StatDataML: An XML Format for Statistical Data

David Meyer[1], Friedrich Leisch[1,4], 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

[4] Institut für Statistik und Decision Support Systems, Universität Wien, Universitätsstraße 5, A-1010 Wien, Austria

Summary

In order to circumvent common difficulties in exchanging statistical data between heterogeneous applications (format incompatibilities, technocentric data representation), we introduce a new data exchange format for statistical data called *StatDataML*. Because it seems natural for complex data structures to separate the raw data from its logical structure, we base *StatDataML* on XML, the Extensible Markup Language. In addition, our design 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. Besides a detailed presentation of the language elements, the paper includes a comparison with other data concepts as well as some examples illustrating the *StatDataML* format and its practical use.

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 obviously could significantly be 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, called *StatDataML*. It is based on XML, the Extensible Markup Language (World Wide Web Consortium, 2000), a physical data format allowing to enrich data with structure and meta-information. In addition, 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 (see Temple Lang & Gentleman, 2001, for a more specific approach representing S objects in XML). Additionally, each object can have an attached 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 (see, e.g., http://www.gla.ac.uk/External/RSS/RSScomp/metamtg.html). 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 Grossmann, 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 (http://www.epros.ed.ac.uk/metanet/), 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 (such as the Inter-University Consortium for Political and Social Research (ICPSR)—see: http://www.icpsr.umich.edu/), and consists of five 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-by-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-by-column-stored data only, the maximal data complexity, again, is limited (e.g., no recursive structures, no higher-dimensional arrays).

The reminder of the paper is organized as follows: after discussing the requirements on statistical data and comparing some of the most popular paradigms, we detail our data format by describing all language elements. The presentation is complemented by a demo session illustrating the use of two implementations, and sample outputs for two selected artificial data sets. The paper closes with implementation issues and the discussion of limitations and the extensibility of *StatDataML*.

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 missingness. If allmost all popular statistical data concepts are part of the specification, the resulting format should be rich enough to allow save & restore operations without loss of information. E.g., both S-PLUS and MATLAB data objects can be saved in StatDataML format and identically restored to S-PLUS *or* MATLAB, respectively. Hence, the format need not be the intersection of all statistical software packages involved, but a superset of what current software products offer, including:

- Special symbols for infinities and undefined values,
- Special symbols for missingness (“not available”),
- Logical data,
- Categorical data (nominal/unordered, ordinal/ordered, or cyclic),
- Numeric data (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 as rectangular forms—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. First, we look at two families of mathematical programming languages: S-PLUS (Insightful Corp., 2003) and R (R Development Core Team, 2003; Ihaka & Gentleman, 1996) as representatives of the S language on the one hand, and the family of MATLAB (The Mathworks, Inc., 2003) and Octave (Eaton, 2003) on the

	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				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.

other. Then, we describe some statistical software: SPSS (SPSS, Inc., 2003), SAS (SAS Institute, Inc., 2003), Minitab (Minitab, Inc., 2003), and XploRe (MD*Tech, 2003). Finally, we also include three spreadsheets: Excel (Microsoft Corp., 2003), StarCalc (Sun Microsystems, Inc., 2003), and Gnumeric (GNOME Foundation, 2003). In spreadsheets, MATLAB/Octave, and XploRe, categorical data can only be represented by strings. Arrays of arbitrary dimension are supported by S-PLUS/R, MATLAB, and XploRe only. Complex numbers are only supported by S-PLUS/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 S languages seem to have the most general design, and we therefore based our work on this paradigm.

3 *StatDataML*

3.1 *StatDataML* is XML

For "statistical data", one would usually think of such things as tabular data (the typical observation times variable format, contingency tables, ...), time series objects, or responses and regressors. 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 (conventionally with extension '`.sdml`') containing several XML elements (so called *tags*), that can formally be 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)—DTD-Elements are set in `typewriter` font. *StatDataML* examples—in addition—have a frame.

3.2 The File Header

The top level ‘StatDataML’ element contains one ‘description’ and one ‘dataset’ element, each optional:

```
<!ELEMENT StatDataML (description?, dataset?)>
<!ATTLIST StatDataML xmlns CDATA
                     #FIXED "http://www.omegahat.org/StatDataML/">
```

Note that all ‘StatDataML’ elements per default live in the ‘StatDataML’ namespace defined by the URI “http://www.omegahat.org/StatDataML/”.

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). Most of these elements can be defined in analogy to the core elements of the Dublin Core Meta-data Initiative specifications (DCMI—see, e.g., Hillmann, 2001): ‘title’, ‘source’, and ‘comment’ are defined in the ‘content’ section (‘comment’ has similar semantic meaning than the ‘description’ element from the DCMI). ‘date’ can be found in the ‘instantiation’ section, but whereas the DCMI only recommends it, we *require* the use of ISO 8601 format (see Section 3.4.3). The ‘creator’ element (from the ‘Intellectual Property’ section) should contain knowledge about the creating application and the *StatDataML* implementation.

In addition to these DCMI elements, we define: ‘version’ (complementing ‘date’ in uniquely identifying the data set), ‘properties’ (offering a well-defined structure to save application-based meta-information), and, finally, ‘class’, basically provided for future extensions, when formal application-independent definitions for complex classes (such as time series or genotype data) will be available.

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 virtually 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 generic structure (it may contain data of any type), in contrast to arrays whose elements are all of the same type. As a consequence, lists can also represent recursive structures because they can also contain lists.

3.4.1 Lists

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

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

The '`dimension`' element may contain several '`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`' should 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|T|F)* >

<!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 na EMPTY>

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

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

If the colors were coded as factor levels, the example would become:

```
<data>
  <e>1</e> <e>2</e> <na/> <e>3</e> <e></e> <e>4<e>
</data>
```

(Note that the association between numbers and labels is defined in the '`type`' section as mentioned above.)
'`<e>`', and '`<ce>`', and '`<na/>`' tags (and also '`<T/>`' and '`<F/>`', see above) can carry an optional '`info`' attribute, allowing the storage of meta-information:

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

IEEE Number Format

Computer systems represent numbers in different ways, depending on their hardware architecture. We require the number format 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 was not implemented (we are not distinguishing between the 'Quiet' and 'Signalling' variants of `NaN` provided by the IEEE standard, the 'signalling' one—if ever used—being more close to `NA` values for which we define a special symbol). 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>
```

When an application reads a *StatDataML* file, the implementation is responsible for the correct casts, i.e. for choosing the appropriate number representation in the computer system.

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' values are represented by the special tags '`<T/>`' and '`<F/>`':

```
<data>
  <T/> <F/>
</data>
```

Date and Time Information

Data of type ‘`datetime`’ has to follow the ISO 8601 specification (see International Organization for Standardization, 2000). *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. The ‘`true`’ and ‘`false`’ attributes can be used to change the default representation of logical values (‘`1`’ and ‘`0`’).

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 the following:

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

In this example, we use the default separator set (`" \n"`), but alternatively, we could have defined a different set of symbols with the ‘`sep`’-attribute, such as colons or semi-colons.

3.5 Implementation issues

Interfaces implementing *StatDataML* should provide options for setting conversion strings for the ‘NA’, ±∞ 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 Session: Exchanging data between R and MATLAB

This example shows how to bring the ‘iris’ data set to MATLAB and the ‘durer’ data to R.

We start in R . . .

```
R : Copyright 2003, The R Development Core Team
Version 1.6.2  (2003-01-10)

## load the StatDataML package
> library(StatDataML)
Loading required package: XML

## load the iris data
> data(iris)

## show the first observation
> iris[1,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa

## write StatDataML file
> writeSDML(iris, file = "iris.sdml")
```

. . . continue in 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(′iris.sdml′, ′list.as.structarray′)

>> x(1)

ans =

    Sepal.Length: 5.1000
     Sepal.Width: 3.5000
    Petal.Length: 1.4000
     Petal.Width: 0.2000
         Species: ′setosa′

>> load durer
>> whos X, c = cellstr(caption)

  Name      Size         Bytes  Class
  X       648x509      2638656  double array

c =

′Albrecht Durer’s Melancolia.′
′Can you find the matrix?′

>> writesdml(X, ′durer.sdml′, ′TITLE′, c{1}, ′TEXTDATA′, ′TRUE′)
```

. . . and return to R again:

```
> durer <- readSDML("durer.sdml", read.description = TRUE)
> attr(durer, "SDMLdescription")$title

[1] "Albrecht Durer’s Melancolia."
```

4.2 Sample output: The integers from 1 to 10

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

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

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

5 Implementation

Currently we have support for R, MATLAB and Octave, and converters for SPSS and Gnumeric are under development. The complete package including all implementations can be found at the homepage of the "Omegahat" project, which aims at providing a variety of open-source software for statistical applications (focusing on web-based software, Java, the Java virtual machine, and distributed computing): http://www.omegahat.org/StatDataML/StatDataML_1.0.tar.gz.

The software for the R system alone is also provided as an R package from the software archive of the R project for statistical conmputing (http://cran.r-project.org/src/contrib/StatDataML_1.0.tar.gz). It 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.

All implementations make use of XML functionality provided by libxml, the XML C library for gnome (http://www.xmlsoft.org/). The R implementation of *StatDataML* in addition requires Duncan Temple Lang's XML package, providing general XML parsing for S engines (http://cran.r-project.org/src/contrib/Omegahat/XML.tar.gz).

Finally, we compare our proof-of-concept implementations to two popular data conversion tools: "DBMS/Copy" (DataFlux, Inc., 2003) and "Stat/Transfer" (Circle Systems, Inc., 2003). Both are software packages for data conversion transforming native source into native destination files using internal transition formats. These internal formats are not documented, but from the feature lists of both software packages, it seems clear that they were designed to cover a feature subset common to most existing data formats, and thus are necessarily limited. For example, both packages only support tabular data (no multi-way tables or recursive structures), and categorical variables are not explicitly supported (Stat/Transfer, however, has limited support for value labels of SAS data). In addition, both software products are commercial and not meant to be user extensible. In contrast, our approach consists in defining a format as general as possible, thus covering a superset of currently available data formats, not influenced by limitations of particular software packages. Such "bottlenecks" should be handled at the implementation level, not at the design level. All software packages supporting *StatDataML* contribute to a virtual data pool easily accessible by new software as soon as our open standard is implemented.

6 Conclusion

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 is easily possible using the standardized "XInclude" specification, that offers a general link tag for XML files (not supported by all XML parsers, however). Note that the URL targeted by such a link is not restricted to a *StatDataML* file: it could also point to a script (e.g., CGI), retrieving data from a data base and transforming it to valid *StatDataML*. But still, we may need to specify even more flexible data requests, possibly by directly supplying the query-information (e.g., in SQL), along with the database URL.

Finally, within '`StatDataML.dtd`', we only describe how a basic dataset should be organized. We currently do not provide DTD-based definitions for classes such as a `S`-dataframe or time series. To model this, we would like to have a principle of inheritance from '`dataset`' such that the basic DTD could be extended or restricted and an XML parser could 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.

Acknowledgements

This research was supported by the Austrian Science Foundation (FWF) under grant SFB#010 ('Adaptive Information Systems and Modeling in Economics and Management Science').
We are particularly grateful to the reviewers for many helpful comments and suggestions.

References

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.

Circle Systems, Inc. (2003). Stat/Transfer version 7. Seattle, WA: DataFlux, Inc., http://www.dataflux.com/.

DataFlux, Inc. (2003). dfPower DBMS/Copy: Release 8.0. Cary, NC: DataFlux, Inc., http://www.dataflux.com/.

Eaton, J. W. (2003). Octave software version 2.0.17, http://www.octave.org/.

GNOME Foundation (2003). Gnumeric software version 1.0.11, http://www.gnumeric.org/.

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

Hillmann, D. (2001). Using dublin core. DCMI Recommendation. http://dublincore.org/documents/2001/04/12/usageguide/.

Hughes, K., Jenkins, S., & Wright, G. (1999). triple-s XML: A standard within a standard. In *Proceedings of the ASC International Conference*. Association for Statistical Computing (ASC).

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

Insightful Corp. (2003). S-PLUS statistical software: Release 6.0. Seattle, WA: MathSoft, http://www.insightful.com/.

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

International Organization for Standardization (2000). *ISO 8601:2000, Data elements and interchange formats - Information Interchange - Representation of dates and times*.

Jenkins, S. (1996). The triple-s survey interchange standard: The story so far. In *Proceedings of the ASC International Conference*, pp. 341–350. Association for Statistical Computing (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.

MD*Tech (2003). XploRe statistical software version 4.4. Berlin, Germany: MD*Tech – Method and Data Technologies, http://www.i-XploRe.de/.

Microsoft Corp. (2003). Excel 2000. Redmond, WA: Microsoft Corp., http://www.microsoft.com/.

Minitab, Inc. (2003). Minitab software: Release 13. State College, PA: Minitab, Inc., http://www.minitab.com/.

SAS Institute, Inc. (2003). SAS® software version 9. Cary, NC, USA: SAS Institute Inc., http://www.sas.com/.

R Development Core Team (2003). R software version 1.6.2, http://www.R-project.org/.

SPSS, Inc. (2003). SPSS statistical software version 11.5. Chicago, Illinois: SPSS Inc., http://www.spss.com/.

Sun Microsystems, Inc. (2003). StarOffice 6.0. Santa Clara, CA: Sun Microsystems, Inc., http://www.sun.com/.

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

The Mathworks, Inc. (2003). MATLAB software: Release 13. Natick, MA: The Mathworks, Inc., http://www.mathworks.com/.

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: the movement and use of survey data across different hardware and software platforms. In *Proceedings of the ASC International Conference*, pp. 297–304. Association for Statistical Computing (ASC).

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.