheapness of evaluating molecule wavefunctions, the orbitals of all molecules specified by criteria such as size could be computed and from the resulting XML output, those with given energies or properties extracted. Similarly, compounds not in the user's database could be listed for potential synthesis; in time the synthesis could be automatic!

This article is the first with a chemical theme to be expressed completely in XML, which we use here as one model for the future development of chemistry books and journals and which could enable data checking at authorship time, extensive crosslinking of information and normalisation of existing information.

Conclusion

Applications for the CML/XHTML system described here might include *inter alia* electronic journals, safety data-sheets as part of drug ratification procedures, patent processing where microscopic capture of information is essential (and failure can lead to the paten being challenged) and e-commerce. One can envisage future parsers able to comprehend both CML and XHTML without the requirement for sophisticated stylesheets. By including CML support in computational chemistry and modelling software, a 'rolling stone' approach can be taken whereby each process the molecule passes through (structure optimisation, property calculation etc.) involves full information retention. This is in contrast to present solutions where information can be lost at every stage. Enhanced applet-CML communication and the possibility of producing CML from a editor [34] also have great potential. Ultimately, server-side processing of XML/XSL stylesheets will allow powerful stylesheet transformations to be accessible using any web browser.

Acknowledgement

Michael Wright would like to thank the Chemical Structure Association (CSA) (http://www.chem-structure.org/title.html) for the award of a bursary and an Exemplarchem prize.

References

1 H. S. Rzepa, P. Murray-Rust and B. J. Whitaker, *Chem. Soc. Revs.,* 1997, 1
2 H. S. Rzepa, B. J. Whitaker and M. J. Winter, *J. Chem. Soc., Chem. Commun.,* 1994, 1907
3 O. Casher, G. Chandramohan, M. Hargreaves, C. Leach, P. Murray-Rust, R. Sayle, H. S. Rzepa and B. J. Whitaker, *J. Chem. Soc., Perkin Trans 2,* 1995, 7
4 P. Murray-Rust, C. Leach and H. S. Rzepa, *Abs. Papers. Am . Chem. Soc.,* 1995, **>210,** 40-COMP
5 For a formal description of CML version 1.0, see P. Murray-Rust and H. S. Rzepa, *J. Chem. Inf. Comp. Sci.,* 1999, **>39,** 928
6 Demonstrations, examples and further information can be found in the ChiMeraL website, *(http://www.xml-cml.org/chimeral/)*
7 MDL information systems, *(http://www.mdli.com)*
8 Brookhaven Protein Data Bank, *(http://www.rcsb.org/pdb/)*
9 World Wide Web Consortium (W3C). Specifications for XML, XHTML, XSL, SVG, Schemas and other standards can be found at this site., *(http://www.w3.org)*
10 P. Murray-Rust, H. S. Rzepa and M. Wright, *Chem. Commun.,* 2000, 1471
11 Chemical Markup Language homesite, *(http://www.xml-cml.org/)*
12 Adobe Acrobat, *(http://www.adobe.com/products/acrobat/main.html)*
13 Microsoft XML developer's site, *(http://msdn.microsoft.com/xml/)*
14 IBM alphaworks - XML/Java development site, *(http://www.alphaworks.ibm.com/)*
15 Sun - Java Technology Homepage, *(http://java.sun.com/)*
16 P. Ertl and O. Jacob, *Theochem,* 1997, **>419,** 113-120
17 J. Brecher, *Chimia,* 1998, **>52,** 658
18 D. Weininger, *JCICS,* 1988, **>28,** 31
19 D. Gezelter and D. Gezelter, Jmol - Open source Java 3D viewer, *(http://www.openscience.org/jmol/)*
20 Marvin display and editing applet - Chemaxon, *(http://www.chemaxon.com/marvin/)*
21 ACD Labs - Structure Drawing Applet, *(http://www.acdlabs.com/products/java/sda/)*
22 W. G. Mallard and P. J. Linstron, *Abstr. Pap. Am. Chem. Soc.,* 1998, **>216,** U526-U526 *(http://webbook.nist.gov/)*
23 R. S. McDonald and P. A. Wilks Jr., *App. Spec.,* 1988, **>42,** 151-162
24 SPC - Spectroscopy Software Solutions, *(http://www.galactic.com/)*
25 A. N. Davies and P. Lampen, *App. Spec.,* 1993, **>47,** 1093-1099
26 P. Lampen, H. Hillig, A. N. Davis and M. Linscheid, *App. Spec.,* 1994, **>48,** 1545-1552 *()*
27 J. Gasteiger, B. M. P. Hendriks, P. Hoever, C. Jochum and H. Somberg, *App. Spec.,* 1991, **>45,** 4-11
28 G. Cottenceau and H. S. Rzepa, JSpec: A Java Spectral Visualiser for JCAMP-DX and SPC Format Files, *(http://www.ch.ic.ac.uk/java/applets/jspec/spectra/)*
29 Corel Draw 9 SVG plugin, *(http://venus.corel.com/nasapps/DrawSVGDownload)*

30 Adobe SVG browser plugin, *(http://www.adobe.com/svg/viewer/install/)*
31 Mayura Draw - editor able to export SVG, *(http://www.mayura.com/)*
32 Csiro SVG Toolkit, *(http://sis.cmis.csiro.au/svg/index.html)*
33 Apache.com - FOP project (XML conversion to Adobe Acrobat), *(http://xml.apache.org/fop/)*
34 JChemPaint, an Open source Java 2D editor S. Krause, E. Willihagen and C. Steinbeck, *Molecules,* 2000, **>5,** 93-98 *(http://www.ice.mpg.de/~stein/projects/JChemPaint/)*
35 A. Homer, *XML IE5 - Programmer's Reference, Wrox Press,* 1999,
36 H. Dietel, P. Deitel, T. Lin and T. Neito, *XML How to Program, Prentice Hall,*
37 H. Dietel, P. Deitel, T. Lin and T. Neito, *The Complete XML Training Course, Student Edition, Prentice Hall,*