Processing CML conventions in Java

Egon L. Willighagen

Sophiaweg 132
NL-6523 NJ Nijmegen
The Netherlands

Abstract

This article describes the first opensource Java implementation of import filters for the Chemical Markup Language (CML). The filters support CML conventions and were tested with two opensource project: Jmol, a 3D molecular viewer, and JChemPaint, a chemical editor. Furthermore, the use of conventions in CML is explained and the reason for using conventions is pointed out. Finally, the implementation is compared with two recently developed techniques for handling CML data.

XML and CML

The eXtensible Markup Language (XML) is a relatively new international standard that is going to change things on the Internet. It was officially recommended by the World Wide Web Consortium (W3C) [1] in February 1998. Some of the important principles in designing XML was that it should support a wide variety of applications, XML documents should be easy process by written programs and XML design should be formal and concise. This design makes XML a convenient language to store information.

Chemical Markup Language (CML) is an XML language and was developed by P. Murray-Rust and H.S. Rzepa to store chemical information [2] in files. In May 1999 they released the first specification of this versatile markup language. Since then two chemical programs started to support CML: Jmol [3] and JChemPaint [4]. Jumbo was the first program that was able to handle CML documents. The first two versions [5a] are written in Java, but have not been updated to the CML 1.0 DTD. Recently, a third version has been released that does handle this DTD, but is written in JavaScript and does only run on specific systems [5b]. A PDB to CML conversion tool is also available [2c].

For both Jmol and JChemPaint import and export were written in 1998, but turned out to be inconvenient: CML is very flexible and the filters needed to be at least as flexible. E.g., the document type definition does not contain a convention on element dependencies nor on the meaning of data: “In this article we very deliberately do not attempt to develop or reconcile chemical ontologies” [2b]. Especially, the missing element dependencies makes parsing CML 1.0 rather difficult.

But readers of these CML documents, either programs or humans, still need to know what the data means. A direct solution for this problem is the use of conventions. “CML is designed to allow conversion from legacy files to CML without information loss” [2b]. Most CML elements have an *convention* attribute for which the value is set to CML by default. The CML convention refers to a abstract, convention-free representation of data. But in certain cases there is no convention-free representation.

For example, the bond order could be represented by the numbers one, two, three. Other types of bonds could be represented higher numbers. The MDL Molfile format [6], for example, uses the number four for aromatic bonds and five, six and seven for "single of double", "single or aromatic" and "double or aromatic" respectively. The bond type eight denotes "any" in this format. But without any notice of a convention one could not determine what four or five means.

In CML the default CML convention can be overwritten with the *convention* attribute mentioned above. Consider the CML document in the example below, which is generated by conversion of a PDB file with the PDB2CML conversion tool [2a].

Example 1. Example CML document using the PDB convention

```
<?xml version="1.0"?>
<!DOCTYPE list SYSTEM "cml.dtd" >
<!--CML conversion of file 75-09-2.pdb
Produced by PDB2CML (c) Steve Zara 1999-->
<list title="molecules" convention="PDB">
   <list title="model">
        <list title="compounds">
            <string title="compound"></string>
        </list>
        <list title="sequence" id="1">
            <list title="" id="1">
                <string builtin="residueType"></string>
                <atom title="atom" id="1">
                    <string title="name">C</string>
                    <coordinate3 builtin="xyz3">
                      -0.254 -0.572 0.101
                    </coordinate3>
                    <float builtin="occupancy">1.00</float>
                    <float title="tempFactor">0.00</float>
```

```
                </atom>
                <atom title="atom" id="2">
                  ... etc ...
                </atom>
              </list>
              <feature title="ter" id="6"/>
          </list>
          <list title="connections"/>
      </list>
</list>
```

To preserve all the data that was contained in the PDB will S.Zara developed this CML PDB convention. Not only does it use specifically features of PDB, like PDB's TER command, but the whole structure of this CML file is specific to the PDB convention. As can be seen the file has a specific structure of molecules-model-sequence-atoms. In the CML file this structure is maintained with the use of the CML's list elements.

A similar thing is done in the JMOL-ANIMATION convention. Each frame of the animation consists of a molecule structure which are all a child element of a list element in this example.

Example 2. Example CML document using the JMOL-ANIMATION convention

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE molecule SYSTEM "cml.dtd" [
  <!ATTLIST list convention CDATA #IMPLIED>
]>
<list convention="JMOL-ANIMATION">
  <molecule id="FRAME1">
    <string title="COMMENT">
      HEAT OF FORMATION = 38.45792 KCAL = 160.90796 KJ;
      FOR REACTION COORDINATE = 3.00000 ÅNGSTROMS
    </string>
    <atomArray>
      <stringArray builtin="id">
        a0 a1 a2 a3 a4 a5 a6 a7 a8
        a9 a10 a11 a12 a13 a14 a15
        a16 a17 a18 a19 a20 a21
        a22 a23 a24
      </stringArray>
      <stringArray builtin="elementType">
        C C C C C H C H H H H H H
        O H H O H H H H H H H H
      </stringArray>
```

```
      <floatArray builtin="x3">
        ... a lot of data ...
      </floatArray>
      <floatArray builtin="y3">
        ... a lot of data ...
      </floatArray>
      <floatArray builtin="z3">
        ... a lot of data ...
      </floatArray>
    </atomArray)
  </molecule>
  <molecule id="FRAME2">
  </molecule>
  ... etc ...
</list>
```

The use of conventions makes the processing unit able to determine whether or not these fragments are separate frames of an animation or if they are fragments of a molecular complex. But now an old problem resurfaces. In the near future a lot of conventions are bound to appear. Old ontologies will use new conventions, and new programs will use new conventions to be able to store chemical information they are not able to do or do not want to do in other conventions.

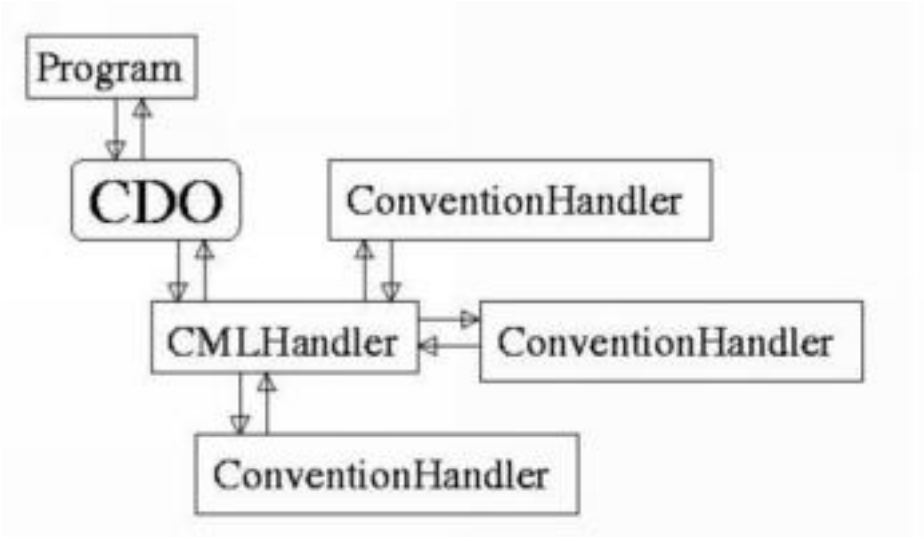
Processing conventions with SAX

Java is a convenient programming language that ports to most operating systems [7]. Software has been written in Java to parse XML data [8] and a programming interface, called SAX, has been written [9]. For both Jmol and JChemPaint I developed CML import filters written in Java that use the SAX interface. In 1999, I rewrote these filters to be able to support conventions for which the algorithms are described here.

SAX compliant parsers use an event-based API. For example, when the start of an element is encountered it signals a handler that processes the document with an startElement event. Other events are endElement, CharacterData and start- and endDocument. More information can be found at the SAX web site [9].

Figure 1. The CML import filter consists of a main handler that dynamically uses convention handlers to parse a CML file. Programs call upon this CMLHandler and use a chemical data

object (CDO) as an interface.



>

The Jmol and JChemPaint import filters consists of an interface with the chemical program (the CDO), a CMLHandler that dynamically uses an appropriate convention handler to process the CML file and store the data in the CDO. The CDO is an object that is specific written for one program. Both Jmol and JChemPaint have a separate CDO. The model can even extend from an object that is already provided by the program. The only restriction is that the object should implement the CDO programming interface [10].

The CDO programming interface is, like SAX, an event based interface. This means that it passes events like startDocument() or startObject() to the CDO, while it parser the CML document. This also means that information and conventions are handled before information is stored into memory.

The next Java code is taken from Jmol and shows how easy it is to use the CML import filter.

Example 3. Code from Jmol that exemplifies the use of the CML import filter.

```
1    public CMLFile(InputStream is) throws Exception {
2        super();
3
4        String pClass = "com.microstar.xml.SAXDriver";
5        InputSource input = new InputSource(is);
6        Parser parser = ParserFactory.makeParser(pClass);
7        EntityResolver resolver = new DTDResolver();
8        DocumentHandler handler =
9            new CMLHandler((CDOInterface)new JMolCDO());
```

```
10        parser.setEntityResolver(resolver);
11        parser.setDocumentHandler(handler);
12        parser.parse(input);
13        frames = (JMolCDO)((CMLHandler)handler).returnCDO();
14    }
```

A program that wants to use the CML import filter should only define a XML parser as is done in line 6 and if the document is to be checked for validity a DTDResolver. The handler that the parser uses actually processes the XML document and is of course the CMLHandler in our case. The CMLHandler takes a CDO that must implement the CDO Interface as stated as an argument (line 8). After the document is parsed in line 11 the CDO can be retrieved again from the CMLHandler and used to access the data. Note that *frames* is the native object in Jmol for the storage of all chemical data!

The internal structure of the handlers is the following. The SAX-events are passed to the root CMLHandler. The CMLHandler determines which convention should be used for this element and child element by checking the convention attribute. If there is a change in convention it dynamically loads a new convention handler, passes over all parsed data until then. In any case the next step will be passing on the event to the convention handler which might be new form then on. So the root handler does not actually process the CML file, it merely manages the processing. When a startElement event is raised it detects a change of convention and if necessary changes the convention handler.

Thus, this system always uses a convention handler that is specific for the convention given in the CML file. If no such convention handler is available it uses the default CML convention handler. Software using the CML import filter can register conventions itself that are not yet supported by the CML import filter. This plug in mechanism makes it possible to add convention handlers without having to upgrade the CML import filter. Handlers of conventions that are of general interest can be incorporated into the CML import filters distribution.

Alternatives

Recently, two alternatives have emerged. One is based on the XML Stylesheet Language (XSL) standard [11] in combination with JavaScript, a scripting language for web browsers [12]. This approach was taken in the Chimeral project [13]. The second alternative is based on the Document Object Model (DOM) [14], like XSL developed by the W3C. This approach was taken in the CML-DOM development [15].

XSL Transformations (XSLT) is a subsection of XSL, and deals with the transformations of the format in which data is stored: "A transformation expressed in XSLT describes rules for transforming a source tree into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree.

The result tree is separate from the source tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added." [12b]. However, besides structure transformation, it can also be used for simple data conversion, but compared to Java the possibilities are limited. The Chimeral project has shown that, in principle, it is possibly to use a combination of JavaScript and XSLT to read CML and convert it into other formats, like Scalable Vector Graphics [16] for depiction in web browsers.

In Java also XSLT stylesheets can be used for transformations [17], and thus the techniques used by Chimeral can be used in Java as well. But since XSLT is focused on direct transformation, it is unsuitable for storage in memory. Neither is is suitable for complex transformation, calculations or specialized searches. For these purposes storage in memory in likely to be necessary.

DOM, however, does do storage in memory. To be precise, it stores the exact XML data and structure in memory and software is expected to deal with the structure and content when necessary. This has the advantage that one can use standard XML techniques, like JDOM [18] or the chemistry specific, CML DOM. But a disadvantage is this standard is very general. The Java classes for DOM, for example, have a lot of functionality that is often not needed for specialized software. Moreover, a program may decide not to load all of the information in the document, but just the information it can handle or needs to handle. When loading a file with DOM, the program always reads the complete document.

The three alternatives, SAX based parsing, XSLT and DOM, have all different characteristics. It is not possible to say which one is best, partly because this depends on the type of problem. Since there had been no test on performance, it is wise to have all three alternatives in mind when generally dealing with CML. If dealing with CML conventions, for example, when multiple conventions are expected, parsing CML with the mechanism discussed in this article seems to be suitable best: data conversion needed when dealing with conventions is difficult in XSLT and since the DOM model does not not even handle conventions, but simply stores the data in memory. At this moment it is unclear how CML DOM is going to handle CML conventions.

Conclusion

CML is a new convenient electronic format to store chemical data. It literally is able to store all chemical data due to its flexible set up. However, due to this flexibility processing of CML is somewhat problematic. The use of convention attributes smoothes the processing of CML files. In this article a successful implementation in Java of a CML import filter that is convention aware, is described. Also, it is compared with two other CML processing techniques.

The latest filters can be downloaded from the Internet [19]. The source code is also freely available from this web site, with a GPL license [20]. The filters can be seen in action in the Jmol and JChemPaint software, mentioned earlier.

References

1. "The World Wide Web Consortium", *http://www.w3c.com/* (http://www.w3.org/)

2. a) "XML-CML Development", *http://www.xml-cml.org/* b) Murray-Rust, P.; Rzepa, H.S. *J.Chem.Inf.Comp.Sci*, 1999 39 928-942 c) "XML-CML Development - PDB2CML", Vers. 1.0, 6 October 2000, *http://www.xml-cml.org/software/pdb2cml.html*

3. "Jmol", Vers. 0.6.1, 3 February 2000, *http://www.openscience.org/jmol/*

4. a) "JChemPaint", Vers. 0.8, 4 January 2001, *http://jchempaint.sourceforge.net/* b) Steinbeck, C.; Krause, S.; Willighagen, E.L. *Molecules*, 2000 5 93-98

5. a) "Jumbo", Vers. 1.0 and Vers. 9802, 18 February 1998, *http://www.vsms.nottingham.ac.uk/vsms/java/jumbo/* b) "Jumbo", Vers. 3.0, 6 October 2000, *http://www.xml-cml.org/jumbo3/*

6. a) Dalby, A.; Nourse, J.G.; Hounshell, W.D.; Gushurst, A.K.; Grier, D.L.; Leland, B.A.; Laufer, J. *J.Chem.Inf.Comp.Sci*, 1992 244 b) "CTfile Formats", *http://www.mdli.com/downloads/literature/ctfile.pdf*

7. "java.sun.com - The Source for Java(TM) Technology", *http://java.sun.com/*

8. a) "OpenText's XML Parser", Vers. 1.0, 2 May 1998, *http://www.microstar.com/aelfred.html* b) "XML Parsers/Processors at XMLSOFTWARE", *http://www.xmlsoftware.com/parsers/*

9. a) "SAX: The Simple API for XML", Vers. 1.0, 11 May 1998, *http://www.megginson.com/SAX/SAX1/* b) "SAX: The Simple API for XML, version 2", Vers. 2.0, 5 May 2000, *http://www.megginson.com/SAX/*

10. "Chemical Data Object Programming Interface (CDOPI)", Version 1.2, 1 January 2000, *http://www.openscience.org/~egonw/cdopi/*

11. a) "Extensible Stylesheet Language (XSL)", *http://www.w3.org/Style/XSL/* b) "XSL Transformations (XSLT)", Vers. 1.0, 16 November 1999, *http://www.w3.org/TR/xslt.html*

12. JavaScript has many flavours by both Netscape and Microsoft and has been standardized in ECMAScript. "Standard ECMA-262: ECMAScript Language Specification", 3rd Edition, December 1999, *http://www.ecma.ch/ecma1/stand/ecma-262.htm*

13. a) "CML Transforms using Stylesheets and Applets", 22 September 2000, *http://www.ch.ic.ac.uk/chimeral/* b) Murray-Rust, P.; Rzepa, H S.; Wright, M.; Zara, S. *Chem.Comm.*, 20001471-1472

14. "Document Object Model (DOM) Level 1 Specification", Vers. 1.0, 1 October 1998, *http://www.w3.org/TR/REC-DOM-Level-1/*

15. "CML Document Object Model (DOM)", 11 January 2001, *http://www.xml-cml.org/cmldom/*

16. "Scalable Vector Graphics (SVG) 1.0 Specification, W3C Candidate Recommendation", Vers. 1.0 Candidate, 2 November 2000, *http://www.w3.org/TR/2000/CR-SVG-20001102/*

17. "Xalan-Java version 1.2.2", Vers. 1.2.2, 1 December 2000, *http://xml.apache.org/xalan/*

18. "JDOM", Vers. beta 5, 7 October 2000, *http://www.jdom.org/downloads/source.html*

19. "Java CML Filter Library", Vers. 1.2.6, 1 Januari 2001, *http://www.openscience.org/~egonw/cml/*

20. "GNU General Public License", Vers. 2, June 1991, *http://www.gnu.org/copyleft/gpl.html*