

Some Applications of XML

Peter Sestoft

Department of Natural Sciences, KVL

Case: Geographical data for area-based farmer subsidies ('hektarstøtte')

The farmer fills in two electronic forms:

- One that specifies field number, crop, field block number and area in hectares (to two decimal places).

These data are obtained from 'Bedriftsløsning Mark' as a comma-separated file, fetched from the homepage of Direktoratet for Fødevareerhverv (DDFE).

- A description of the precise form and location of the field.

These data are obtained via DFFE's web service markkort.glr-chr.dk in XML, from 'Dansk Markdatabase'.

The system was developed by DFFE and Dansk LandbrugsRådgivning, Landscentret.

(Thanks to Jens Bligaard from Landscentret for help with this).

Applications of XML

589 applications of XML are listed at <http://xml.coverpages.org/xmlApplications.html>

This list is far from exhaustive.

Here are some examples:

- Storing and exchanging geographical data; including 'hektarstøtte' computations.
- Scalable vector graphics (SVG): high-quality browser-independent graphics.
- Storing text documents and spreadsheets (OpenOffice, and future versions of Microsoft Word and Excel).
- Representing the structure of molecules (Chemical Markup Language), including biomolecules.
- Representing Bayesian networks and decision diagrams (XBN).
- Representing the result of biosequence database searches (NCBI Blast).
- Representing biosequences such as DNA and proteins (NCBI Genbank).
- Representing gene expression experiments: MicroArray and Gene Expression Markup Language (MAGE-ML)
- StatDataML: representation and exchange of statistical data (between R, MatLab, Octave, ...).

```
<?xml version="1.0" encoding="UTF-8"?>
<agr:FieldDesc ...>
  <agr:ApplicantNumber>123</agr:ApplicantNumber>
  <agr:FieldYear>2003</agr:FieldYear>
  <agr>UserAddress>
    <agr>UserFirstName>Peter</agr>UserFirstName>
    <agr>UserMiddleName>F.</agr>UserMiddleName>
    <agr>UserSurName>Farmer</agr>UserSurName>
    <agr>UserRoadName>Udkaersvej</agr>UserRoadName>
    <agr>UserHouseNumber>15</agr>UserHouseNumber>
    <agr>UserPostalCode>8200</agr>UserPostalCode>
    <agr>UserPostalDistrictName>Aarhus N</agr>UserPostalDistrictName>
  </agr>UserAddress>
  <agr:FieldGeography>
    <agr:FieldNumber>1-0</agr:FieldNumber>
    <agr:FieldCartesianArea>12345678</agr:FieldCartesianArea>
    <agr:FieldBlockId>
      <agr:SerialNumber>571229</agr:SerialNumber>
      <agr:SectionNumber>34</agr:SectionNumber>
    </agr:FieldBlockId>
    <agr:FieldOriginInfo>
      <agr:OriginInfoProducer>LR</agr:OriginInfoProducer>
      <agr:OriginInfoMethod>DMDB ver. 1.0</agr:OriginInfoMethod>
      <agr:OriginInfoMeasurePrecision agr:measureUnit="m">1</agr:OriginInfoMeasurePrecision>
    </agr:FieldOriginInfo>
    ... field polygon info ...
  </agr:FieldGeography>
</agr:FieldDesc>
```

Part of a field polygon description

```
<agr:FieldPolygon srsName="http://www.opengis.net/gml/srs/epsg.xml#82343">
  <gml:outerBoundaryIs>
    <gml:LinearRing>
      <gml:coordinates>
        571195.118702,6229449.30173 571209.318711,6229549.81452
        571213.614997,6229582.19897 571215.387937,6229602.48549
        571216.418716,6229609.98421 571218.373072,6229617.66289
        571220.261459,6229622.74202 ...
      </gml:coordinates>
    </gml:LinearRing>
  </gml:outerBoundaryIs>
  <gml:innerBoundaryIs>
    <gml:LinearRing>
      <gml:coordinates>
        571660.544206,6229456.99042 571694.386737,6229449.94162
        571694.568154,6229455.73063 571694.056887,6229460.33984 ...
      </gml:coordinates>
    </gml:LinearRing>
  </gml:innerBoundaryIs>
  ...
</agr:FieldPolygon>
```

The gml tags are from Geography Markup Language, an XML-based format.

It is promoted by the Open GIS Consortium (www.opengis.org)

Example SVG: Rectangle, circle, ellipse, line, polyline, ...

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" "http://www.w3.org/TR/SVG/DTD/svg10.dtd"
[
  <!-- include external DTDs -->
  <include href="http://www.w3.org/TR/SVG/DTD/svg10.dtd" type="DTD" />
  <include href="http://www.w3.org/2000/svg" type="definition" />
  <include href="http://www.w3.org/1999/xlink" type="definition" />
]
<svg viewBox="0 0 270 400"
  xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  <defs> ... </defs>
  <g id="mainlayer">
    <rect class="red str" x="15" y="15" width="100" height="50"/>
    <text class="fnt" x="44" y="88">rect</text>
    <rect class="blu str" x="150" y="15" width="100" height="50" rx="12" ry="18"/>
    <text class="fnt" x="140" y="88">rect (rounded)</text>
    <circle class="yel str" cx="62" cy="135" r="20"/>
    <text class="fnt" x="36" y="180">circle</text>
    <ellipse class="gre str" cx="200" cy="135" rx="50" ry="20"/>
    <text class="fnt" x="170" y="180">ellipse</text>
    ...
    <polyline style="fill:none; stroke:red; stroke-width:2" points="160,200 180,230 230,230"/>
    <text class="fnt" x="156" y="255">polyline</text>
  </g>
</svg>
```

Source <http://www.carto.net/papers/svg/samples/shapes.svg>

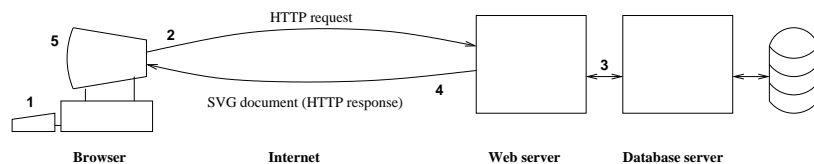
Case: Scalable Vector Graphics (SVG)

Representing drawings as images (.png, .jpg, .gif, ...) takes too much space and gives poor scalability.

SVG is an XML-based format for representing drawings, for making them interactive, and for animating them.

SVG can be displayed by plugins to most modern browsers (MS Internet Explorer, Mozilla).

An SVG document is just text, so it is easy for a Web server to generate high-quality graphics on the fly:



(1) User fills in form; (2) Request is sent to Web server; (3) Web server generates graphics as SVG document from database; (4) SVG document is sent back to browser; (5) browser interprets SVG document and displays (interactive) graphics.

Case: OpenOffice text documents

OpenOffice/StarOffice can read and write documents in MS Word format, but its own format is XML-based:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE office:document-content PUBLIC "-//OpenOffice.org//DTD OfficeDocument 1.0//EN" "http://openoffice.org/docbook/1.0/office.dtd" [
  <!-- include external DTDs -->
  <include href="http://openoffice.org/docbook/1.0/office.dtd" type="DTD" />
]
<office:document-content ... >
  <office:font-decls>
    ...
  </office:font-decls>
  <office:automatic-styles>
    <style:style style:name="T1" style:family="text">
      <style:properties fo:font-weight="bold"/>
    </style:style>
    <style:style style:name="T2" style:family="text">
      <style:properties fo:font-weight="normal"/>
    </style:style>
    ...
  </office:automatic-styles>
  <office:body>
    <text:p text:style-name="Standard">Simple
      <text:span text:style-name="T1">bold</text:span>
      <text:span text:style-name="T2"> or </text:span>
      <text:span text:style-name="T3">italic</text:span>
      <text:span text:style-name="T4"> text.</text:span>
    </text:p>
  </office:body>
</office:document-content>
```

Case: Microsoft Office WordML

Royalty-free XML schemas for MS Word 2003 were published 17 November 2003 at

<http://rep.oio.dk/Microsoft.com/officeschemas/Welcome.htm>

A minimal Word document:

```
<?xml version="1.0"?>
<?mso-application progid="Word.Document"?>
<w:wordDocument
  xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml">
  <w:body>
    <w:p>
      <w:r>
        <w:t>Hello, World.</w:t>
      </w:r>
    </w:p>
  </w:body>
</w:wordDocument>
```

The `?mso-application` processing directive tells MS Windows to use MS Word to open this document.

Note that the schema is different from the OpenOffice/StarOffice schema.

The use of XML does **not** guarantee that data are immediately interchangeable.

But XML, XML Schemas (tomorrow) and XSLT make transformation from one format to the other much easier.

XML in bioinformatics

Much of practical bioinformatics is concerned with parsing ill-structured text.

Either old flat databases or output from various programs.

For instance, the output from a biosequence database search (BlastP):

```
>gi|30021543|ref|NP_833174.1| Beta-lactamase, type II [Bacillus cereus ATCC 14579]
gi|29897098|gb|AAP10375.1| Beta-lactamase, type II [Bacillus cereus ATCC 14579]
      Length = 264
```

```
Score = 76.3 bits (186), Expect = 2e-13
Identities = 46/50 (92%), Positives = 46/50 (92%), Gaps = 3/50 (6%)
```

```
Query: 1 MKKNT--VGLCVGLLGTIQFVSTISSVQASQKVEKTVIYNETGTISISQ 47
      MKKNT VGLCVGLLGTIQFVSTISSVQASQKVEKTVI NETGTISISQ
Sbjct: 8 MKKNTLLKVGLCVGLLGTIQFVSTISSVQASQKVEKTVIKNETGTISISQ 57
```

...

Perl scripts are written again and again just to parse this kind of data.

For instance to extract the alignment, or make statistics on gap counts, ...

But Blast can output XML instead! A more sensible starting point for further processing...

E-government infrastructure: Offentlig Information Online (www.oio.dk)

Goals:

- Improved exchange of data within the public sector and between public and private enterprises
- Improved data processing and better access to existing data, and reuse of those.
- Easier implementation of e-services.

Means:

- Use XML-based formats for data exchange within the public sector in Denmark.

A committee supervises the design of XML-based formats.

See <http://www.oio.dk/XML/standardisering/XMLkomite/>

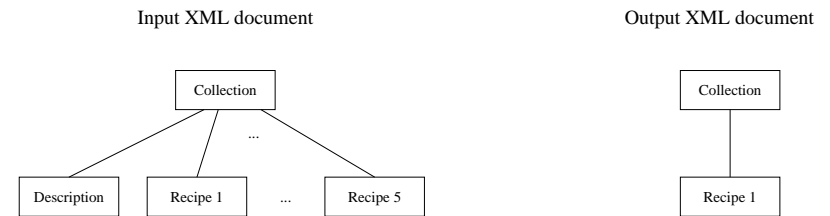
```
<Hit>
  <Hit_num>1</Hit_num>
  <Hit_id>gi|30021543|ref|NP_833174.1|</Hit_id>
  <Hit_def>Beta-lactamase, type II [Bacillus cereus ATCC 14579] &gt;gi|29897098|gb|AA
  <Hit_accession>NP_833174</Hit_accession>
  <Hit_len>264</Hit_len>
  <Hit_hsp>
    <Hsp>
      <Hsp_num>1</Hsp_num>
      <Hsp_bit-score>76.2554</Hsp_bit-score>
      <Hsp_score>186</Hsp_score>
      <Hsp_evalue>1.58742e-13</Hsp_evalue>
      <Hsp_query-from>1</Hsp_query-from>
      <Hsp_query-to>47</Hsp_query-to>
      <Hsp_hit-from>8</Hsp_hit-from>
      <Hsp_hit-to>57</Hsp_hit-to>
      <Hsp_query-frame>1</Hsp_query-frame>
      <Hsp_hit-frame>1</Hsp_hit-frame>
      <Hsp_identity>46</Hsp_identity>
      <Hsp_positive>46</Hsp_positive>
      <Hsp_gaps>3</Hsp_gaps>
      <Hsp_align-len>50</Hsp_align-len>
      <Hsp_qseq>MKKNT--VGLCVGLLGTIQFVSTISSVQASQKVEKTVIYNETGTISISQ</Hsp_qseq>
      <Hsp_hseq>MKKNTLLKVGLCVGLLGTIQFVSTISSVQASQKVEKTVIKNETGTISISQ</Hsp_hseq>
      <Hsp_midline>MKKNT VGLCVGLLGTIQFVSTISSVQASQKVEKTVI NETGTISISQ</Hsp_midline>
    </Hsp>
  </Hit_hsp>
</Hit>
...
```

Using XML from Java and Other Programming Languages

Peter Sestoft

Department of Natural Sciences, KVL

The intended transformation



In JDOM (and XML) a node can have only one parent.

Hence must a recipe must be detached from the old document before it can be added to the new document.

Using XML from the Java Programming Language

In Java, an XML document can be represented by an object tree (JDOM = Java Document Object Model).

Example: Extracting the first recipe from the recipe collection:

```
import java.io.*;
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;

class Example1 {
    public static void main(String[] args)
        throws IOException, JDOMException {
        Document d = new SAXBuilder().build(new File(args[0]));
        Namespace ns = Namespace.getNamespace("http://recipes.org");
        Element e = new Element("collection");
        e.addContent(d.getRootElement().getChild("recipe", ns).detach());
        Document n = new Document(e);
        new XMLOutputter().output(n, System.out);
    }
}
```

XML text document \implies Java objects \implies Java objects \implies XML (text) document.

Using Java to build an XML document from scratch

```
class Example2 {
    public static void main(String[] args)
        throws IOException, JDOMException {
        Element element = new Element("sequence");
        Element firstNumber = new Element("number");
        Element secondNumber = new Element("number");
        firstNumber.setText("3");
        secondNumber.setText("5");
        element.addContent(firstNumber);
        element.addContent(secondNumber);
        Document doc = new Document(element);
        new XMLOutputter().output(doc, System.out);
    }
}
```

This outputs the XML document doc in a fairly compact form:

```
<?xml version="1.0" encoding="UTF-8"?>
<sequence><number>3</number><number>5</number></sequence>
```

Or more readably, using `new XMLOutputter(Format.getPrettyFormat()).output(...)`:

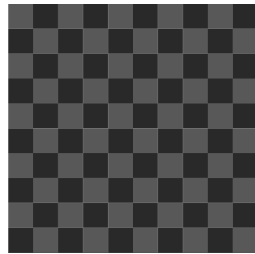
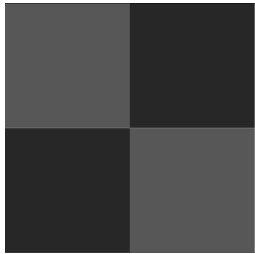
```
<?xml version="1.0" encoding="UTF-8"?>
<sequence>
  <number>3</number>
  <number>5</number>
</sequence>
```

JDOM: Attributes, Elements, Documents

Let's use Java to generate SVG documents of this form:

```
<?xml version="1.0" encoding="UTF-8"?>
<svg viewBox="0 0 40 40">
  <g>
    <rect fill="red" x="0" y="0" width="20" height="20" />
    <rect fill="blue" x="20" y="0" width="20" height="20" />
    <rect fill="blue" x="0" y="20" width="20" height="20" />
    <rect fill="red" x="20" y="20" width="20" height="20" />
  </g>
</svg>
```

The SVG file gives a 2-by-2 red/blue checkerboard (left); a 10-by-10 version is shown to the right:



Input and output from compressed files

Raw XML is very verbose. Compression can reduce file size by a factor 10–30.

In fact, OpenOffice/StarOffice documents are stored as compressed XML.

An OpenOffice document `simple.sxw` is actually a zip-file:

```
Archive:  simple.sxw
 Length   Date    Time    Name
-----
 2437    04-26-04  21:20   content.xml
 4619    04-26-04  21:20   styles.xml
 1097    04-26-04  21:20   meta.xml
 6247    04-26-04  21:20   settings.xml
  752    04-26-04  21:20   META-INF/manifest.xml
-----
 15152
          5 files
```

The essential parts of a Java program generating the SVG files

`setAttribute` adds an Attribute to an Element.

`addContent` adds an Element (child) to an Element.

```
int size = Integer.parseInt(args[0]);
Element svgElement = new Element("svg");
svgElement.setAttribute("viewBox", "0 0 " + 20*size + " " + 20 * size);
Element g = new Element("g");
svgElement.addContent(g);
for (int row=0; row<size; row++) {
  for (int col=0; col<size; col++) {
    Element rectElement = new Element("rect");
    String color = (row+col)%2==0 ? "red" : "blue";
    rectElement.setAttribute("fill", color);
    rectElement.setAttribute("x", Integer.toString(col*20));
    rectElement.setAttribute("y", Integer.toString(row*20));
    rectElement.setAttribute("width", "20");
    rectElement.setAttribute("height", "20");
    g.addContent(rectElement);
  }
}
Document doc = new Document(svgElement);
new XMLOutputter(Format.getPrettyFormat()).output(doc, System.out);
```

Compressing an XML file while writing it

Create an XML document containing 10 000 numbers;

```
int count = 10000;
Element element = new Element("sequence");
for (int i=0; i<count; i++) {
  Element number = new Element("number");
  number.setText(Integer.toString(i));
  element.addContent(number);
}
Document doc = new Document(element);

// Write uncompressed
XMLOutputter serializer = new XMLOutputter(Format.getPrettyFormat());
FileOutputStream fos = new FileOutputStream("numbers.xml");
serializer.output(doc, fos);
fos.close();

// Write compressed
ZipOutputStream zos = new ZipOutputStream(new FileOutputStream("numbers.zip"));
zos.putNextEntry(new ZipEntry("data.xml"));
serializer.output(doc, zos);
zos.close();
```

The uncompressed file is 248 957 bytes; the compressed one is 26 116 bytes (2.6 byte per number).

Compressed XML can be quite efficient for data interchange!

Using XML from the R statistics package

StatDataML: for data interchange between R, Matlab, Octave, SPLUS.

```
> library(StatDataML);
> data(chickwts);
> writeSDML(chickwts, "chicken.sdml");
```

SAS 8.1 has some limited XML import/export capabilities.

The StatDataML input/output preserves NaN and infinities:

```
> a <- (-3 : 3);
> a
[1] -3 -2 -1  0  1  2  3
> b <- 1/sqrt(a);
> b;
[1]      NaN      NaN      NaN      Inf 1.0000000 0.7071068 0.5773503
> writeSDML(b, "naninf.sdml");
> d <- readSDML("naninf.sdml");
```

Using XML from the Perl programming language

These facilities are in the Perl XML::DOM package.

XML documents are manipulated using a document object structure as in Java.

```
use XML::DOM;
my $parser = new XML::DOM::Parser;
my $doc = $parser->parsefile ("data.xml");

# print the (text) contents of 'number' children of 'sequence' elements

for my $seqnode ($doc->getElementsByTagName("sequence")) {
  for my $numbernode ($seqnode->getElementsByTagName("number")) {
    print $numbernode->getFirstChild()->getNodeValue() . "\n";
  }
}

# Print doc file
$doc->printToFile("out.xml");

# Print to string
print $doc->toString;
```