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

Summary

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 that 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 (such as storage modes and floating point specifications) in spite of supporting conceptional representation issues (such as 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

*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.

list of properties (corresponding to S attributes), providing storage of meta-information.

Interestingly, data exchange of *statistical* data *per se* did not get much attention in the literature so far. On the other hand, there has been considerable interest in meta-data in statistics[2]. This is a closely related topic, because data exchange necessarily results in giving information about the data structure, which simply is meta-data.

Kent & Schuerhoff (1997) introduced a useful typology by distinguishing: conceptual meta-data (for definition and standardizing statistical concepts), operational meta-data (for automating statistical activities), logistic meta-data (for storing, moving, and retrieving data), documentary meta-data (for the end-user), and processing meta-data (about dynamic aspects of statistical processing—for the latter, see Grossman, 2000). Our work is a contribution to the third category: logistic meta-data for data storage.

One of the first attempts trying to provide an XML-based meta-data structure is the Data Documentation Initiative (DDI—see, e.g., Thomas & Block, 2001). The DDI project participates in the metanet project[3], aiming at developing standards for describing statistical meta-data and statistical information systems. The DDI project provides an XML specification for whole social sciences data collections[4], and consists of 5 main sections: document description (describing the meta-data itself), study description (providing information about the described data collection such as source, copyrights and keywords), data files description (physical format, layout and structure of the data files), variable description (data type, missing values, summary statistics, ...), and a section for other material (reports, publications). Although a wide range of meta-data is covered by this standard, it is biased towards survey data: the data format has limited flexibility (currently, only multidimensional tables are supported, but no recursive, tree-like structures).

One further approach into the direction of using XML for statistical data exchange is "triple-s XML", a standard for interchanging survey data (see, e.g., Hughes et al., 1999; Jenkins, 1996; Wills, 1992, and `http://www.triple-s.org/`). This XML standard describes a meta-data format, giving additional meaning to separately distributed data stored in row/column format, basically including the names and column ranges of the different variables, but also specifying variable types and value ranges. But because this format relies on row/column-stored data only, the maximal data complexity, again, is limited (e.g., no recursive structures, no higher-dimensional arrays).

2 Requirements on Statistical Data

Statisticians need a data format that is both flexible enough to handle all different kinds of statistical data (from time series to micro-array data), and specialized enough to incorporate statistical notions such as scales and factors. Such a data format should feature:

- Special symbols for infinities and undefined values,

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

[3] `http://www.epros.ed.ac.uk/metanet/`

[4] such as the Inter-University Consortium for Political and Social Research (ICPSR)—see: `http://www.icpsr.umich.edu/`

- Special symbols for missingness ("not available"),
- Logical data,
- Categorical data (nominal/unordered, ordinal/ordered, or cyclic),
- 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), and
- Meta-data (on all hierarchical levels).

Vectors should be indexable arbitrarily—in order to build matrices or multidimensional arrays. Lists allow 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[5]/R[6] and MATLAB[7]/ Octave[8]), statistical software (SPSS[9], SAS[10], Minitab[11], XploRe[12]), and spreadsheets (Excel[13], StarCalc[14], Gnumeric[15]). In spreadsheets, MATLAB/Octave, and XploRe, categorical data can only be represented by strings. Arrays of arbitrary dimension are supported by Splus/R, MATLAB, and XploRe 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. This comparison clearly shows that all paradigms, except S based languages, have limitations. Therefore, our design was basically inspired by S, and later became more generalized.

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*), that can be formally described with a special data definition language (DTD)—see the World Wide Web Consortium (2000) recommendation. Note that quoting is needed for the special XML characters '&', '<', and '>' by using '`&amp;`', '`&lt;`', and '`&gt;`', respectively. In the following, we will go through the rules in the 'StatDataML.dtd' file (the DTD as a whole is given in the Appendix).

[5]For Splus see: `http://www.insightful.com`
[6]For R see: Ihaka & Gentleman (1996) and `http://www.R-project.org`
[7]For MATLAB see: `http://www.mathworks.com`
[8]For Octave see: `http://www.octave.org`
[9]For SPSS see: `http://www.spss.com`
[10]For SAS see: `http://www.sas.com`
[11]For Minitab see: `http://www.minitab.com`
[12]For XploRe see: `http://www.i-XploRe.de/`
[13]For Excel see: `http://www.microsoft.com`
[14]For StarCalc see: `http://www.staroffice.com` or `http://www.openoffice.com`
[15]For Gnumeric see: `http://www.gnumeric.org`

	R/Splus	MATLAB	Octave	spreadsheet	SPSS	SAS	Minitab	XploRe
$\pm\infty$, NaN	yes	yes	yes					yes
missingness	yes		yes	yes	yes	yes	yes	
logical	yes	(yes)		yes	yes	yes	yes	
nominal	yes	strings	strings	strings	coding	yes	strings	
ordinal	yes				yes	yes	yes	
integer	yes	yes				yes		
real	yes	yes	yes	yes	yes	yes	yes	yes
complex	yes	yes	yes					
character	yes	yes	yes	yes	yes	yes	yes	yes
date/time	yes	yes	yes	yes	yes	yes	yes	
matrix	yes	yes	yes	yes	yes	yes	yes	yes
arrays	yes	yes					yes	
lists	yes	yes	yes					yes
meta-data	yes			yes	yes			yes

Table 1: Data representation capabilities of different software packages. Empty cells mean 'no'. 'spreadsheet' includes EXCEL, Gnumeric and StarCalc.

3.2 The File Header

The top level '`StatDataML`' element contains one '`description`' and one '`dataset`' element, each optional:

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

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

The '`description`' element is used to provide meta-information about a dataset—typically not needed for computations on the data itself:

```
<!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)>
```

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.

3.4 The ‘dataset’ element

We define a ‘dataset’ element either as a list or as an array:

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

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 the S language, or the corresponding “arrays” and “cell-arrays” in MATLAB). The basic property of a list is its recursive structure, in contrast to arrays that 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)*>
```

The ‘dimension’ element contains one or more ‘dim’ tags, depending on the number of dimensions:

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

Each of them has ‘size’ as a required attribute, and may 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), again lists (allowing complex and even recursive structures), or ‘empty’ tags (indicating non-existing elements, corresponding to ‘NULL’ in S).

3.4.2 Arrays

Arrays are blocks of data objects of the same elementary type with dimension information used for memory allocation and data access (indexing):

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

The ‘dimension’ and ‘properties’ elements are identical to the corresponding ‘list’ tags. The ‘listdata’ block gets replaced by the ‘data’ (or ‘textdata’) element that contains the data itself. The ‘type’ element contains all information about the statistical data type:

```
<!ELEMENT type (logical | categorical | numeric | character | datetime)>

<!ELEMENT logical EMPTY>
<!ELEMENT categorical (label)+>
<!ELEMENT numeric (integer | real | complex)?>
<!ELEMENT character EMPTY>
<!ELEMENT datetime EMPTY>
```

It must contain exactly one ‘logical’, ‘categorical’, ‘numeric’, ‘character’, or ‘datetime’ tag. The ‘categorical’ tag must—and the ‘numeric’ element may—contain additional elements, providing even finer type characterization.

The ‘categorical’ tag carries a ‘mode’ attribute that can be ‘unordered’ (“factors”), ‘ordered’, or ‘cyclic’ (e.g., days of the week)—‘unordered’ is the default:

```
<!ELEMENT categorical (label)+>
<!ATTLIST categorical mode (unordered | ordered | cyclic) "unordered">
```

In addition, the factor labeling has to be specified by the means of one or more ‘label’ tags:

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

The ‘label’ element has a mandatory ‘code’ attribute specifying the levels’ integer value, and optionally contains a name. 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. As an example, consider the type specification of a color factor:

```
<type>
  <categorical mode="unordered">
    <label code="1">red</label>
    <label code="2">green</label>
    <label code="3">blue</label>
    <label code="4">yellow</label>
  </categorical>
</type>
```

In the data section (see below), only the codes will be used.

Finally, the ‘numeric’ element may contain a further tag, allowing the distinction of ‘integer’, ‘real’, and ‘complex’ data:

```
<!ELEMENT numeric (integer | real | complex)?>

<!ELEMENT integer (min?, max?)>
<!ELEMENT real (min?, max?)>
<!ELEMENT complex>
```

If 'numeric' is left empty, the data is assumed to be real. For 'integer' and 'real', one optionally can specify the data range using the 'min' and 'max' tags, allowing the parsing software both to choose a memory-saving storage mode and to check the data validity:

```
<!ENTITY % RANGE "#PCDATA | posinf | neginf">
<!ELEMENT min (%RANGE;)>
<!ELEMENT max (%RANGE;)>
```

As an example, consider the type specification for the integers from 1 to 10:

```
<type>
  <numeric>
    <integer>
      <min>1</min> <max>10</max>
    <integer/>
  <numeric/>
<type/>
```

The content of 'min' and 'max' can also be '<posinf/>' and '<neginf/>' for $+\infty$ and $-\infty$, respectively.

3.4.3 The 'data' tag

If 'data' is used (especially recommended for character data), then each element of the array representing an existing value is encapsulated in '<e>'...'</e>' pairs (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/>'):

```
<!ELEMENT data (e|ce|na)* >
<!ATTLIST data true  CDATA "1"
               false CDATA "0">

<!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;)* >

<!ATTLIST e  info CDATA #IMPLIED>
<!ATTLIST ce info CDATA #IMPLIED>
<!ATTLIST na info CDATA #IMPLIED>
```

'<na/>', '<e>', and '<ce>' tags can carry an optional 'info' attribute, allowing the storage of meta-information:

```
<e>120<e/> <e info="unsure">123<e/> <na info="data deleted">
```

As another 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>
  <e>red</e> <e>green</e> <na/> <e>blue</e> <e></e> <e>yellow<e>
</data>
```

If the colors are coded as factor levels (see the notes on categorical type specification above), the example would become:

```
<data>
  <e>1</e> <e>2</e> <na/> <e>3</e> <e></e> <e>4<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 (Institute of Electrical and Electronics Engineers, 1985), 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, e.g., as follows:

```
<data>
  <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, containing 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>
  <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 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). 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 and 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">
```

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 (such as *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">
  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-like structure, writing a *StatDataML* file in R, reading this file in MATLAB) is realized.

4.1.1 A session with R

```
R : Copyright 2002, The R Development Core Team
Version 1.6.1 (2002-11-01)

## 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>
    <type> <numeric> <integer/> <numeric/> <type/>
    <data>
      <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 character variable '`a`' (with one missing value), and one numerical variable '`b`'.

a	A	B		D	E
b	0.5	$+\infty$	4.5	NaN	1.0

```
<?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="5"/>
        </dimension>
        <type> <character/> <type/>
        <data>
          <e>A</e> <e>B</e> <na/> <e>D</e> <e>E</e>
        </data>
      </array>
      <array>
        <dimension>
          <dim size="5"/>
        </dimension>
        <type> <numeric/> <type>
        <data>
          <e>0.5</e> <e><posinf/></e> <e>4.5</e> <e><nan/></e> <e>1.0</e>
        </data>
      </array>
    </listdata>
  </list>
</dataset>

</StatDataML>
```

5 Implementation

Currently we have support for R, MATLAB and Octave, and converters for SPSS and Gnumeric are under development.

- The *StatDataML* R package provides an implementation of *StatDataML* I/O routines for R. The two functions ‘`writeSDML`’ and ‘`readSDML`’ implement writing and reading for *StatDataML* files. With this implementation, it is possible to write and read R data objects without loss of information (`http://cran.r-project.org/src/contrib/StatDataML_1.0.tar.gz`).
- The code uses Duncan Temple Lang’s XML package, providing general XML parsing for S engines (`http://cran.r-project.org/src/contrib/Omegahat/XML.tar.gz`).
- The complete Omegahat package, also including the MATLAB/Octave implementation, can be found at: `http://www.omegahat.org/StatDataML/StatDataML_1.0.tar.gz`.
- Both the XML package and the MATLAB/Octave implementation use `libxml`, the XML C library for gnome (`http://www.xmlsoft.org/`).

6 Conclusion: Limitations and Extensibility

StatDataML seems general and flexible enough to cover most of statisticians’ data representation needs. There are some limitations, though:

First, it might 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 opinion 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. A further issue—especially for large datasets—are distributed data structures: as an example, 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 continuous change, using *StatDataML* files as structured link-lists. Both should easily be possible using the standardized “XInclude” specification, that offers a general link tag for XML files.

This leads to yet unresolved topics: how may users specify possibly remote database access? At least, we would need some query-information (e.g., in SQL) supplied along with the database URL.

Finally, within ‘`StatDataML.dtd`’, we describe how a basic dataset should be organized. We currently do not provide definitions for classes such as 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. But to our knowledge, this can not be done with standard XML—restrictions necessitate the specification of a new DTD.

References

Becker, R. A. & Chambers, J. M. (1984). *S. An Interactive Environment for Data Analysis and Graphics*. Monterey: Wadsworth and Brooks/Cole.

Becker, R. A., Chambers, J. M., & Wilks, A. R. (1988). *The New S Language*. London: Chapman & Hall.

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

Chambers, J. M. & Hastie, T. J. (1992). *Statistical Models in S*. Chapman & Hall.

Grossman, W. (2000). Use of metadata in the statistical production process. *Computational Statistics*, **15**(1), 41–51.

Hughes, K., Jenkins, S., & Wright, G. (1999). *triple-s XML: A Standard Within A Standard*. ASC.

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*.

Jenkins, S. (ed.) (1996). *The triple-s Survey Interchange Standard: The Story So Far*. ASC.

Kent, J.-P. & Schuerhoff, M. (1997). Some thoughts about a metadata management system. In Ioannidis, Y. E. & Hansen, D. M. (eds.), *Ninth International Conference on Scientific and Statistical Database Management, Proceedings, August 11-13, 1997, Olympia, Washington, USA*, pp. 174–185. IEEE Computer Society.

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

Thomas, W. L. & Block, W. C. (2001). An introduction to the data documentation initiative (ddi). In *Proceedings of the ICPSR Biennial Meeting of Official Representatives*.

Wills, P. (1992). *Data Use and Reuse*. SGCSA.

World Wide Web Consortium (2000). *Extensible Markup Language (XML), 1.0 (2nd Edition)*. Recommendation 6-October-2000. Edited by Tim Bray (Textuality and Netscape), Jean Paoli (Microsoft), C. M. Sperberg-McQueen (University of Illinois at Chicago and Text Encoding Initiative), and Eve Maler (Sun Microsystems, Inc. - Second Edition). Reference: http://www.w3.org/TR/2000/REC-xml-20001006.

Appendix: the *StatDataML* `.dtd` file

```
<!-- StatDataML DTD version="1.0" -->

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

<!-- document description tags -->

<!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)>

<!-- basic elements -->

<!ELEMENT dataset (list | array)>

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

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

<!-- dimension elements -->

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

<!-- type elements -->

<!ELEMENT type (logical | categorical | numeric | character | datetime)>

<!ELEMENT logical EMPTY>

<!ELEMENT categorical (label)+>
<!ATTLIST categorical mode (unordered | ordered | cyclic) "unordered">
<!ELEMENT label (#PCDATA)>
<!ATTLIST label code CDATA #REQUIRED>

<!ELEMENT numeric (integer | real | complex)?>
<!ELEMENT integer (min?, max?)>
<!ELEMENT real (min?, max?)>
<!ELEMENT complex>
<!ENTITY % RANGE "#PCDATA | posinf | neginf">
<!ELEMENT min (%RANGE;)>
<!ELEMENT max (%RANGE;)>
```

```
<!ELEMENT character EMPTY>

<!ELEMENT datetime EMPTY>

<!-- data/textdata tags -->

<!ELEMENT data (e|ce|na)* >
<!ATTLIST data true  CDATA "1"
               false CDATA "0">

<!ELEMENT textdata (#PCDATA) >
<!ATTLIST textdata sep CDATA " \"
                   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">

<!-- e/ce/na elements -->

<!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;)*>

<!ATTLIST e  info CDATA #IMPLIED>
<!ATTLIST ce info CDATA #IMPLIED>
<!ATTLIST na info CDATA #IMPLIED>
```