Chemical Markup, XML, and the World Wide Web. 4. CML Schema

Peter Murray-Rust*,† and Henry S. Rzepa*,‡

Unilever Centre for Molecular Informatics, Department of Chemistry, University of Cambridge, Lensfield Road, Cambridge, CB2 1EW, England, and Department of Chemistry, Imperial College London, SW7 2AZ, England

Received December 12, 2002

A revision to Chemical Markup Language (CML) is presented as a XML Schema compliant form, modularized into nonchemical and chemical components. STMML contains generic concepts for numeric data and scientific units, while CMLCore retains most of the chemical functionality of the original CML 1.0 and extends it by adding handlers for chemical substances, extended bonding models and names. We propose extension via new namespaced components for chemical queries, reactions, spectra, and computational chemistry. The conformance with XML schemas allows much greater control over datatyping, document validation, and structure.

1. INTRODUCTION

In 1995 we first proposed a markup language called "Chemical Markup Language" or CML. This language was based on the use of SGML (Standard Generalized Markup Language) using the DTD (Document Type Definition) as the mechanism for constraining structure and vocabulary in SGML documents.[1] We recognized at the time that this approach only provided part of the toolset we needed for chemical information objects but found that SGML provided sufficient mechanisms for a prototype language. The original design was based on three components: (*a*) chemical information or "CML", (*b*) general scientific concepts (experimental markup language, referred to as "TecML"), and (*c*) text and graphics. The third component was adequately supported by HTML, but we needed to create the others. The first consisted of chemical components, such as molecules and atoms, and the second comprised general datatypes such as float, array, and strings. The complete toolkit was made by merging the three DTDs (CML, TecML, and HTML).

At that stage SGML did not support simple mechanisms for managing what are now called namespaces,[2] and the mechanisms for DTD-based validation were complex and fragile. Although the design was modular, there were no tools to support this, and for ease of implementation and robustness the first two components were merged into a single language, now referred to as CML1.0 and published in 1999.[3] This design not only recognized the importance of namespaces and modular design but also adopted the practice of markup languages at that time and defined the language vocabulary.

Since 1999 the XML community has made significant progress in design and the software tools to support it.[2] Namespaces are fully accepted, and all modern XML-based protocols are fully namespace-aware. This has been essential for modular design, though it is still at an early stage. We showed its potential for molecular sciences in a number of publications where the documents used two or more namespaces.[4−8] It is possible to process these documents with any XML tool and to apply namespace-specific operations to components in the documents.

The most important developments for general use since the original CML publication[3] have been the introduction of XML Stylesheets (XSL) and XML Schemas (XMLSchema).[2] XSL is a general XML-processing language which is often the first-choice tool and whose use in chemistry we shall describe in a follow-up article in this series. It allows transformation of one XML document into another or into non-XML data and has a rule-based approach to processing. XML Schema, which we describe here, provides for validation of information in a much more powerful manner than the DTD. Both XSL and XMLSchema are fully namespace-aware.

The increasing interest in using CML[8] has convinced us of the need to recast it in a more tightly modular manner and to identify the components more cleanly. In addition we recognize that users may wish to add their own concepts, to mix CML with other namespaces and to make use of many other operations which are common in XML. The CML 1.0 vocabulary had proved robust and useful[3] and could support a wide range of concepts, but there were additional requirements that needed to be included (e.g. chemical substances). Some of the original elements were not described in great detail, such as formula and electron. Some macromolecular concepts (sequence and feature) had no particular benefit over the growing use of XML in the macromolecular community, and we now do not intend to support these in future releases.

The DTD mechanism gives no support to datatypes; there is no way of requiring that a string of characters represents a number for example. This was a serious drawback in the commercial world, and the drive for the next generation of validation tools was driven by this need. There have been several approaches to this, but the main one has recently culminated in the publication of "XML Schemas" from the

* Corresponding author e-mail: h.rzepa@ic.ac.uk.
† University of Cambridge.
‡ Imperial College London.

10.1021/ci0256541 CCC: $25.00
Published on Web 04/11/2003

Chart 1

```
<p:entry id="p123" sku="P-123-4">
 <p:price amount="1000" unit="ml">
 <p:currency="USD">100</p:currency>
 <p:currency="GBP">66</p:currency>
 </p:price>
 <c:molecule id="p1" title="phosphine" xmlns:c="http://www.xml-cml.org/schema/cml2/core>
 <c:atomArray>
 <c:atom elementType="P" hydrogenCount="3"/>
 </c:atomArray>
 </c:molecule>
</p:entry>
```

W3C (World Wide Web Consortium).[2,9] Although there are other valuable approaches (RELAX-NG and Schematron are two)[10] we base our approach on XMLSchemas because we believe that it will be the primary method and that software tools for this approach will continue to be developed. However the concepts in this article can be readily translated into the other approaches.

The use of DTDs created an artificially complex syntax for CML 1.0. Because of the lack of datatyping it was necessary to create objects such as ⟨integer⟩ and to add semantics with a special attribute builtin. This increased the verbosity of CML and made it less attractive to humans, who would (not unreasonably) fail to see why the language was so constructed. It also made software creation more difficult. The introduction of schemas has given us an excellent opportunity to simplify the syntax while greatly increasing the power of this approach. Fortunately, XSL stylesheets can easily translate between different versions of CML syntax, so that conversion from e.g. CML 1.0 to the new family of CML components (collectively known as CML 2.0) and indeed to any future versions is a straightforward process.

The next sections show in detail why we believe the correct approach is to modularize CML, how the basic concepts in CML have evolved, and where we present the core of the new and formally frozen CML as an XML schema.

2. THE CASE FOR MODULARIZATION

If information can be packaged into modules the benefits are considerable:

- The need for understanding the context often disappears. This is a major gain in reusing information and writing the software. If "cml:molecule" has the same semantics regardless of whether it is contained within a database, a primary publication, a regulatory submission or associated with a spectrum, then a single set of software modules fits all the circumstances.
- Complex documents can be created with components produced in parallel. If the molecular information does not have semantic collisions with (say) clinical trial data, the chemistry in a submission can be compiled independently.
- Each module (defined by a namespace) can be viewed as consisting of a set of reusable components with associated software. Thus SVG (Scalable Vector graphics[11]) content can be associated with "an SVG viewer", it is not necessary to know the details of the implementation. Similar we can create "CML viewers" which perform independently of where they are invoked.
- Each domain can concentrate on just those components which are key to its ontology. In implementing CML 1.0, we found the need to implement text and image viewers (because XHTML and SVG were not then available). Now we can concentrate on just the chemical aspects.
- Software systems will often consist of a generic XML framework with specialist software modules (currently "plug-ins" or "ActiveX components", but the technology is always changing).
- Complex documents can be analyzed as a set of noninteracting components, making searches much easier to implement. If all Web-resident molecules are given a namespace belonging to CML, then it is conceptually easy to retrieve all such components. There will always be a need to create indexes for most searches to speed up performance, but the indexing tools need only look for namespaced molecules.
- Modularization often encourages the implementer to think more clearly about the semantics. A first pass at creating a markup language often benefits from being modularized, even if the modules are later found to be unnecessary, and combined for easier understanding.
- Some complex concepts are naturally built up of separate components ("aggregation"). Thus an entry in a chemical catalog could include price and molecular information. An entry might appear as in Chart 1.

The CML and other components do not interact other than that the molecular information is an integral part of the entry

Modularization is not always possible. This happens when concepts are not "atomic"—e.g. where A relies on information in B. This is an example of context-dependency and can be very difficult to model. For example, the concept of "molecule" varies widely and many chemists may want to know the physical state of the substance, the atomic constitution or experimental properties before they use the term. Our usage of cml:molecule is purely operational and provides simply an element to label with information or a container for information on formula or connectivity. Thus both these examples

```
<cml:molecule id="a1" title="black oil"/>

<cml:molecule id="a1" title="water">
<formula conciseForm="H 2 O 1"/>
</cml:molecule>
```

are valid CML, but neither implies anything about physical state, oligomerization, etc. This information can be added explicitly if it is important. We have traded a generic concept which is (probably) impossible to implement for a precise specific concept that can be reliably and consistently implemented. Authors and users of markup languages must be very aware of precisely what semantics are associated with any element or attribute. We deliberately discourage any implicit semantics; thus the English title "water" should not be taken to imply any physical state.

In many cases there are relationships between components, and no general way of implementing them automatically. Thus angle represents an angle between three different atoms. The implementation can be precise since the mandatory atomrefs 3 attribute is constrained (by XSLT) to reference three distinct existing atoms, *referenceable from the current context*. There is no requirement that these atoms are in the same molecule, because it could be useful to define intermolecular angles. Most users would make the *implicit* assumption that this is only a useful concept when the target atoms share coordinate systems than can be correlated. We have not yet formalized molecular coordinate systems (an important concept in computational chemistry where molecular superimposition, dynamics trajectories, etc. can give rise to several coordinate systems). When this is done, we might reasonably tighten the semantics to *reference three existing atoms sharing a common coordinate system or where transformation matrices between the coordinate systems exist.* This condition can be formally represented using XSLT.

3. REVISION OF CML CONCEPTS

3.1. Standardization of the Vocabulary and Harmonization with Other Initiatives. We recognize that there will be several approaches to markup languages in any domain, and it is important that these are interoperable. Syntax is relatively unimportant since syntaxes can be interconverted using XSL stylesheets. More important is the need to ensure that semantics and ontologies are clearly defined; in domains such as chemistry this should mean that consistent software should be created for commonly agreed semantics. Thus "elementType" might refer to elements in the periodic table, while "atomType" could be a set of user-defined views of an atom's properties or use. Failure to define semantics leads to confusion and poor software; robust semantics is a major benefit of the CML Schema approach.

There are several methods for defining and enforcing semantic consistency. Prose descriptions, as given in our 1999 article[3] are useful, but usually contain so much ambiguity that software is difficult to write. With procedural software such as C++ or Java programs, the source code is often not available and different software often gives conflicting results. It is difficult to extract the semantic concepts from the source code and in many cases this is not available. A formal language is therefore fundamental and XML Schema is a much more powerful formal language than use of the SGML DTD. Conformance tests—a set of test data with defined processing behavior—are also valuable but usually only enforce, or test, a subset of the constraints. A rule-based system is therefore also valuable, and we believe the XML stylesheet language (XSLT) will be useful for formalizing the rules and concepts.[2]

It is critical that all approaches to describing and constraining semantics are explicit, highly documented, and Open (OpenSource and OpenData). XML Schemas recognize this and give several mechanisms for supporting this. Since XML Schemas use XML syntax, they are easy to process as documents and can be reused in many ways. This avoids versioning errors, transcription problems, and ensures consistency. For example, the Appendix in this article has been directly created by transforming the current CML Schema specification so as to ensure that no incompatibilities are introduced by "human error". OpenData approaches encourage the sharing of semantic developments. Thus the Life Sciences Research Group of the Object Management Group[12] are developing a chemical ontology for implementation in the CORBA/IDL framework, which could make use of the semantics and ontology we have developed for CMLCore while working with a different syntax. This would avoid most of the ontological impedance that arises with uncoordinated efforts. In a similar fashion we expect CML could be used by bioOntologies[13] where there is a need for "small molecule" chemistry.

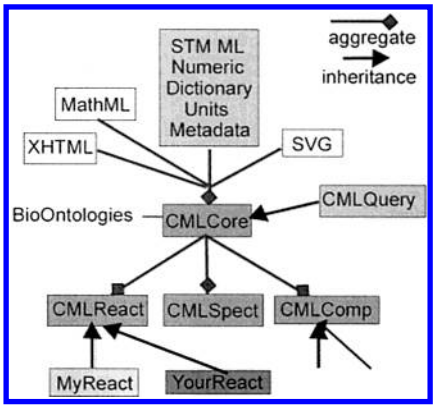

Figure 1. Schematic showing the CML2 Family of Components and their relationship to other XML Markup Languages.

3.2. Separation of Chemical Concepts into STMML (Nonchemical) and CML (Chemical). Within the case for modularization made above, we have therefore chosen to revert to the original design. Two important decisions have been taken. First, the nonchemical aspects of CML have been removed into a new markup language, now called STMML (Scientific, Technical and Medical Markup Language).[14] Second, CML itself has been modularized into a core and extended components (Figure 1).

STMML contains generic concepts for numeric data, structures such as arrays and matrices, scientific units, error information, and metadata/dictionaries, and CMLCore (and all other components) can use elements defined in STMML. These include ⟨*item*⟩ for a simple scalar object, ⟨*array*⟩ for a list of similar objects, etc. This is an example of aggregation—CMLCore is a superset of STMML. However all STM elements retain the STMML namespace (and our examples use the prefix stm:). Thus an ⟨cml:atom⟩ could contain a ⟨stm:matrix⟩, perhaps to represent its polarizability. STMML has been fully described elsewhere,[14] and we simply list the elements here with a brief description (Table 1).

3.3. Modularization of CML into a Chemical Core Subset (CMLCore) and an Extended Chemical Functionality. In considering extensions to the CML functionality, we anticipated that each area would be sufficiently rich semantically that it would require specific software to be written as well as specialized vocabularies. Figure 1 shows how a CMLCore could be used as the basis for such extension via query functionality (CMLQuery), spectra (CMLSpect), reactions (CMLReact), and computational experiments (CMLComp), together with associated areas such as biomolecular science ("bioontologies").[13]

3.4. Updating the CMLCore Syntax. This is an overview of CMLCore Schema, mainly based on its elements. It is

Table 1. STMML Elements and Their Descriptions

Objects Representing Data Types	
item: a simple datatype (especially derived from W3C types)	object: complex datatype with arbitrary content
Elements to Structure the Data	
array: a (one-dimensional) array of homogeneous objects	matrix: a (two-dimensional) matrix of homogeneous objects, with constraints on structure
list: a (one-dimensional) array of arbitrary heterogeneous objects	table: a (two-dimensional) table, where columns are homogeneous:
Processes	
actionList: a list of ⟨action⟩s	action: a step in a process
Dictionary	
dictionary: a dictionary, made up of entries	relatedEntry: a link to a related Entry
entry: a dictionary entry	alternative: an alternative name
annotation: annotation for an entry, based on W3C Schemas	enumeration: enumeration of possible values for an entry
documentation: human-readable annotation	definition: definition of the entry
appinfo: machine processable annotation	description: description associated with the entry
Relationships and Links	
xlink: a map between two elements	simpleLink: another simpleLink, using XLink syntax:
link: a simple link	
Metadata	
metadataList: a list of <metadata> elements	metadataType: the type of the metadata: (DublinCore-based− extensible to Chemistry)
metadata: an item of metadata	
Scientific Units	
unitList: A list of ⟨*unit*⟩ s	
unit: a scientific unit, both SI and derived	dimensionType: dimension
	dimension: dimension
UnitType: type of the unit	

Chart 2

```
<molecule id="a123" title="methanol">
 <atomArray>
 <atom id="o1">
 <string builtin="elementType">O</string>
 <integer builtin="hydrogenCount">1</integer>
 </atom>
 <atom id="c2">
 <string builtin="elementType">C</string>
 <integer builtin="hydrogenCount">3</integer>
 </atom>
 </atomArray>
 <bondArray>
 <bond>
 <string builtin="atomRef">o1</string>
 <string builtin="atomRef">c2</string>
 <string builtin="order">1</string>
 </bond>
 </bondArray>
</molecule>
```

Chart 3

```
<molecule id="a123" title="methanol">
 <atomArray>
 <atom id="o1" elementType="O" hydrogenCount="1"/>
 <atom id="c2" elementType="C" hydrogenCount="3"/>
 </atomArray>
 <bondArray>
 <bond atomRefs="o1 c2" order="1"/>
 </bondArray>
</molecule>
```

not the complete schema, which is included in the Supporting Information for this article. Familiarity with CML DTD 1.0 will be useful as much of the vocabulary and semantics are unchanged. Reuse is particularly easy for attributes since they are simpleTypes and only depend on other XML Schema primitives or other simpleTypes. Some elements can also be easily reused e.g. crystal, which defines its own attributes and only requires a few simpleTypes. Some elements have considerably enhanced content models, including molecule, electron, and formula.

Datatypes were managed in CML V1.0 through the specific elements ⟨float⟩, ⟨integer⟩, and ⟨string⟩. Not only is this a very limited range, requiring major revisions if new types are to be added, but it required an ugly and artificial attribute builtin to add the semantics. Moreover DTDs could not be used to validate the value of the attribute against the dataType. XML Schemas now allow us to add datatypes to XML attributes, so that many atom and bond properties can be expressed more simply. Thus all attributes and most textual content are now strongly datatyped. For some the builtin xsd:* types are adequate but others have been defined with maximum and minimum values (e.g. angle and torsion are restricted to appropriate values). Special datatypes have been constructed for referring to groups of atoms with precise cardinality. These may be validated with XSLT.

It is important to stress that we have not had to change the CMLCore concepts significantly. CMLCore is easier to implement than CML1.0, and it is trivial to convert between them using XSLT stylesheets. Elements which used to be children of atom and bond are now attributes in CML Schema. For example the CML1.0 syntax shown in Chart 2 is more verbose and harder to implement than the CMLCore equivalent shown in Chart 3.

All the CML attribute values are now datatyped and can be validated by generic XML Schema tools. Thus

- id must contain an alphanumeric string with only limited punctuation and no whitespace
- elementType must correspond to an enumerated list of element symbols (the Periodic Table)
- hydrogenCount must be a non-negative integer
- atomRefs must have at least two references to different atom id values
- order must correspond to an enumerated list

XML Schema tools and XSLT can enforce all these constraints without invoking any specifically chemical software. The rules are written in formal languages (XML Schema and XSLT) and are therefore completely open to inspection and should be universally and consistently applicable regardless of the actual software used. This approach

Table 2. CMLCore Elements

Structuring Elements	
cml	
Nonmolecular Concepts	
amount	substance
	substanceList
property	propertyList
Fundamental Chemical Objects	
atom: atom, always within ⟨molecule⟩ or ⟨formula⟩	electron
bond: bond, always within ⟨molecule⟩ or ⟨formula⟩	molecule: a general container for atoms, bonds, and formula
crystal: crystallographic cell parameters	
Atom-Related Objects	
atomArray: a container for a list of atoms	atomParity: stereochemical descriptor for an atom:
Bond-Related Objects	
bondArray: a container for a list of bonds	BondStereo: stereochemistry associated with a bond
Molecule-Related Objects	
formula: stoichiometry, including hierarchy and aggregation	angle: angle between 3 atoms
identifier: unique identifier, from either algorithm or lookup table	length: length between two atoms
name: a formal or trivial name, as specified by a stated authority	torsion: torsion between 4 atoms
symmetry: formal symmetry, specified by group theory or other	

allows the content models of many elements to become much more precise. Content models using the older DTD approach usually had to be either very rigid or very flexible. The former precluded innovation, while the latter did not permit useful validation. XML Schemas have allowed us to create much more powerful content models, especially for ⟨molecule⟩ and ⟨formula⟩.

3.5. Additional Core Chemical Concepts. CML has proved stable in supporting the communality of explicit and implicit concepts in current molecular and chemical data files. The current concepts in CML are basically those in CML V1.0 but with the addition of the following elements to the core.

• **3.5.1. Substances.** Two new elements amount and substance (with a container substanceList) have been added. These are essential for managing accounts of macroscopic chemical processes such as details of syntheses and for describing solutions and mixtures, i.e., it is designed to cover information in "common chemical discourse". Together with elements already described in STMML, there is now a sufficient vocabulary to describe the synthesis of compounds, analytical procedures, and the determination of properties.

In general relationships will require specific software to be implemented, and it is useful to make them as explicit as possible. An interesting chemical example is

"100ml of 0.1M NaOH"

This is surprisingly complex to implement in such a way that it is automatically machine-processable. It involves a concentration (M), which is a relationship between quantities (1 mole of NaOH related to 1 dm^3). We found we needed to develop a concept substance and a container substanceList to represent a number of similar relationships (solutions, mixtures, etc.). It also requires the lookup of units and (assuming we wish to know the mass of substance) the calculation of the molar mass. A current representation in CML is shown in Chart 4.

Chart 4

```
<cml:substanceList id="s1" title="100 ml of 0.1M NaOH">
<cml:amount units="unit:ml">100</cml:amount>
<cml:substance id="s1">
<cml:amount units="unit:l">1</cml:amount>
<cml:molecule id="h2o" ref="mols:water"/>
</cml:substance>
<cml:substance id="s2">
<cml:amount units="unit:mole">0.1</cml:amount>
<cml:molecule id="naoh" formula="Na 1 O 1 H 1"/>
</cml:substance>
</cml:substanceList>
```

• **3.5.2. Extended Bonding Models.** The bonding model in CML V1.0 implies well-defined valence bonds. We here include extension to "delocalized" systems, which are manageable in several ways:

◦ Multicenter bonds. The atomRefs attribute allows several atoms to be referenced in a bond; thus B−H−B bonds might be described as one three-center bond

◦ π-bonds. A two-center bond can reference other bonds rather than just atoms. We give an example below in the schema for π-donors bonds

◦ The explicit use of electron. A set of electrons can be linked to a set of atoms without formally describing a bonding model

• **3.5.3. Names and Nomenclature.** Originally names (especially for molecules) were given as a (single) title attribute or as string children of molecule. Proper management of names is critical for identification of chemical substances, and the name element has been introduced as an optional/repeatable child of molecule. In general names cannot be algorithmically generated from connection tables and represent any string identifying a molecule. The authority or convention should be indicated by a convention attribute:

```
<molecule id="a123">
<name convention="IUPAC">benzene</name>
<name convention="CAS">71-43-2</name>
</molecule>
```

The recent IUPAC project on a unique identifier generated algorithmically from the connection table (IChI)[16] generates an XML representation which we shall expect to formally include in CML when the syntax is finalized. Options could include a IUPAC-specific namespace, a child of name or integration into future CML.

3.6. The Full CMLCore Element List. A complete list of CMLCore elements is shown in Table 2; the machine-processable schema is available via the Supporting Information.

3.7. Extended Chemical Concepts. In addition to these additions to the core, our modular design anticipates several major additional concepts, which we propose here in the form of extended CML modularized languages. We emphasize that

CMLCore (and all other components) will be able to use elements defined in other markup languages and *vice versa*. The most common are the generic W3C languages (XHTML, SVG, MathML, RDF),[2] but many other tools such as DOCBOOK can also be aggregated. Whether CML elements contain (say) SVG elements, or *vice versa*, is mainly a matter of style; XSLT can transform between different approaches (If a sibling relationship is used, it helps to have a common container.). Similarly, a collection of enzyme mechanisms might represent the substrates, products, and reaction mechanism with CMLReact.

• **CMLQuery** has many of the same elements and attributes as CMLCore, but they have different semantics. CMLCore usually represents a definitive substance, while CMLQuery involves a more flexible grammar. Thus ⟨hydrogenCount⟩ can specify a range of values which would fail validation in CMLCore. This is an example of inheritance and overriding; CMLQuery inherits the basic concepts in CMLCore and where appropriate changes (overrides) the semantics and syntax.

• **Fuzzy Molecular Constitution.** CML is designed to describe a single molecular species which is exactly described. In cases where the constitution is incompletely known, or where a specification describes a set of similar molecules (e.g. combinatorial chemistry) the tools in CMLQuery will be used.

• **Reactions using CMLReact.** CML V1.0 had an element for reaction, but the semantics and usage were relatively basic.[5] With the need to represent atom mapping, reaction conditions, and more complex topologies we feel that a richer description than the current reaction will be required, and we have therefore removed reaction from the Core Schema, although it continues to be a usable component in CML V1.0. We now propose that CMLReact will use CMLCore to describe the molecules in a reaction and adds elements to describe the relationships between them.

• **Computational Chemistry and Related Concepts.** We are actively designing an extension to manage concepts derived from computations, currently designated "CMLComp" in this article. This will manage input and output to molecular mechanics, dynamics, and quantum mechanics. Besides the specific vocabularies of these disciplines, there are general structures that cover sets of atomic and bond properties (e.g. multiple conformations, molecular vibrations, atomic orbital coefficients, etc.) CMLComp (Computational Chemistry Markup Language)[8] uses CMLCore to describe the molecules associated with a computation. Many properties of molecules, atoms and bonds can be simply aggregated as children (⟨*item*⟩ or ⟨matrix⟩, etc.) but multiple or dynamic values of coordinates, charges, etc. require additional semantics to describe the relationships.

• **CMLSpect.** This will use CMLCore to describe the molecules associated with a spectrum. In its simplest form these are two noninteracting sets of components. However it is being developed so that relations between the two (e.g. peak assignment) can be explicitly represented. We are also aware that other groups are working in these areas of molecular science, and CML is designed to interoperate with these *even if we do not know their precise details*. For example, an initiative in the spectroscopy community is developing SpectroML.[15] CMLSpect is intended as a basic approach to supporting spectra, sufficient for simple applications (e.g. x−y data and linking peaks to structural features) and is not intended to be a comprehensive description of the experimental procedures. However it can be used as a stub from which other more specific or more comprehensive languages can be derived. The technical aspects of creating other languages (aggregation and inheritance) are described later.

Chart 5

```
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:h="http://www.w3.org/1999/xhtml"
 xmlns="http://www.xml-cml.org/schema/cml2/core"
 version="1.0">

 <xsl:template match="molecule">
 <h:h2><molecule ID:<xsl:value-of select="@id"/></h:h2>
 </xsl:template>

</xsl:stylesheet>
```

4. NAMESPACES

Namespaces are not formally part of XML V1.0 but have become universal in all XML-based languages and are supported by all XML-tools and APIs such as DOM and SAX. The namespace concept is simple: every element and every attribute in a document can have a unique namespace associated with it. Uniqueness is created through a namespace URI, normally containing the domain name of the "owner" of the vocabulary, though other systems (e.g. ISBN) could be used. Despite its URI-based syntax the namespace URI does *not* have to exist physically and there is no requirement to connect to any network. Note that the namespace prefix is arbitrary; although "xsl" is used above, any unique prefix would suffice (Chart 5).

This example contains three namespaces and is actually part of an XSL stylesheet which shows how information can be transformed from one namespace (CMLCore Schema) to another (XHTML):

• http://www.w3.org/1999/XSL/Transform. This URI defines the XSLT (XSL Transformation) namespace and (in this example) associates all the elements prefixed by xsl: with it. Since there are likely to be different versions of XSLT (requirements for XSLT2.0 have been collected) it will signal to software systems what functionality is required.

• http://www.w3.org/1999/xhtml. This denotes the namespace for XHTML, and elements prefixed (in this case) with h: will belong to that namespace.

• http://www.xml-cml.org/schema/cml2/core. A single *default namespace* without prefixes is allowed, in this case for CMLCore Schema. Note that CML DTD1.0 has a completely different namespace (http://www.xml-cml.org/dtd/cml1_0_1.dtd). This allows the processing systems to decide exactly what version of the software to use. At present there is no universal method for automatically associating actual software (e.g. Java class libraries) with a namespace URI but this will certainly become common.

The second example (Chart 6) shows how different namespaces can be mixed in a composite document.

Here the default namespace is XHTML, and elements from another namespace (CMLCore) are contained in HTML elements (<li>). If a DTD were to be used to validate this document, it would mark it as invalid since the content model for <li> cannot contain CML elements. Using XML

Chart 6

```
<html
 xmlns="http://www.w3.org/1999/xhtml"
 xmlns:cml="http://www.xml-cml.org/schema/cml2/core">
<html>
 <p>We can supply the following set of molecules:</p>
 <ul>
 <li>
 <cml:molecule id="p1" title="phosphine">
 <cml:atomArray>
 <cml:atom elementType="P" hydrogenCount="3"/>
 </cml:atomArray>
 </cml:molecule>
 </li>
 <li><cml:molecule id="p2" title="penguinone"/></li>
 </ul>
</html>
```

Chart 7

```
<html
 xmlns="http://www.w3.org/1999/xhtml"
 xmlns:c="http://www.xml-cml.org/schema/cml2/core"
 xmlns:cml="http://www.penguinone.com/schema"
>
<html>
 <p>We can supply the following set of molecules:</p>
 <ul>
 <li>
 <c:molecule id="p1" title="phosphine">
 <c:atomArray>
 <c:atom elementType="P" hydrogenCount="3"/>
 </c:atomArray>
 </c:molecule>
 </li>
<!-- not a CML molecule -->
 <li><cml:molecule foo="p2" bar="penguinone"/></li>
 </ul>
</html>
```

Schemas and XSLT, the components can be separately validated. As an example, the following XPath expression will locate all CML molecule elements in a document, regardless of the other namespaces used:

```
select="//*[local-name()='molecule' and namespace-uri()=
                  'http://www.xml-cml.org/schema/cml2/core'"
```

(XPath is the syntax for addressing nodes within XML documents and is used by protocols such as XSLT and XQuery. Note that the namespace prefix is *not* required and it is dangerous to hard-code as in

```
select="cml:molecule"
```

as this will fail for any other (equally valid) prefix. Note also that other elements which happen to have the elementName molecule with a different namespace will not be selected, as in Chart 7.

Even though the second molecule has the (possibly misleading) prefix cml it does not belong to the CMLCore namespace and will not be retrieved by the XPath expression. This provides a mechanism (if required) for mixing different versions of CML, although this should always be done with caution.

5. THE NEED FOR VALIDATION

It has been estimated that a large proportion of software errors—up to 50%—arise from "invalid data", i.e., data input that does not conform to the concepts for which the software was written. Nonconformance can be due to data corruption, inputting data for a different version of a program, data out of range, or otherwise meaningless value. There are famous cases where use of the wrong scientific units have been catastrophic. Perhaps the most pernicious is when a user has a different interpretation of the semantics; the data appear valid but are actually inconsistent with the program.

Chart 8

```
<!-- the bond to be validated -->
<bond atomRefs="a15 a27"/>
...
<!-- validation -->
<!-- the first and second strings in atomRefs2 attribute -->
<xsl:variable name="at1" select="string-before(@atomRefs2,' ')"/>
<xsl:variable name="at2" select="string-after(@atomRefs2,' ')"/>
<xsl:variable name="grandparent" select="../../"/>

test="$grandParent/atomArray/atom[@id=$at1] and
 $grandParent/atomArray/atom[@id=$at2] and
 not($at1 = $at2)"
```

Validation goes a long way to solving these problems. Essentially a schema is a machine processable contract between the program author and the user. The programmer uses the schema to write the program, and we have found it to be a very powerful tool. In fact, for many schemas useful code can be automatically generated from the schema. If the data can be shown to be valid, then the program should process them reliably. If it does not, this is a program error. If the data are invalid, it is the user's responsibility. This is far more important than is generally realized and is a major tool in increasing the quality of software.

Validity is a machine-based concept. The more that the human view of validity can be formally encoded, the more useful it becomes. For example, are "T", "Phe", and "Bz" valid atom Types? Software often does not define openly what values are allowed. Using XML Schemas we can give a precise enumeration of allowed values. If the formalCharge on an atom is not given, is this an error or is there a defined default? (In CMLCore this defaults to 0;) Is "0.5" allowed? No, it must be specified as an integer.

XML Schemas allow two main types of validation:

1. Datatyping. The form and the value of the string must conform. XML Schema gives 43 datatypes, all of which can be used in STMML and CMLCore. In addition patterns can be imposed. For example a molecule ID could be required to be of the form

[A-Z]{6}([0-9][0-9])?

specifying six alpha characters followed by two optional digits (e.g. AABHTZ, ABCDEF01 from the Cambridge structural database). Note, however, that XML Schema has no support for scientific units as datatypes.

2. Document structure. It is possible to specify the attributes (and their datatypes and defaults) for all elements. An element's content model (the child nodes) can also be defined, with considerable power. A molecule can contain a formula, and a formula can contain either other formula or atomArray elements.

There are many rules however that cannot be enforced by these two approaches. The XSLT language, which supports the formulation of rules, is good at specifying constraints, including those which are context dependent. For example, a bond must reference two distinct existing atoms in the document. This is expressible in XSLT as in Chart 8. This consists of the following operations:

- find the first string in the atomRefs 2 attribute (in the example "a15") and assign to the symbolic variable at1
- repeat for the second string (at2 will be "a27")
- Find the grandParent element of the bond (should be ⟨molecule⟩, already enforced by the schema.)

Chart 9

```
defines a <bond> in the target document
Schema entry for bond
 <xsd:element name="bond" id="el.bond">
<xsd:annotation>
human readable documentation
 <xsd:documentation>
 <div class="summary">A bond between <a href="el.atom">atom</a>s, or between atoms and bonds</div>
 <div class="general"><p>
 <tt>bond</tt> is a child of <tt>bondArray</tt> and contains
 bond information. Bond must refer to at least two atoms (using
<a href="st.atomRefs2Type">atomRefs2</a>)
but may also refer to more for multicentre bonds. Bond is often EMPTY but
 may contain <a href="el.electron">electron</a>, <a href="el.length">length</a>
or <a href="el.stereo">stereo</a> elements.</p></div>
examples
 <div class="example">
 <pre>
 <bondArray>
 <bond id="b1" atomRefs2="a3 a8" order="D">
 <electron bondRef="b1"/>
 <BondStereo>C</BondStereo>
 </bond>
 <bond id="b2" atomRefs2="a3 a8" order="S">
 <BondStereo convention="MDL" conventionValue="6"/>
 </bond>
</bondArray>
</pre>
 </div>
 <div class="example">
 <pre>
 <!-- Zeise's salt: [Cl3Pt(CH2=CH2)]- -->
<atomArray>
 <atom id="pt1" elementType="Pt"/>
 <atom id="cl1" elementType="Cl"/>
 <atom id="cl2" elementType="Cl"/>
 <atom id="cl3" elementType="Cl"/>
 <atom id="c1" elementType="C" hydrogenCount="2"/>
 <atom id="c2" elementType="C" hydrogenCount="2"/>
</atomArray>
<bondArray>
 <bond id="b1" atomRefs2="c1 c2" order="D"/>
 <bond id="b2" atomRefs2="pt1 cl1" order="S"/>
 <bond id="b3" atomRefs2="pt1 cl2" order="S"/>
 <bond id="b4" atomRefs2="pt1 cl3" order="S"/>
 <bond id="b5" atomRefs="pt1" bondRefs="b1"/>
</bondArray>
 </pre>
 </div>
 </xsd:documentation>
machine-processable validation
 <xsd:appinfo id="val-bond">
 <val:comment>Validate Bonds</val:comment>
 <val:template match="bond">
 <val:comment>Atom Refs for 2-atom bond</val:comment>
 <val:variable name="at1" select="substring-before(normalize-space(@atomRefs2),' ')"/>
 <val:variable name="at2" select="substring-after(normalize-space(@atomRefs2),' ')"/>
 <val:comment>Do both atoms exist in current molecule context?</val:comment>
 <val:if test="not(key('atoms', $at1))">
 <val:call-template name="error">
 <val:with-param name="error">BOND (<val:value-of select="@id"/>):
ATOMREF not found: <val:value-of select="$at1"/></val:with-param>
 </val:call-template>
 </val:if>
 </val:template>
 </xsd:appinfo>
 </xsd:annotation>
contentModel (possible children)
 <xsd:complexType>
 <xsd:choice>
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
 <xsd:element ref="electron">
 <xsd:annotation>
 <xsd:documentation>
 <div class="summary">One or more electrons associated with the bond</div>
 <div class="general"><p>. The <a
href="st.bondRefType">bondRef</a> on the <tt>electron</tt> should
 point to the id on the bond. We may relax this later and allow
 reference by context.(We </p>
 </div>
 </xsd:documentation>
 </xsd:annotation>
 </xsd:element>
<xsd:element ref="BondStereo">
 <xsd:annotation>
 <xsd:documentation>
 <div class="summary">The stereo convention for the bond</div>
 <div class="general"><p>only one convention allowed</p>
 </div>
 </xsd:documentation>
 </xsd:annotation>
 </xsd:element>
```

Chart 9 (Continued)

```
<xsd:element ref="length">
 <xsd:annotation>
 <xsd:documentation>
 <div class="summary">the length between the atoms</div>
 <div class="general"><p>This is either an experimental measurement
 or used to build up internal coordinates (as in a z-matrix) (only one allowed)</p>
 <p>We expect to move length as a child of <a href="el.molecule">molecule</a> and
 remove it from here</p>
 </div>
 </xsd:documentation>
 </xsd:annotation>
 </xsd:element>
</xsd:choice>
elements in CML1 that have better replacements
 <!-- CML-1 (deprecated) -->
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
 <xsd:element ref="float" minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element ref="integer" minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element ref="string" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:choice>
 </xsd:choice>
generic attributes
<xsd:attributeGroup ref="tit_id_conv_dictGroup"/>
 <xsd:attributeGroup ref="ref"/>
attributes for bond (some omitted for conciseness)
 <xsd:attribute name="atomRefs2" type="atomRefs2Type" >
 <xsd:annotation>
 <xsd:documentation>
 <div class="summary">The two atoms in the bond</div>
 <div class="general"><p>. This will be the normal
reference attribute on the bond element. The order of atoms is preserved and may
 matter for some conventions (e.g. wedge/hatch or donor bonds)</p>
 </div>
 </xsd:documentation>
 </xsd:annotation>
 </xsd:attribute>
 <xsd:attribute name="atomRefs" type="atomRefArrayType">
 <xsd:annotation>
 <xsd:documentation>
 <div class="summary">The atoms in the bond</div>
 <div class="general"><p>. This is designed for multicentre bonds
(as in delocalised systems or electron-deficient centres.
The semantics are experimental at this stage.
As an example, a B-H-B bond might be described as
 <tt><bond atomRefs="b1 h2 b2"/></tt></p>
 </div>
 </xsd:documentation>
 </xsd:annotation>
 </xsd:attribute>
 <xsd:attribute name="order" type="orderType">
 <xsd:annotation>
 <xsd:documentation>
 <div class="summary">The order of the bond</div>
 <div class="general"><p>There is NO default. This order is for
 bookkeeping only and is not related to length, QM calculations
 or other experimental or theoretical calculations.
 see <a href="st.orderType"><tt>orderType</tt></a></p>
 </div>
 </xsd:documentation>
 </xsd:annotation>
 </xsd:attribute>
 </xsd:complexType>
 </xsd:element>
```

• Does it have an atom grandchild with id of $at1, and another with id of $at2 and are $at1 and $at2 different? If so the bond is valid.

These procedures may appear unfamiliar and verbose, but the benefit is that is gives a precise answer (yes/no) and is also openly readable. This means that the validation procedure is fully visible, and it should be absolutely clear whether given data conform or not.

Our schema design allows for the validation rules to be associated with elements and attributes (with the ⟨appinfo⟩ element). It is possible to create a complete validation tool automatically from the schema. We shall publish the complete XSLT validation suite separately—here we include one example of the approach (see Chart 9). Note, of course, that anyone authoring a program to output CML can include a validation against the schema before the data are output. This should ensure that the data output from one CML-aware program can be input into another CML-aware program without concern about data validity (against CMLCore). Also, authors may wish to add additional constraints to their input (e.g. limit the size of molecules, the atomic number of elements that can be processed, etc.).

5.1. XML Datatypes. STM data requires a range of datatypes, many of them representable by a character string such as "1,234E+05", "Sb", and "2001-12-31". In XML V1.0 these cannot be validated, but XML Schema introduces 43 datatypes for ⟨xsd:simpleType⟩s. If a value does not conform, the document is invalid. Among the constraints (some are referred to as facets) are as follows:

• The type (number, integer, nonNegativeInteger, date, URI, etc.)

• The value. An enumerated list of values can be given; alternatively maximum and/or minimum values can be set

Table 3. DataTypes for STMML and CML

name	description	example
	STMML	
coordinate2Type	a 2-D coordinate	1.2 2.3
coordinate3Type	a 3-D coordinate	1.2 2.3 3.4
dataTypeType	data types definable in a dictionary entry	xsd:float
delimiterType	delimiters for separating numbers	/
errorBasisType	basis of errors	range
errorValueType	type of errors	standard error
idType	an ID	a123
matrixType	matrix type	upperTriangular
namespaceRefType	namespaced data	core:mpt
unitsType	scientific units	units:ml
	CML	
atomRefType	atom reference by ID	a23
atomRefs2Type	reference to 2 atoms	a23 a31
atomRefs3Type	reference to 3 atoms	a23 a31 a34
atomRefs4Type	reference to 4 atoms	a23 a31 a22 a2
atomRefArrayType	reference to an array of atoms	a23 a31 a22 a45...
atomIDType	conformant atom ID	a23
bondRefType	reference to an extant bond	b17
bondRefArrayType	reference to an array of bonds	b17 b22 b34...
elementTypeType	element symbol	Sb
elementTypeArrayType	array of element symbols	F Cl Br I
formalChargeType	allowed charge	−3
formulaType	concise formula	H 2 S 1 O 4
hydrogenCountType	hydrogen count	1
isotopeType	allowed isotope value	13
lengthType	bond length	1.23
nonNegativeAngleType	unsigned angle	109.4
occupancyType	occupancy	0.73
orderType	bond order symbol	S
spacegroupType	spacegroup symbol	*P*21/*c*
stereoType	bond stereochemistry	W
substanceListType	role of substancelist	solution
torsionAngleType	allowed torsion angle	−320

• The lexical pattern. This conforms to a POSIX regular expression giving character types/values, and possible occurrence counts

It is possible to construct new datatypes *derived from* XML Schema datatypes. Examples of types defined in CML and STMML are listed in Table 3.

5.2. XML Structure. XML Schema constrains the structure of documents in two ways:

• Specifying the attributes that an element may have. The constraints on the structure are fairly similar to those expressible by DTDs, but there are additions to support inheritability of attributes.

• Specifying the content model of elements. There are several developments:

◦ The all connector goes some way toward specifying that certain child elements must be present, but that order is unimportant. We do not, yet, use this in CMLCore.

◦ It is possible to constraint elements by their namespace (e.g. to require that only certain CML elements are allowed as content, but that elements from other namespaces may or may not be forbidden). Thus it is possible to require that atom only contains atomParity or isotope from the CML namespace, but it could contain other elements from foreign namespaces not known explicitly. (An example might be SVG—each atom could contain a graphical object.)

◦ New complexTypes can be built and reused. This facility is not currently used in CMLCore, but as the number of derivative languages increases it is likely to be valuable.

5.3. XML Rules. An XML Schema is a set of rules to which a document must conform. It has hardcoded mechanisms for datatypes and the attributes and direct content of elements. Constraints outside these cannot be expressed in XML Schema. A common need is the conditional dependence of one part of the document on another part: "if part A has this structure and/or values, then part B must have that structure and/or values". A typical example is as follows: "if a molecule has fewer than four atoms, it cannot have any torsion elements". This requires a knowledge of the context which can be provided by adding XSLT rules.

6. THE CML SCHEMA (CMLCORE.XSD)

XML Schemas are complex (the formal XML Schema recommendations run to over 300 pages and require a 60 page primer to get started).[2] They take the form of XML documents, by convention given the .xsd suffix. We here introduce schemas in the form of examples from our specifications. Schemas are designed to be easily documented, and since they are in XML can be transformed into common formats such as XHTML and (using XSL-FO) into PDF. They can be searched and can be reorganized into different presentations. Documentation for the schema itself and for most components (elements, attributes, attributeGroups, etc.) can been produced using programs to extract the text directly from the schema and to associate it with the component being used, perhaps on an online help system or for automatic archival. The top-level element for this is annotation which can contain documentation for human readers and appinfo for machine processing. The latter is a major advance in that it allows a wide variety of functionality to be added on a per-element or per-attribute basis.

The schema definitions for both CMLCore and STMML use a subset of the XMLSchema vocabulary:

- "Top-level" elements
- **annotation.** Manages document for the *schema*
- **element.** Defines an element (cf. <!ELEMENT in DTDs)
- **attribute.** Defines an attribute (cf. <!ATTLIST in DTDs)
- **group**,**attributeGroup.** Mainly for ease of maintenance (cf. parameter entities in DTDs)
- **simpleType.** Defines a simpleType (essentially a datatyped character string. The type can be built-in (e.g. xsd:integer or can be derived from a builtin type (all are ultimately derived from xsd:string. simpleTypes can be used either in attribute values or for text content (#PCDATA in DTDs)
- **complexType.** Defines a complexType (a type built up from simpler components such as other elements and/or attributes)
- **Content-related.** These elements constrain the content of elements:
- **choice.** A series of alternatives (cf the "|" connector in DTDs)
- **sequence.** An ordered list of possible children (cf. the "," connector in DTDs)
- **all.** An unordered list of possible children (cf the "&" connector in SGML DTDs)
- **any.** Any child content (with constraints on namespaces (cf ANY in DTDs)

Occurrence counts for elements and the above constructs now have explicit attributes (minOccurs, maxOccurs) which allow for greater power in representing cardinality.

6.1. An Example—Part of the Bond Definition. This extract from the CMLCore Schema shows (Chart 9) many of the constructs that we use. It relies on previous definition of some simpleTypes and attributeGroups. Comments for this article are shown in italics.

CONCLUSIONS

We have described here the principles behind the recasting of chemical markup language into a core framework for handling molecule-based data, generic nonchemical components for carrying numeric data and extended chemical components which include chemical queries, reactions, spectra, and computational chemistry. XML Schemas can be used with generic processing tools to validate both the structure of XML documents containing CML data components and the datatypes of the components. For the first time, this approach allows a significant amount of chemistry to be formally validated without the need for custom software and opens the possibility of reliable numerical and chemical information interchange between diverse sources such as publishing and journals, instrument-derived data, extraction from databases, molecular modeling, and simulation, and grid-based[17] applications.

APPENDIX. ELEMENTS IN CML AND STMML: EACH ELEMENT (IN BOLD) IS FOLLOWED BY ITS CONTENT MODEL (ITALICS) AND ITS ATTRIBUTES

CML2 Elements

amount
- title
- id
- convention
- dictRef
- units

angle
- title
- id
- convention
- dictRef
- atomRefs3
- units
- errorValue
- errorBasis
- min
- max
- ref

array
- title
- id
- convention
- dictRef
- dataType
- errorValues
- errorBasis
- minValues
- maxValues
- units
- delimiter
- size
- ref

atom
((name|array|matrix|scalar|atomParity|electron)|(float*|integer*|string*)*)*
- id
- count
- elementType
- formalCharge
- hydrogenCount
- nonHydrogenCount
- isotope
- occupancy
- x2
- x3
- xFract
- xy2
- xyz3
- xyzFract
- y2
- y3
- yFract
- z3
- zFract
- title
- convention
- dictRef
- ref
- role

atomArray
(atom+|array|(floatArray*|integerArray*|stringArray*)*)*
- title
- id
- convention

- dictRef
- elementType
- count
- formalCharge
- hydrogenCount
- nonHydrogenCount
- isotope
- occupancy
- x2
- x3
- xFract
- y2
- y3
- yFract
- z3
- zFract
- atomID
- ref

atomParity
- title
- id
- convention
- dictRef
- atomRefs4

bond
((electron|bondStereo|length)|(float*|integer*|string*)*)*
- title
- id
- convention
- dictRef
- ref
- atomRefs2
- atomRefs
- bondRefs
- order

bondArray
(bond+|array|(floatArray*|integerArray*|stringArray*)*)*
- title
- id
- convention
- dictRef
- bondID
- atomRef1
- atomRef2
- order

bondStereo
- atomRefs4
- atomRefArray
- title
- id
- convention
- dictRef
- conventionValue

cml
*(, ANY [lax])**
- title
- id
- convention
- dictRef

crystal
(scalar{6,6},symmetry?)
- z
- title
- id
- convention
- dictRef

electron
()
- title
- id
- convention
- dictRef
- atomRef
- atomRefs
- bondRef
- bondRefs
- count
- ref

float
- builtin
- convention
- dictRef
- id
- title
- min
- max
- units
- unitsRef

floatArray
- builtin
- convention
- dictRef
- id
- title
- min
- max
- size
- units
- unitsRef

formula
((formula|atomArray))*
- title
- id
- convention
- dictRef
- count
- formalCharge
- concise

identifier
(ANY [lax])
- version
- tautomeric

integer
- builtin
- convention
- dictRef
- id
- title
- min
- max
- units
- unitsRef

integerArray
- builtin

- convention
- dictRef
- id
- title
- min
- max
- size
- units
- unitsRef

length
- title
- id
- convention
- dictRef
- atomRefs2
- units
- errorValue
- errorBasis
- min
- max
- ref

list
(ANY [lax])
- title
- id
- convention
- dictRef
- type

matrix
- dataType
- delimiter
- rows
- columns
- units
- title
- id
- convention
- dictRef
- matrixType
- errorValues
- errorBasis
- minValues
- maxValues

metadata
- name
- convention
- content

metadataList
(metadata+)

molecule
(metadataList,formula?,identifier?,name*,symmetry?, crystal?,(molecule*|(atomArray,bondArray?,electron*, (length|angle|torsion)*)),(scalar*,array*,matrix*,list*)*, (float*|integer*|string*)*)*
- dictRef
- convention
- title
- id
- ref
- formula
- count
- chirality
- formalCharge
- spinMultiplicity
- symmetryOriented name
- id
- convention
- dictRef
- role

observation
*(ANY [lax])**
- title
- id
- convention
- dictRef
- type
- count

property
(metadataList,name*,(scalar|array|matrix)*)*
- dictRef
- convention
- title
- id
- ref
- role
- state

propertyList
(metadataList,name*,(property|observation)*)*
- dictRef
- convention
- title
- id
- ref
- role

scalar
- title
- id
- convention
- dictRef
- dataType
- errorValue
- errorBasis
- min
- max
- units

string
- builtin
- convention
- dictRef
- id
- title

stringArray
- builtin
- convention
- dictRef
- id
- title
- min
- max
- size
- delimiter

substance
(metadataList,amount?,(molecule*|name*|property*))*
- dictRef

•convention
•title
•id
•type
•role
•ref
•count
•state

substanceList
(metadataList,amount?,substance*,propertyList?)*
•dictRef
•convention
•title
•id
•type
•ref
•role

symmetry
(matrix)*
•dictRef
•convention
•title
•id
•pointGroup
•spaceGroup
•irreducibleRepresentation
•number

torsion
•title
•id
•convention
•dictRef
•atomRefs4
•units
•errorValue
•errorBasis
•min
•max
•ref

STMML Elements

action
*(ANY [lax])**
•title
•id
•convention
•dictRef
•start
•startCondition
•duration
•end
•endCondition
•units
•count
•ref
•type

actionList
*(ANY [lax])**
•title
•id
•convention
•dictRef
•start
•startCondition
•duration
•end
•endCondition
•units
•count
•type
•order

alternative
•type

annotation
*(documentation|appinfo)**

appinfo
*(ANY [lax])**

array
•title
•id
•convention
•dictRef
•dataType
•errorValues
•errorBasis
•minValues
•maxValues
•units
•delimiter
•size
•ref

definition
*(ANY [lax])**
•source

description
*(ANY [lax])**
•class

dictionary
(unitList, annotation*, description*, entry*)*
•title
•id
•convention
•dictRef
•href

dimension
()
•name
•power

documentation
*(ANY [lax])**
•id

entry
((alternative| annotation| definition| description| enumeration| relatedEntry))*
•title
•id
•convention
•dataType
•rows
•columns
•recommendedUnits
•unitType
•minExclusive
•minInclusive
•maxExclusive

•maxInclusive
•totalDigits
•fractionDigits
•length
•minLength
•maxLength
•units
•whiteSpace
•pattern
•term

enumeration
(annotation)
•value

link
(ANY)
•title
•id
•convention
•dictRef
•from
•to
•ref
•role
•href
•type

list
(ANY [lax])
•title
•id
•convention
•dictRef
•type

matrix
•dataType
•delimiter
•rows
•columns
•units
•title
•id
•convention
•dictRef
•matrixType
•errorValues
•errorBasis
•minValues
•maxValues metadata
•name
•content

metadataList
(metadata+)

object
*(ANY [lax])**
•title
•id
•convention
•dictRef
•type
•count

observation
*(ANY [lax])**
•title
•id
•convention
•dictRef
•type
•count relatedEntry
•type
•href

scalar
•title
•id
•convention
•dictRef
•dataType
•errorValue
•errorBasis
•min
•max
•units

stmml
*(ANY [lax])**
•title
•id
•convention
•dictRef

table
(array+)
•rows
•columns
•title
•id
•convention
•dictRef

unit
*(description|annotation)**
•id
•abbreviation
•name
•parentSI
•unitType
•multiplierToSI
•constantToSI

unitList
(unitType,unit*)*
•title
•id
•convention
•dictRef
•href

unitType
(dimension)*
•id
•name

Obsolete CML

float
•builtin
•convention
•dictRef
•id
•title
•min
•max
•units

- unitsRef

floatArray
- builtin
- convention
- dictRef
- id
- title
- min
- max
- size
- units
- unitsRef

integer
- builtin
- convention
- dictRef
- id
- title
- min
- max
- units
- unitsRef

integerArray
- builtin
- convention
- dictRef
- id
- title
- min
- max
- size
- units
- unitsRef

string
- builtin
- convention
- dictRef
- id
- title

stringArray
- builtin
- convention
- dictRef
- id
- title
- min
- max
- size
- delimiter

Supporting Information Available: An XSD schema, examples, and an Acrobat version of the documented schema. The copyright for the Supporting Information is held by P. Murray-Rust and H. S. Rzepa. This material is available free of charge via the Internet at http://pubs.acs.org.

REFERENCES AND NOTES

(1) Murray-Rust, P.; Leach, C.; Rzepa, H. S. Chemical Markup Language. *Abstr. Pap. Am. Chem. Soc.* **1995**, *210*, 40-COMP Part 1.

(2) For formal definitions of all terms used in XML, along with specifications and tools, see the main W3C pages at http://www.w3c.org/.

(3) Murray-Rust, P.; Rzepa, H. S. *J. Chem. Inf. Comput. Sci.* **1999**, *39*, 928. For current specifications and supporting materials, see http://cml.sourceforge.net/ and http://www.xml-cml.org/.

(4) Murray-Rust, P.; Rzepa, H. S.; Wright, M.; Zara, S. *Chem. Comm.* **2000**, 1471−1472.

(5) Murray-Rust, P.; Rzepa, H. S.; Wright, M. *New J. Chem.* **2001**, 618−634.

(6) Murray-Rust, P.; Rzepa, H. S. *J. Chem. Inf. Comput. Sci.* **2001**, *41*, 1124.

(7) Gkoutos, G. V.; Murray-Rust, P.; Rzepa, H. S.; Viravaidya, C.; Wright, M. *Internet J. Chem.* **2001**, article 12.

(8) For reviews, see: Murray-Rust, P.; Rzepa, H. S. *Chem. Intl.* **2002**, *24*(4), 9−13. Liao, Y.-M.; Ghandan, Y. H. *Anal. Chem.* **2002**, *74*, 389A−390A. Murray-Rust, P.; Rzepa, H. S. Chapter. In *Chemoinformatics − From Data to Knowledge. Part 2. Advanced Topics*; Gasteiger, J., Ed.; 2003; in press.

(9) For the use of Schemas for medical applications, see: Hoelzer, S.; Schweiger, R. K.; Boettcher, H. A.; Tafazzoli, A. G.; Dudeck, J. *Med. Informatics Internet Medicine* **2001**, *26*(2), 131−46.

(10) Document Schema Definition Languages: http://www.dsdl.org/, Relax-NG: http://www.oasis-open.org/committees/relax-ng/compact-20021121.html, Schematron: http://www.ascc.net/xml/resource/schematron/schematron.html.

(11) For a specification and applications, see: http://www.w3.org/TR/SVG/.

(12) Anagnostaki, A.; Pavlopoulos, S.; Koutsouris, D. *Medinfo* **2001**, *10*, 77−81. See http://www.omg.org/.

(13) Stevens, R.; Goble, C.; Horrocks, I.; Bechhofer, S. *IEEE Trans. Inf. Technol. Biomedicine* **2002**, *6*(2), 129−34.

(14) Murray-Rust, P.; Rzepa, H. S. *Data Sci.* **2002**, *1*(1), 84−98; *1***(2)**, 1−65.

(15) Ruhl, A. M.; Schafer, R.; Kramer, G. W. *JALA* **2001**, *6*(6), 76−82. See, also: http://www.mel.nist.gov/div826/msid/sima/02_instrmnt_chem-refdata.html.

(16) Stein, S. E.; Heller, S. A.; Tchekhovskoi, D. V. *Abs. Pap. Am. Chem. Soc.* **2001**, *222*, Chicago, IL, United States, August 26−30, CINF-005.

(17) Talia, D. *IEEE Internet Computing* **2002**, *6*, 67−71. For details of Open Grid Services Archtecture, see: http://www.globus.org/ogsa/.

CI0256541