XML4MAT: Inter-conversion between Matlab™ structured variables and the markup language MbML

Jonas S Almeida*, Shuyuan Wu, Eberhard O. Voit
Department of Biometry & Epidemiology, Medical University of South Carolina
135 Cannon Street, Suite 303, P.O. Box 250835, Charleston, SC 29425, USA

* corresponding author, almeidaj@musc.edu

Abstract

The Matlab™ programming environment and related public license environments such as Octave are gaining in popularity for the identification of algorithms and the rapid prototyping of applications in bioinformatics. At the same time, there is a strong push to standardize the identification of extended modelling languages (XML) and their underlying ontologies, to facilitate bioinformatic integration of data and methods. We hereby introduce a new *m*-file library, XML4MAT, that supports the inter-conversion between any Matlab™ structured variable and a specialized extended markup language (XML), designated as MbML. The library developed also includes functions to import non-MbML compliant XML structures. The functionality described is achieved without object-oriented programming, which makes it ideal for inclusion in declarative programming and implicitly turns *m-structures* into general-purpose object models for data structures. The new library is made freely available at *http://bioinformatics.musc.edu/xml4mat*. It is ideally suited for 1) computation of XML structures in Matlab programming environments and 2) its inter-conversion to and from a specialized markup language, MbML. This also enables using Matlab structures as a format to identify new markup languages that are MbML compliant, with the corresponding gain in clarity and computability for bioinformatic applications in that environment.

Keywords
XML, Matlab, data structures, ontology, XML4MAT, MbML

Introduction

The unfolding of the "post-sequence era" (Kanehisa 2003) has been accompanied by increasingly diverse data formats and methods. Correspondingly, the initial focus on algorithm development for sequence analysis is expanding to establish a more general computational basis for systems biology (Kitano, 2002). The pursuit of this goal is challenged not only by the complexity of interactions in biological systems but also by the diverse nature of the data that need to be integrated in systems modelling. Consequently, comprehensive programming environments that are suited for calculus and matrix algebra, string processing, as well as for computationally intensive numerical and symbolic methods for differential calculus, are becoming a critical necessity. Moreover, issues of general access and relevance for life scientists mandate that such environments must be simple enough to permit fast learning and training.

Balancing computational power with simplicity and versatility in programming environments has been a long-sought goal within the engineering community. As a result, programming environments were developed with the intent of creating such an

industry standard. The premier example is arguably Matlab™ (Mathworks Inc., http://www.mathworks.com), but other options exist, such as the similarly conceived open source GNU public license Octave (http://www.octave.org) and the somewhat analogous Scilab (http://www.scilab.org). The popularity of these environments is now beginning to expand from the engineering community into the bioinformatics community and others that face similar computational challenges of multidisciplinary integration. The convenient combination of simplicity and power in these programming environments is based on the transparent use of intuitive languages that are dynamically linked to C, C++, JAVA, FORTRAN and other languages endowed with efficient compilers. The code written in the Matlab™ language is often referred to as *m-code* or *m-functions*, because of the conventional *.m* suffix in filenames originally proposed by Mathworks Inc. By extension, the structured variables defined in this environment will be designated as *m-variables* in this report. Over the years, numerous public domain and a fair amount of commercial *m-function* libraries ("toolboxes") have been created for almost every conceivable area of scientific computation, from theoretical biology to statistical mechanics, some of them making generous use of the recently incorporated support of *regular expressions* (Fried, 2002).

The emerging urge to integrate diverse computing techniques in bioinformatics applications has been accompanied by the parallel necessity of including very diverse types of data. The challenge presented by these differing data types is to capture and store the data themselves simultaneously with their experimental and methodological context ("metadata"), with the goal of creating truly new *information*. Since the early days of bioinformatics, when GeneBank was configured by formatting DNA sequence data using ANS.1 standards (Benson *et al*., 2002), a consensus has gradually been emerging to standardize data storage and handling using extended markup languages (XML). Today, XML is *de facto* the standard for intercommunication between applications (Ambrosio, 2003), as well as for the communication of biological data (Achard *et al*., 2001; Freier *et al*., 2002; Hanisch *et al*., 2002; Juty *et al*., 2001; Kitano, 2002; Lacroix, 2002; Martin, 2001; Matsuno *et al*., 2001). It is even argued by some (Barillot and Achard, 2000) that XML will eventually reach the status of *lingua franca* for computation in both biology and other areas of science.

The convenience of Matlab™ for computation and that of extended markup languages (XML) for representation calls for tools of inter-conversion between XML data structures and Matlab's structured variables. Four specific applications in particular would benefit from a library supporting the inter-conversion:

a) Data warehousing of *m*-variables. The *m*-variable full text equivalent, namely an alphanumeric XML string (MbML), could be stored as individual entries in text fields of relational databases. Consequently, in addition to searching the data structure explicitly, this would also enable faster plain-text queries using regular expressions and other text processing tools that will recognize structures with specific components.
b) Web-services. The deployment of web-services using *m*-code could be easily configured to process arbitrary submissions formatted as XML (for example using SOAP standards). This significantly shortens the typical development cycle, which consists of identifying a novel theoretical solution, developing the corresponding algorithm, and deploying it.
c) Fast identification of novel schemas for diverse biological data types. When a new experimental method is devised, the new data types often trigger secondarily the identification of a data model. Ultimately, the relevant context should be closely

reflected by the structure of the variables, which is facilitated by the flexibility of structure and dimensionality accommodated by *m*-variables. Consequently, the *m*-variable can be viewed as a clarifying template for schema formalisation, because the inter-conversion would automatically create an equivalent XML schema.

d) Bioinformatics training. Bioinformatics trainees have to focus their attention on both understanding biology and making use of computer science. The use of XML is often too technical for novices, requiring a good command of object oriented programming and object model structures. The inter-conversion proposed here sidesteps these technical issues by allowing, at least initially, the use of the more intuitive Matlab environment to identify both new algorithms and markup languages for new data structures.

The need for inter-conversion tools described above has been recognised by other researchers and software developers. The recognition of this need has driven the development of a number of tools elsewhere, in addition to Matlab's own native commands to read, write and process XML:

http://www-sop.inria.fr/epidaure/personnel/Guillaume.Flandin/xml/index.php
http://www.geodise.org/Pages/xml_toolbox.htm

The tool proposed here differs from those others by the combination of its simplicity, unrestricted public license release and by not relying on explicit object model structures (such as Document Object Model, DOM, and Java Data Object Model, JDOM), which may complicate its use in a Matlab-like declarative programming environment. Instead, the *m-variable* structure itself is used as a *de facto* object model.

Results

Formal representation of MbML structures

The XML4MAT *m*-file toolbox presented here supports direct inter-conversion between Matlab-type variables of arbitrary structure (*m-variables*) and properly formed XML alphanumeric strings. The syntax of the resulting markup language is designated as MbML. A conscious effort was made not to use object models as to make the inter-conversion transparent for declarative programming. In practice, *m*-variables are themselves used as *de facto* object models which has the advantage of containing the inter-conversion within the convenience of a Matlab-like declarative programming environment. A more conventional albeit more technically demanding solution would have been to use Java Object Model (JOM) as an intermediate between the *m*-variable and the MbML equivalent string, as implemented elsewhere (see Introduction).

An important formal issue is the validation of MbML strings. The proposed MbML syntax is too general to be described by conventional DTD or XSDL standards. However, any specific MbML generated from a structured variable can certainly be conventionally documented. In particular, it is easily possible to automatically generate the DTD and XSDL documentation using one of widely available web-based tools such as (http://www.pault.com/pault/dtdgenerator and http://www.hitsw.com/xml_utilites).

MbML Syntax

The MbML syntax closely follows the structure of *m*-variable objects. Accordingly, the tag name is the variable (root) or field (branch) name, and the attributes are the size and coordinates of the content elements. Thus, the general syntax is

<*name* class="*class value*" size="*size value*" ith="*coordinate values*">*contents*</*name*>

where *name* is the variable or field name and the three tag attributes are described in Table 1. The syntax is recursive, which means that *contents* is formatted with similar syntax and so forth until the character or numeric content is reached.

Two main properties emerge from the syntax description in Table 1. First, the use of default values for class and size anticipates the possibility of parsing any XML statement into an MbML compliant string. This can be achieved by converting all attributes in non-MbML compliant XML into new nested fields with the attributes as new fields. This leads to a universal conversion and mathematical processing of XML: ***any XML string → MbML ←→ any m-variable*** (see next section for description of supporting functions).

Second, the *ith* attribute is not required for inter-conversions, because it is derived directly from the attribute *size* vector elements. Its inclusion in syntax of MbML has the purpose of facilitating the updating of existing *m*-variables using MbML statements. Conveniently, the *ith* attribute uses linear indices to document which contents are being updated. Consequently, if specified, this attribute is a simple vector, regardless of the dimensionality (*i.e.*, the length of vector *size*) of the contents.

Structure of inter-conversion toolbox XML4MAT

The general structure of the toolbox (*m-file* library) is described in figure 1. The inter-conversion between m-variables and MbML compliant XML strings is the central feature of the toolbox, supported by the functions *xml2mat* and *mat2xml*. This conversion is achieved without resorting to any external object model, which turns Matlab variables (m-variables) into *de facto* object models.

As described in table 1, MbML compliance involves adhering to syntactic rules that make a restricted use of tag attributes. Consequently, in order to parse other XML formats into MbML, it is necessary to turn all non-MbML compliant attributes into tagged contents. This is implemented by the *mbmling* function (figure 1). The conversion of non-compliant MbML XML strings into m-variables has to deal with the general nature of XML formats, which does not impose consistent cross-reference between multiple tags occurring multiple times within the same tag. As a consequence, two options exist for importing XML into MbML compliance. If no cross-reference consistency is to be enforced, *xml2cell* will convert the XML structure into nested cell arrays. Otherwise, a conversion into dimensional structures, using *xml2struct*, is more appropriate.

A number of peripheral functions provide additional functionality, such as an approach for encoding of special characters (*spcharin*, *spcharout*) that in one step overcome incompatibilities with transfer media (e.g. http), database applications and Matlab string representation. This particular feature is worth noting because *xml2mat* will automatically decode MbML strings encoded by *spcharin*. Similarly, mat2xml will automatically encode character content in anticipation of potential conflicts with Matlab display, for example, as regards the " ' " character. Additional peripheral features are described in the documentation of the m-files, displayed in the Matlab command window by using *help*.

Illustration of inter-conversion between m-variables and MbML strings

As an illustrative example, an *m*-variable was created to describe a genetic regulation dynamic model that was originally proposed by (Hlavacek and Savageau, 1996). The system was formulated as an S-system (Savageau, 1976; Voit, 2000) and used recently as a benchmark problem for the reverse engineering of biological networks (Kikuchi *et al.*, 2003: Table 1, Fig. 1).

The *m*-variable *ode* contains all relevant information to define the corresponding system of differential equations:

```
ode =
    subject: 'genetic regulation network'
       refs: {'Bioinformatics 16(11): 1023-1037'  'Bioinformatics 19(5): 643-650'}
         dx: 'a.*prod(x.^g,2)-b.*prod(x.^h,2)'
         x0: [5x1 double]
          a: [5x1 double]
          g: [5x5 double]
          b: [5x1 double]
          h: [5x5 double]
```

This *m-variable* is included with the toolbox in the *ode.mat* file. Using mat2xml it can be converted into the equivalent MbML string:

```
>> mbml=spcharout(mat2xml(ode,'ode'))
mbml =
<ode class="struct" size="1 1"><subject class="char" size="1 26">genetic
regulation network</subject><refs class="cell" size="1 2"><cell class="char" size="1
32"> Bioinformatics 16(11): 1023-1037</cell><cell class="char" size="1 29">
Bioinformatics 19(5): 643-650</cell></refs><dx class="char" size="1 31">
a.*prod(x.^g,2)-b.*prod(x.^h,2)</dx><x0 class="double" size="5 1">0.7 0.12 0.14 0.16
0.18</x0><a class="double" size="5 1">5 10 10 8 10</a><g class="double" size="5 5">0 2
0 0 0 0 0 -1 0 0 1 0 0 2 0 0 0 0 0 0 2 -1 0 0 -1 0</g><b class="double" size="5 1">10 10
10 10 10</b><h class="double" size="5 5">2 0 0 0 0 0 2 -1 0 0 0 0 2 0 0 0 0 0 2 0 0 0
0 0 2</h></ode>
```

The resulting MbML string may, optionally, be shortened by taking advantage of the default values of the MbML attributes (Table 1), which is achieved using the function *simplify_mbml*. The shorter string was saved in the text file ode.xml, included with the toolbox, which is visualized in figure 2 using Microsoft Explorer web-browser.

By comparing this string with the full syntax, one notices the absence of *size* attributes for horizontal vectors and of *class* attributes for character (but not numeric) fields. Either version of MbML, encoded or not, will be fully converted back to the exact same original *m*-variable by using the command *xml2mat*:

```
>> xml2mat(mbml)
ans =
    subject: 'genetic regulation network'
       refs: {'Bioinformatics 16(11): 1023-1037'  'Bioinformatics 19(5): 643-650'}
         dx: 'a.*prod(x.^g,2)-b.*prod(x.^h,2)'
         x0: [5x1 double]
          a: [5x1 double]
          g: [5x5 double]
          b: [5x1 double]
          h: [5x5 double]
```

A definite demonstration that the conversion of m-variables is bi-directional, independently of the use of encoding or simplification can be appreciated by finding out that the value of the interpretation of the expression below by Matlab is 1 (unit):

```
prod((mat2xml(xml2mat(simplify_mbml(mat2xml(ode))))==mat2xml(ode))*1)
```

Discussion

The results section was centred on the documentation of the inter-conversion of MbML and m-variables (Figure 1). This is achieved in the MbML syntax (Table 1) by making a restricted use of tag attributes, which in non MbML compliant XML is often undistinguishable from non-tagged contents. That is often a source of confusion as it frequently reflects a subjective option between styles of representation rather than a difference in the underlying data model. It can be argued, perhaps more forcefully for data sets that will likely be processed in a Matlab-like environment, that a clearer distinction between attributes and content is achieved by identifying the structure of the data first as an *m-variable*. Indeed, the attributes of MbML tags are restricted to providing information about the way contents should be computed. This may be of particular relevance to bioinformatic applications given the highly structured nature of molecular biology data and the current proliferation of multiple data models for the same types of data.

Since inter-conversion between MbML and *m-variables* is bijective, it raises the possibility of other uses of MbML representations of m-variables beyond the narrow goal of representing m-variables as strings. For example, a much wider variety of symbolic processing operations can now be applied to the comparison of m-variables, by processing their MbML representations. As above, for the discussion of the advantages of distinct use of contents and attributes, this may be of particular relevance to bioinformatic applications given the highly structured nature of molecular biology data.

Conclusions

A new *m-file* library ("toolbox") is offered for the inter-conversion between Matlab-like structured variables (m-variables), and a specialized markup language, MbML, is accordingly defined. This toolbox differs from existing ones (see Introduction) by its small size, suitability for declarative programming and focus of the dimensional structures that make this programming environment so popular. It was suggested that in addition to the practical value of centring the identification of the MbML schema on the structure of m-variables, MbML compliance may have a deeper value by explicating dimensionality in the data structure through a specialised use of tag attributes.

Methods

Computation

The accompanying toolbox was developed in the Matlab™ 6.5 environment (Mathworks Inc). However, there is relatively little development effort in porting the library to public licensed environments like Octave or maybe even Scilab. This may be done in the future, after the current version goes through one more development cycle (as an open source library), in order to consider fundamental changes in algorithm design in response to user feedback. On the other hand, *m-file* programming has captured the interest of other open source developments, namely the MatPy project using Python (http://matpy.sourceforge.net/), which may offer better conditions for porting XML4MAT to GNU Public License platforms. With regard to supported operating systems, XML4MAT it requires a functional Matlab™ installation. Consequently it currently runs on supported Windows, Linux, UNIX and

Macintosh platforms (see Mathworks Inc, http://www.mathworks.com, for operating system support details).

Availability

The XML4MAT toolbox is freely available at http://bioinformatics.musc.edu/xml4mat.

Acknowledgements

This work was supported in part by the NHLBI Proteomics Initiative through contract N01-HV-28181, and a Cancer Center grant from the Department of Energy (C.E. Reed, PI).

References

Achard, F., Vaysseix, G. and Barillot, E. (2001) XML, bioinformatics and data integration. Bioinformatics, 17, 115-25.

Ambrosio, J. (2003) Toolmakers embrace XML. Application Development Trends, 10, 34-36.

Barillot, E. and Achard, F. (2000) XML: a lingua franca for science? Trends Biotechnol, 18, 331-3.

Benson, D.A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Rapp, B.A. and Wheeler, D.L. (2002) GenBank. Nucleic Acids Res, 30, 17-20.

Freier, A., Hofestadt, R., Lange, M., Scholz, U. and Stephanik, A. (2002) BioDataServer: a SQL-based service for the online integration of life science data. In Silico Biol, 2, 37-57.

Fried, J.E.F. (2002) Mastering Regular Expressions. O'Reilly.

Hanisch, D., Zimmer, R. and Lengauer, T. (2002) ProML--the protein markup language for specification of protein sequences, structures and families. In Silico Biol, 2, 313-24.

Hlavacek, W.S. and Savageau, M.A. (1996) Rules for coupled expression of regulator and effector genes in inducible circuits. J Mol Biol, 255, 121-39.

Juty, N.S., Spence, H.D., Hotz, H.R., Tang, H., Goryanin, I. and Hodgman, T.C. (2001) Simultaneous modelling of metabolic, genetic and product-interaction networks. Brief Bioinform, 2, 223-32.

Kanehisa M., Bork, P. (2003) Bioinformatics in the post-sequence era. Nature Genetics, 33, 305-10.

Kikuchi, S., Tominaga, D., Arita, M., Takahashi, K. and Tomita, M. (2003) Dynamic modeling of genetic networks using genetic algorithm and S-system. Bioinformatics, 19, 643-650.

Kitano, H. (2002) Systems biology: a brief overview. Science, 295, 1662-4.

Lacroix, Z. (2002) Biological data integration: wrapping data and tools. IEEE Trans Inf Technol Biomed, 6, 123-8.

Martin, A.C. (2001) Can we integrate bioinformatics data on the Internet? Trends Biotechnol, 19, 327-8.

Matsuno, H., Doi, A., Hirata, Y. and Miyano, S. (2001) XML documentation of biopathways and their simulations in Genomic Object Net. Genome Inform Ser Workshop Genome Inform, 12, 54-62.

Savageau, M.A. (1976) Biochemical Systems Analysis: a Study of Function and Design in Molecular Biology. Addison-Wesley, Reading, MA.

Voit, E.O. (2000) Computational Analysis of Biochemical Systems. Cambridge University Press, Cambridge, UK.

Figures

Figure 1 - General structure of XML4MAT m-file toolbox core functions

The central feature is the inter-coversion between MbML strings and m-variables, supported by the two central functions *mat2xml* and *xml2mat*. A second group of functions deals with non-MBML compliant XML structures in order to link them to the central inter-conversion functionality.

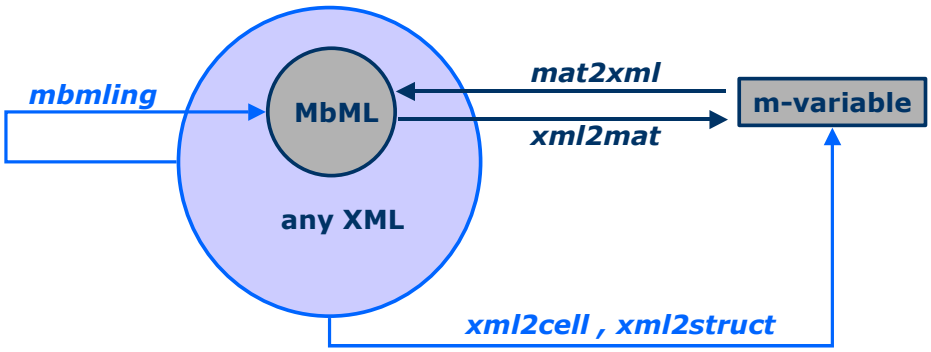

Figure 2 - Example of MbML structure

Representation of ode m-variable as an MbML string, as displayed by Microsoft Explorer version 6. This display can be reproduced simply double clicking the ode.xml file provided with the toolbox.

```
- <ode class="struct">
    <subject>genetic regulation network</subject>
  - <refs class="cell">
      <cell>Bioinformatics 16(11): 1023-1037</cell>
      <cell>Bioinformatics 19(5): 643-650</cell>
    </refs>
    <dx>a.*prod(x.^g,2)-b.*prod(x.^h,2)</dx>
    <x0 class="double" size="5 1">0.7 0.12 0.14 0.16 0.18</x0>
    <a class="double" size="5 1">5 10 10 8 10</a>
    <g class="double" size="5 5">0 2 0 0 0 0 0 -1 0 0 1 0 0 2 0 0 0 0 0 2 -1 0 0 -1 0</g>
    <b class="double" size="5 1">10 10 10 10 10</b>
    <h class="double" size="5 5">2 0 0 0 0 0 2 -1 0 0 0 0 2 0 0 0 0 0 2 0 0 0 0 0 2</h>
  </ode>
```

Tables

Table 1 - Description of attributes in MbML tags.

Each tag has the attributes described in this table. However, their values do not have to be explicit - if not specified, default values will be assumed (detailed in Description column). The existence of default values is used by *simplify_mbml* to simplify MbML representation, as described below by the illustrative example in the Results section.

Attribute	Values	Description
Class	double char struct cell	Corresponds to native classes in Matlab, with the same name containing, respectively, numbers, strings, structures and cells. **Conditions**: if this attribute is absent, its value will be assumed to be "char" unless the content is tagged, in which case it will

		be assumed to be "struct".
Size	Vector of integers	Each element of the vector sets a new dimension with that many instances. This vector is the same as the one that the Matlab native function *size()* would return for the same variable. **Conditions:** This attribute does not have to be specified if there is a single dimension, *e.g.*, if the contents are single elements or linear vectors of elements.
Ith	Vector of integers	This optional attribute is useful for a more compact definition of sparse matrices or to make small updates in large objects. It specifies which elements of the matrix are included between the tags. **Conditions:** the default value of the i-th vector is an integer progression between 1 and the product of all elements of size vector ($\Pi(size)$).

Additional files

Additional file 1 – XML4MAT toolbox

m-files, examples and public license files, compressed as a .zip file.