StatDataML: An XML Format for Statistical Data*

David Meyer[1], Friedrich Leisch[1], Torsten Hothorn[2] and Kurt Hornik[1,3]

[1] Institut für Statistik und Wahrscheinlichkeitstheorie, Technische Universität Wien, Wiedner Hauptstraße 8-10/1071, A-1040 Wien, Austria

[2] Institut für Medizininformatik, Biometrie und Epidemiologie, Friedrich-Alexander-Universität Erlangen-Nürnberg, Waldstraße 6, D-91054 Erlangen, Germany

[3] Institut für Statistik, Wirtschaftsuniversität Wien, Augasse 2-6, A-1090 Wien, Austria

Abstract. In order to circumvent common difficulties in exchanging statistical data between heterogeneous applications (format incompatibilities, technocentric data representation), we introduce an XML-based markup language for statistical data, called *StatDataML*. After comparing StatDataML to other data concepts, we detail the design which borrows from the language S, such that data objects are basically organized as recursive and non-recursive structures, and may also be supplemented with meta-information.

Keywords. Data Exchange, Data Design, XML

1 Introduction

Data exchange between different tools for data analysis and data manipulation is a common problem: different applications use different and often proprietary and undocumented formats for data storage. Import/export filters are often missing or insufficient, and if ever, focus on technical aspects (like storage modes and floating point specifications) in spite of supporting conceptional representation issues (like scales or representation of categorical data). The currently high costs for data exchange hence could be significantly reduced by the use of a well-defined common data exchange standard, if only because software packages would just need to provide one single mechanism.

The aim of this paper is to introduce such a data exchange standard for statistical data: the XML-based markup language *StatDataML*. The design borrows from the language S (see e.g., Chambers, 1998), such that data objects are basically organized as recursive structures (lists) and non-recursive structures (arrays), respectively. [1] Additionally, each object can have an attached list of properties (corresponding to S attributes), providing storage of meta-information.

*This research was supported by the Austrian Science Foundation (FWF) under grant SFB#010 ('Adaptive Information Systems and Modeling in Economics and Management Science').

[1] See Temple Lang & Gentleman (2001) for a more specific approach representing S objects in XML.

2 Requirements on Statistical Data

Statisticians need a data format which is both flexible enough to handle all different kinds of statistical data (from time series to decision trees), and specialized enough to incorporate statistical notions like scales and factors. Such a data format should feature:

- special symbols for infinities and undefined values
- special symbols for missingness ("not available")
- logical data
- categorical data (unordered, ordered or cyclic[2])
- numeric data (in the storage modes integer, real and complex)
- character data (strings)
- date/time information
- vectors (objects with elements of the same type)
- lists (objects with—possibly different—elements of any type)

Vectors should be indexable arbitrarily—in order to build matrices or multidimensional arrays. Lists allow for complex and even recursive structures (for they can contain lists again).

Table 1 compares some software products regarding these criteria: two families of mathematical programming languages (Splus[3]/R[4] and MATLAB[5]/ Octave[6]), statistical software (SPSS[7], SAS[8], Minitab[9]) and spreadsheets (Excel[10], StarCalc[11], Gnumeric[12]). In spreadsheets and MATLAB/Octave, nominal data can only be represented by strings. Arrays of arbitrary dimension are supported by Splus/R and MATLAB only. Complex numbers are only supported by Splus/R and MATLAB/Octave. The latter cannot handle missingness. IEEE special values are not supported by Excel, StarCalc, SPSS, SAS and Minitab.

3 StatDataML

3.1 StatDataML is XML

For "statistical data" one would usually think of such things as tabular data, time series objects, responses and regressors or contingency tables. Programs that produce such data store it on disk, using either a binary format or a text format. `StatDataML` files are XML files, thus ordinary text files, with extension `.sdml`, containing several XML elements (so called *tags*), which can be formally described with a special data definition language (DTD)—see the World Wide Web Consortium (2000) recommendation. Note that Quoting is

[2] like days of a week
[3] For Splus see: `http://www.insightful.com`
[4] For R see: Ihaka & Gentleman (1996) and `http://www.R-project.org`
[5] For MATLAB see: `http://www.mathworks.com`
[6] For Octave see: `http://www.octave.org`
[7] For SPSS see: `http://www.spss.com`
[8] For SAS see: `http://www.sas.com`
[9] For Minitab see: `http://www.minitab.com`
[10] For Excel see: `http://www.microsoft.com`
[11] For StarCalc see: `http://www.staroffice.com` or `http://www.openoffice.com`
[12] For Gnumeric see: `http://www.gnumeric.org`

	R/Splus	MATLAB	Octave	Excel/ StarCalc	Gnumeric	SPSS	SAS	Minitab
IEEE	yes	yes	yes	no	yes	no	no	no
NA	yes	no	no	yes	yes	yes	yes	yes
logical	yes	(yes)	no	yes	yes	yes	yes	yes
nominal		strings	strings	strings	strings	coding	yes	strings
unordered	yes					yes	yes	yes
ordered	yes					yes	yes	yes
cyclic	no	no	no	no	no	no	no	no
numeric	yes	yes			yes			
integer	yes	yes	no	no	no	(yes)	yes	no
real	yes	yes	yes	yes	yes	yes	yes	yes
complex	yes	yes	yes	no	no	no	no	no
character	yes	yes	yes	yes	yes	yes	yes	yes
date/time	yes	yes	yes	yes	yes	yes	yes	yes
arrays	yes	yes	no	no	no	no	no	no
matrix	yes	yes	yes	yes	yes	yes	yes	yes
lists	yes	yes	yes	no	no	no	no	no

Table 1: Data representation capabilites of different software packages

needed for the special XML characters &, < and > by using `&`, `<` and `>`, respectively.

character	quote
&	&amp;
<	&lt;
>	&gt;

Table 2: Quoting characters in StatDataML

In the following, we will go through the rules in the `StatDataML.dtd` file (the DTD as a whole is given in the Appendix).

3.2 The File Header

```
<!ELEMENT StatDataML (description?, dataset?)>
```

The top level `StatDataML` element contains one `description` and one `dataset` element, each optional. It should contain the `StatDataML` namespace:

```
<StatDataML xmlns="http://www.ci.tuwien.ac.at/StatDataML">
...
</StatDataML>
```

(The URL defining the name space does not physically exist; its only purpose is to guarantee a unique name.)

3.3 The `description` element

```
<!ELEMENT description (title?, source?, date?, version?,
comment?, creator?, class?, properties?)>

<!ELEMENT title (#PCDATA)>
<!ELEMENT source (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT creator (#PCDATA)>
<!ELEMENT class (#PCDATA)>
<!ELEMENT properties (list)>
```

The `description` element is used to provide meta-information about a dataset that is typically not needed for computations on the data itself. It consists of eight elements: `title, source, date, comment, version, creator` and `class` are simple strings (`PCDATA`), whereas `properties` is a `list` element (see next section). `date` should follow the ISO 8601 format (see below). The `creator` element should contain knowledge about the creating application and the `StatDataML` implementation, `properties` offers a well-defined structure to save application-based meta-information, and, finally, the `class` element will contain the class name, if any. There are some discussions about meta data in statistics[13]: one could think of extending the `description` element in such that extended information with logical markup can be stored.

3.4 The `dataset` element

```
<!ELEMENT dataset (list | array)>
```

We define a `dataset` element either as an array or as a list. We use arrays and lists as basic "data types" in StatDataML because every data object in statistics can be decomposed into a set of arrays and lists (as in Splus/R, or like the corresponding arrays and cell-arrays in MATLAB). The basic property of a list is its recursive structure, in contrast to arrays which are always non-recursive. If one thinks about data as a tree, lists would be the branches and arrays the leaves.

3.4.1 Lists

A list contains three elements: `dimension, properties` and `listdata`:

```
<!ELEMENT list (dimension, properties?, listdata)>
<!ELEMENT listdata (list | array | empty)*>

<!ELEMENT dimension (dim*)>
<!ELEMENT dim (e*)>
<!ATTLIST dim size CDATA #REQUIRED>
<!ATTLIST dim name CDATA #IMPLIED>
```

The `dimension` element contains one or more `dim` tags, depending on the number of dimensions. Each of them has `size` as a required attribute, and may

[13] e.g., `http://www.gla.ac.uk/External/RSS/RSScomp/metamtg.html`

optionally contain up to `size` names, specified with `<e>...</e>` tags. In addition, the dimension as a whole can be attributed a name by the optional `name` attribute. Note that arrays, like the whole dataset, can also have additional `properties` attached, corresponding, e.g., to attributes in S. The `listdata` element may either contain arrays (with the actual data), or again lists, which allows for complex and even recursive structures, or `empty` tags indicating non-existing elements, corresponding to `NULL` in S.

3.4.2 Arrays

```
<!ELEMENT array (dimension, properties?,
                 nominalCategories?, (data | textdata))>
```

Arrays are blocks of data objects of the same elementary type with dimension information used for memory allocation and data access (indexing). The first two elements, `dimension` and `properties`, are identical to `list`. The `listdata` block gets replaced by the `data` (or `textdata`) element which contains the data itself. In addition, there is an optional element for categorical data, `nominalCategories`, which specifies the levels coding for categorical data:

```
<!ELEMENT nominalCategories (label)+>
<!ELEMENT label (#PCDATA)>
<!ATTLIST label code CDATA #REQUIRED>
```

The `label` element has a mandatory `code` attribute specifying the levels' integer value, and contains a name (which is optional). If no name is given, the application should use the numerical code instead. The order of the `label` elements also defines the ordering relation of the levels for ordinal data.

3.4.3 The `data` tag

```
<!ENTITY % TYPES "logical|nominal|numeric|character|datetime">
<!ENTITY % MODES "unordered|ordered|cyclic|integer|real|complex">

<!ELEMENT data (e|ce|na)* >
<!ATTLIST data true  CDATA "1"
               false CDATA "0"
               type (%TYPES;) "character"
               mode (%MODES;) #IMPLIED>

<!ELEMENT na EMPTY>

<!ENTITY % REAL "#PCDATA|posinf|neginf|nan">

<!ELEMENT e (%REAL;)* >
<!ELEMENT posinf EMPTY>
<!ELEMENT neginf EMPTY>
<!ELEMENT nan EMPTY>

<!ELEMENT ce (r,i) >
<!ELEMENT r (%REAL;)* >
<!ELEMENT i (%REAL;)* >
```

If `data` is used (especially recommended for character data), then each element of the array representing an existing value is encapsulated in `<e>...</e>` (or `<ce>...</ce>` for complex numbers). For missing values, `<na/>` has to be used, empty values are just represented by `<e></e>` (or simply `<e/>`).
The `type` attribute specifies the statistical data type, as logical, nominal, numeric, character and date/time. The optional `mode` attribute allows for further specification: `nominal` data could be `unordered` ("factors"), `ordered` and `cyclic` (e.g., days of week), whereas `numeric` data could be `integer`, `real` or `complex`.
As an example, consider a character dataset formed by color names, with one value missing (after `green`), and one being empty (after `blue`). The corresponding `data` section would appear as follows:

```
<data type="nominal" mode="unordered">
<e>red</e> <e>green</e> <na/> <e>blue</e> <e></e> <e>yellow<e>
</data>
```

IEEE Number Format

The implementation is responsible for the correct casts. The number format has to follow the IEEE Standard for Binary Floating Point Arithmetic (see Institute of Electrical and Electronics Engineers, 1985), which is implemented by most programming languages and system libraries. However, the IEEE special values `+Inf`, `-Inf` and `NaN` must explicitly be specified by `<posinf/>`, `<neginf/>` and `<nan/>`, respectively, to facilitate the parsing process in case the IEEE standard were not implemented. These special values could appear as follows:

```
<data type="numeric" mode="real">
  <e>1.23</e> <e><posinf/></e> <e><nan/></e> <e>2.43</e>
</data>
```

Complex Numbers

Complex numbers are enclosed in `<ce>...</ce>` tags which contain exactly one `<r>...</r>` tag (for the real part) and one `<i>...</i>` tag (for the imaginary part). Apart from that, the same rules as for `<e>...</e>` apply:

```
<data type="numeric" mode"=complex">
  <ce> <r>12.4</r> <i>1</i> </ce>
</data>
```

Logical Values

The `true`- and `false`-attributes can be used to change the default representation of logical values (`1` and `0`).

```
<data type="logical" true="T" false="F">
  <e>T</e> <e>F</e>
</data>
```

Date and Time Information

Data of type `datetime` has to follow the ISO 8601 specification (see International Organization for Standardization, 1997). StatDataML should only make use of the complete representation in extended format of the combined calendar date and time of the day representation:

CCYY-MM-DDThh:mm:ss±hh:mm

where the characters represent Century (C), Year (Y), Month (M), Day (D), Time designator (T; indicates the start of time elements), Hour (h), Minutes (m) and Seconds (s).

[C] ... Century
[Y] ... Year
[M] ... Month
[D] ... Day
[T] ... Time designator (indicates the start of time elements)
[h] ... Hour
[m] ... Minutes
[s] ... Seconds

For example, the 12th of March 2001 at 12 hours and 53 minutes, UTC+1, would be represented as: 2001-03-12T12:53:00+01:00 .

3.4.4 The `textdata` tag

For (memory/storage space) efficiency we also define `textdata`, a second way of writing data blocks using arbitrary characters (typically whitespace) for separating elements instead of `<e>...</e>`.

```
<!ELEMENT textdata (#PCDATA) >
<!ATTLIST textdata sep           CDATA " \n"
                   na.string     CDATA "NA"
                   null.string   CDATA "NULL"
                   posinf.string CDATA "+Inf"
                   neginf.string CDATA "-Inf"
                   nan.string    CDATA "NaN"
                   true          CDATA "1"
                   false         CDATA "0"

                   type (%TYPES;) "character"
                   mode (%MODES;) #IMPLIED>
```

In this case the complete data block is included in a single XML tag; because only a single character is used as separator, one needs 6 bytes less per element. The use of `textdata` even provides more compact results when compression tools (like *zip*) are used, and is recommended if such tools are not available or if their use is not desirable. The set of separator characters is defined by the optional attribute `sep`. The attributes `na.string` and `null.string` define the strings to be interpreted as missing or empty values (default: `NA` and `NULL`). `posinf.string`, `neginf.string` and `nan.string` are used to specify the corresponding IEEE special values. An additional "advantage" is that textdata blocks are not parsed by the XML parser, which can drastically reduce the

memory footprint when reading a file, because many parsers represent the complete XML data as a nested tree. This results in one branch for each array element and typically needs much more memory than just the element itself. Our color-example could look similar to following:

```
<textdata na.string="N/A" null.string="EMPTY" mode=character>
red green EMPTY blue N/A yellow
</textdata>
```

We could also have defined a different set of separator symbols with the `sep`-attribute, e.g. colons or semi-colons.

3.5 Implementation issues

Interfaces implementing StatDataML should provide options for setting conversion strings for the `NA`, $\pm\infty$ and `NaN` entities if they are not supported, but with no defaults. Unsupported elements with no default conversion should cause an error, thus forcing the user to explicitly specify a conversion rule. All conversions effectively done should be reported by a warning message.

4 Examples

4.1 Demo Sessions

We show how a sample task (creating a simple matrix, writing a StatDataML-file in MATLAB, reading this file in R) is realized.

4.1.1 A session with R

```
    R : Copyright 2001, The R Development Core Team
Version 1.3.0 Under development (unstable) (2001-03-20)

## load the StatDataML-package

> library (StatDataML)
Loading required package: XML

## Create a simple data structure

> X <- list(matrix (c(1,2,3,4),2,2))
> X[[2]] <- 12 + 3i
> X[[3]] <- "Test"
> X[[4]] <- list (a="Test 2", b=33.44)
> dim(X) <- c(2,2)

## Show what we got

> X
     [,1]        [,2]
[1,] "Numeric,4" "Character,1"
[2,] "Complex,1" "List,2"

> X[[1,1]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> X[[1,2]]
[1] "Test"

> X[[2,1]]
[1] 12+3i

> X[[2,2]]
$a
[1] "Test 2"

$b
[1] 33.44

## write it to the .sdml file

> writeSDML (X, "test.sdml")
```

4.1.2 A session with MATLAB

```
                    < M A T L A B >
          Copyright 1984-2000 The MathWorks, Inc.
               Version 6.0.0.88 Release 12

>> path (path, 'StatDataML')
>> X = readsdml ('test.sdml')

X =

          [2x2 double]    'Test'
    [12.0000+ 3.0000i]    [1x1 struct]

>> X{1}

ans =

     1     3
     2     4

>> X{2}

ans =

  12.0000 + 3.0000i

>> X{3}

ans =

Test

>> X{4}

ans =

    a: 'Test 2'
    b: 33.4400
```

4.2 Sample output

4.2.1 The integers from 1 to 10

```
<?xml version="1.0"?>
<!DOCTYPE StatDataML PUBLIC "StatDataML.dtd" "StatDataML.dtd">

<StatDataML xmlns="http://www.omega.org/StatDataML/">

<description>
  <title>The integers from 1 to 10</title>
  <source>MATLAB</source>
  <date>2001-10-10T14:40:01+0200</date>
  <version></version>
  <comment></comment>
  <creator>MATLAB-6.0.0.88 (R12):StatDataML_1.0-0</creator>
  <class></class>
</description>

<dataset>
  <array>
    <dimension>
      <dim size="10"></dim>
    </dimension>
    <data type="numeric" mode="integer">
      <e>1</e> <e>2</e> <e>3</e> <e>4</e> <e>5</e>
      <e>6</e> <e>7</e> <e>8</e> <e>9</e> <e>10</e>
    </data>
  </array>
</dataset>

</StatDataML>
```

4.2.2 A more complex example

The following example represents a table with two variables (which could be, e.g., a dataframe in S and a structure in MATLAB): one factor `a` with levels `fa` and `fb`, and one numeric variable.

a	fa	fa	fa	fa	fa	fb	fb	fb	fb	fb
b	0.5	$+\infty$	4.5	NaN	1.0	3.0	3.2	1.3	2.4	3.5

```
<?xml version="1.0"?>
<!DOCTYPE StatDataML PUBLIC "StatDataML.dtd" "StatDataML.dtd">

<StatDataML xmlns="http://www.omega.org/StatDataML/">

<description>
  <title>A small dataframe</title>
  <source>MATLAB</source>
  <date>2001-10-10T14:43:02+0200</date>
  <version></version>
  <comment></comment>
  <creator>MATLAB-6.0.0.88 (R12):StatDataML_1.0-0</creator>
  <class></class>
</description>

<dataset>
  <list>
    <dimension>
      <dim size="2"> <e>a</e> <e>b</e> </dim>
    </dimension>

    <listdata>
      <array>
        <dimension>
          <dim size="10"></dim>
        </dimension>
        <data type="character">
          <e>fa</e> <e>fa</e> <e>fa</e> <e>fa</e> <e>fa</e>
          <e>fb</e> <e>fb</e> <e>fb</e> <e>fb</e> <e>fb</e>
        </data>
      </array>
      <array>
        <dimension>
          <dim size="10"></dim>
        </dimension>
        <data type="numeric" mode="real">
          <e>0.5</e> <e><posinf/></e> <e>4.5</e> <e><nan/></e> <e>1.0</e>
          <e>3.0</e> <e>3.2</e>       <e>1.3</e> <e>2.4</e>    <e>3.5</e>
        </data>
      </array>
    </listdata>
  </list>
</dataset>

</StatDataML>
```

5 Resources

- Duncan Temple Lang's XML package provides general XML parsing for S engines (`http://cran.r-project.org/src/contrib/Omegahat/XML.tar.gz`)
- The package StatDataML provides a beta implementation of StatDataML I/O routines for Splus/R and MATLAB/Octave. The two functions `writeSDML` and `readSDML` implement writing and reading for StatDataML-files. With this implementation it will be possible to write and read R data objects without loss of information (`http://cran.r-project.org/src/contrib/Devel/StatDataML_0.3-1.tar.gz`)
- The XML C library for gnome (`http://www.xmlsoft.org/`)

6 Conclusion: Limitations and Extensionability

StatDataML seems general and flexible enough to cover most of statisticians' data representation needs. Currently we have support for Splus, R, MATLAB and Octave (the software is available at `http://www.omegahat.org/`), and converters for SPSS and Gnumeric are under development. There are some limitations, though:

Especially for large datasets, a distributed data structure could be helpful: e.g., one could think about just distributing the description part of a StatDataML-file, allowing the receiver to decide on whether to retrieve the data or not. Another application would be data subject to continous change, using StatDataML-files as structured link-lists. Both should easily be possible using the standardized "XInclude" specification, which offers a general link tag for XML files. But there will be still a remaining issue: how may user specify (possibly remote) database access? At least, we would need some query-information (e.g., in SQL) supplied along with the database URL.

Furthermore, it would be helpful to have some form of authentication, which means that everyone can read a `StatDataML` file but cannot manipulate the data without violating the signature. Our opionion is that this problem should not be solved within StatDataML. One could make use, e.g., of "XML signature", which seems to be an appropriate solution.

Fnally, within `StatDataML.dtd` we describe how a basic dataset should be organized. We (currently) do not provide definitions for classes like a dataframe or time series in DTD format. To model this, we would like to have a principle of inheritance from `dataset`, such that the basic DTD can be extended or restricted and an XML parser can validate objects of certain classes. To our knowledge, this can not be done with standard XML—restrictions necessitate the specification of a new DTD.

Appendix: the StatDataML `.dtd` file

```
<!-- StatDataML DTD version="0.3" -->

<!ELEMENT StatDataML (description?, dataset?)>

<!ELEMENT description (title?, source?, date?, version?, comment?,
```

```
                      creator?, class?, properties?)>
<!ELEMENT title (#PCDATA) >
<!ELEMENT source (#PCDATA) >
<!ELEMENT date (#PCDATA) >
<!ELEMENT version (#PCDATA) >
<!ELEMENT comment (#PCDATA) >
<!ELEMENT creator (#PCDATA) >
<!ELEMENT class (#PCDATA) >

<!ELEMENT dataset (list | array)>

<!ELEMENT list (dimension, properties?, listdata) >
<!ELEMENT listdata (list | array | empty)*>

<!ELEMENT empty EMPTY>

<!ELEMENT array (dimension, properties?, nominalCategories?,
                 (data | textdata))>

<!ELEMENT nominalCategories (label)+>
<!ELEMENT label (#PCDATA)>
<!ATTLIST label code CDATA #REQUIRED>

<!ENTITY % TYPES "logical|nominal|numeric|character|datetime">
<!ENTITY % MODES "unordered|ordered|cyclic|integer|real|complex">

<!ELEMENT data (e|ce|na)* >
<!ATTLIST data true  CDATA "1"
               false CDATA "0"
               type (%TYPES;) "character"
               mode (%MODES;) #IMPLIED>

<!ELEMENT textdata (#PCDATA) >
<!ATTLIST textdata sep CDATA " \n"
                   na.string CDATA "NA"
                   null.string CDATA "NULL"
                   posinf.string CDATA "+Inf"
                   neginf.string CDATA "-Inf"
                   nan.string    CDATA "NaN"
                   true CDATA "1"
                   false CDATA "0"
                   type (%TYPES;) "character"
                   mode (%MODES;) #IMPLIED>

<!ELEMENT na EMPTY>

<!ENTITY % REAL "#PCDATA|posinf|neginf|nan">

<!ELEMENT e (%REAL;)* >
<!ELEMENT posinf EMPTY>
<!ELEMENT neginf EMPTY>
<!ELEMENT nan EMPTY>

<!ELEMENT ce (r,i) >
```

```
<!ELEMENT r (%REAL;)* >
<!ELEMENT i (%REAL;)* >

<!ELEMENT dimension (dim*)>
<!ELEMENT dim (e*)>
<!ATTLIST dim size CDATA #REQUIRED>
<!ATTLIST dim name CDATA #IMPLIED>

<!ELEMENT properties (list)>
```

References

Chambers, J. M. (1998). *Programming with Data: a guide to the S Language*. Springer.

Ihaka, R. & Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, **5**(3), 299–314.

Institute of Electrical and Electronics Engineers (1985). *IEEE Standard 754-1985 (R 1990), Standard for Binary Floating-Point Arithmetic*.

International Organization for Standardization (1997). *ISO 8601:1997, Data elements and interchange formats - Information Interchange - Representation of dates and times*.

Temple Lang, D. & Gentleman, R. (2001). RSXMLObjects: Reading and writing S objects in XML. S Package. http://www.omegahat.org/RSXMLObjects.

World Wide Web Consortium (2000). *Extensible Markup Language (XML), 1.0 (2nd Edition)*. http://www.w3.org.