DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing
March 15-17, Vienna, Austria
http://www.ci.tuwien.ac.at/Conferences/DSC-2001
K. Hornik & F. Leisch (eds.) *ISSN 1609-395X*

GEE solvers: case studies in DSC design and implementation

Vincent Carey[*]

Abstract

GEEs (generalized estimating equations) provide a framework for flexibly modeling clustered data. Generalized linear model (GLM) components are used to specify marginal mean and variance functions, and "working" covariance models specify multivariate structure. The indefiniteness of the estimation and inference framework is a basis for criticism from theoretical quarters (Crowder, Bka 1995), but also a basis for interesting interface design challenges and opportunities. I will comprehensively describe the redesign of S4/R-targeted GEE solvers. Basic issues include a) choice of language, b) representation of complex clustered data structures to accommodate, e.g., responses and predictors obtained on discordant timing sequences; c) inheritance from and interoperation with existing tools for multivariate modeling; d) choice of class/method decomposition to support recognition of statistical data types; d) weak implementation methods to ease retargeting to DSC platforms as they mature. Peripheral issues include a) exploitation of XML-based literate programming methods; b) automatic generation of javadoc-like hypertext doc for S4/R classes and methods.

1 Introduction

Software development and distribution is central to effective methodological research in statistics. Of the 1500 or so citations of Liang and Zeger's generalized

[*]Channing Laboratory, Harvard Medical School, 181 Longwood Avenue, Boston MA 02115 USA; `stvjc@gauss.med.harvard.edu`. I am grateful to Robert Gentleman and Duncan Temple Lang for sharing pertinent aspects of their current work. This work was supported in part by NIH NHLBI grant HL 40619.

estimating equation (GEE) method for generalized linear modeling of clustered responses [8], the vast majority are applications using freely distributed noncommercial software (SAS IML macros [5], S/R functions from `lib.stat.cmu.edu`, CRAN [1], or XLISPSTAT [10]). Another substantial fraction consists of criticisms or elaborations based on simulation or extension of the method using the aforementioned codes.

Techniques of software development and distribution change along with the prevailing architectures of integrated statistical computing environments such as R, S, SAS. This paper is a highly personal and non-synoptic description of choices facing the developer of a widely used implementation of GEE for R/S. I will consider the choice of language, data structure protocol, development environment, and planning for future requirements in the context of redesigning GEE to exploit recent technological advances in languages and statcomp environments.

2 Brief overview of GEE

Standard generalized linear models (GLMs) for a scalar discrete or continuous outcome Y with expectation $EY = \mu$ are defined by a linear predictor $\eta = X^t_{1\times p}\beta$, a link function $g(\mu) = \eta$ and a scaled mean-variance relationship $\text{var}Y = \sigma^2 V(\mu)$. Let $D = \partial\mu/\partial\beta$. For N independent realizations y_i of Y with covariates x_i, maximum quasi-likelihood estimates of β are found by solving

$$\sum_i D_i^t V^{-1}[y_i - \mu_i(\beta)] = 0.$$

When the y_i are n_i-vectors of dependent observations such that (suitably defined) residual vectors have "working" covariance $W_i(\alpha_{q\times 1})$, the GEE for fixed α has the form

$$U_G(\beta) = \sum_i D_i^t W_i(\alpha)^{-1}[y_i - \mu_i(\beta)] = 0.$$

Liang and Zeger obtain the asymptotic distribution of solutions to $U_G(\beta) = 0$ when a $\sqrt{N}$-consistent estimator of α is "plugged in", but Crowder [3] presents a case in which misspecification of W (the parametric structure for the residual covariance) renders $U_G(\beta) = 0$ unsolvable. Various recent papers concern efficiency/feasibility gains or losses associated with choice of form of W and choice of estimator for α. Recent research concerning analysis of incomplete data introduces an additional response-specific weight that may need to be estimated. My aim is to provide software that allows users access to the best available choices of these components of the model. Sometimes the best available choice is one the user has discovered personally. The software should allow the user to employ that choice just as personally.

3 Choices facing the developer

GEE solvers can obviously inherit familiar user interface design features from more basic GLM fitting procedures. Protocols for specifying the regression model, link and variance functions can generally be inherited directly. Internals can generally not be inherited owing to the diagonal weight in the standard GLM; GEE involves a block diagonal weight. Interfaces permitting flexible specification of W and estimation of α are highly desirable, but no dominant idiom has emerged.

3.1 Language options

It is often desirable to use more than one language to cover different aspects of the interface/analysis programming problem for medium sized statistical analysis procedures. I have considered:

- **S and C**. The `gee()` routines available from statlib have endured for over a decade with one major revision to accommodate S formulas. Performance is acceptable but hand-coded memory management of matrix computations has been leaky. No emphasis was given in the design to the support of user extensibility.

- **Pure S**. The first release of YAGS (Yet Another GEE Solver) is still viable and offers complete flexibility in the specification of working covariance models and estimation procedures. It is too slow to be attractive but is a good prototype/testing platform for extensions.

- **S and C++**. The latest version of YAGS includes a C++ matrix library initially written by me in 1986. Performance is good but portability, safety, and extensibility concerns are considerable. The prospect of exploiting S4 classes and their extension to R is a motivation to major revision.

- **R and Java**. YAGS with C++ has not been ported to R yet owing to S4 class unavailability. Freely distributed numerical libraries for Java and the newly released R/S/Java interface make feasible a full redesign of GEE using Java. This is of interest from several perspectives, including greater portability, rigorous object-orientation, and potential reuse in applets.

Yet to be carefully considered are the new OOP classes for R/S or the massive Java resources of Omegahat. In summary, a developer has a wide arrange of plausible and interesting options for flexible re-implementation of medium-sized procedures, but a systematic reckoning of costs and benefits of various choices is unavailable. Details of a few of the aforementioned options will be reviewed in later sections.

3.2 Data structure protocols

A basic question posed by all users considering adoption of a new procedure is: "What format needs to be imposed on my data for the procedure to be applicable?" While it is straightforward for YAGS to accommodate standard rectangular R/S data frames (with one or more columns providing information on clustering/dependencies among responses), there are compelling reasons to consider the possibility of dealing with more irregular data structures. A key example is provided by J. K. Lindsey [9], who notes the importance of dealing with responses and covariates obtained on disjoint timing sequences. Efficient representation of such data can be accomplished with XML markup, which can subsequently be converted to extended data frame structures through work of D. T. Lang and R. Gentleman. Below I will discuss an XML protocol for encoding dependent data with no restrictions on commonality of timing sequences. The redesign of GEE solvers so that they can operate directly on more faithful representations of complex observations is a high priority, but depends upon resolution of the representation protocol and XML transformation framework.

3.3 Documentation protocols for evolving procedures

Three documentation processes need focused attention in distributed statcomp procedure development.

- **Literate programming [6, 11] for the development core.** Integrated code and detailed documentation/testing scripts/ test results are highly valuable in procedure development and distribution.
- **API doc/protocols supporting remote enhancements and reuse.** The `javadoc` system for automatic hypertext documentation of module families is an attractive model and can be emulated for R/S classes, as will be illustrated below.
- **Substantive/tutorial doc for end users**. This typically has two components: concentrated, technical "on-line" pages to be accessed at time of invocation, and "off-line" texts that give narrative support. Synchronization of such documents with evolving implementations is challenging. It is particularly important that key architectural features (e.g., argument lists) of current implementations not be misrepresented in end-user documentation.

Sections 4.1.1 and 4.1.2 below consider new approaches to literate programming and class-based API doc generation.

3.4 Planning for future requirements

While it is impossible to anticipate in detail new requirements that will emerge from the advance of technique and performance/integrity expectations among statcomp users and developers, it is worth considering how novel, unfamiliar features of improved statcomp software could be made more familiar and attractive to users if they were incorporated into a widely used procedure like GEE in advance of general demand. Two directions worthy of exploration are:

- **Statistical data types**. Methodology for analyzing incomplete, censored, truncated or error-prone data has grown and earned much respect within and outside of statistics. Data should routinely be marked to indicate that they are e.g., measured with error or subject to detection limits, and procedures should be sensitive to such marks, invoking an appropriate variation if available, or refusing to process if none exists.

- **Security enhancements**. The use of digital signatures on distributed procedures should be helpful in various respects, affording assurance that an invoked procedure is in fact the one distributed by its developer.

4 Some details of current approaches to GEE for distributed statcomp

I will now discuss details of GEE solver redesign in light of the topics sketched above. The review will travel from documentation, to data structure protocols, and finally to issues of program design and performance.

4.1 Aspects of documentation

4.1.1 Literate programming tools

A convenient approach to XML-based literate programming has been facilitated by work of Duncan Temple Lang [7]. It is assumed that a single XML file will be the basis for a variety of source and documentation modules. Call the base file `geeweb.xml`, in anticipation of *weaving* to documentation and *tangling* to source code. Literate programming resources include

- An XML tag set for marking up the web file. The set is currently defined by `article.dtd` and `article.xsl` in the Omegahat distribution.
- XSL programs for weaving (`article.xsl`) and tangling (`codeOnly.xsl`), using XSLT processors.

At present, the weave target format is HTML. Tangling does not involve prettyprinting. Figure 1 provides a simple example. `noweb` users will observe immediately the added verbosity of the XML approach. Advantages of the XML approach include use of a standardized tool set (XML/XSL as opposed to Icon/awk/$\LaTeX$) and sharp distinction between content and structural information facilitating easy retargeting. At the bottom of the figure, example translation commands are provided.

4.1.2 Javadoc-like hypertext generation

The `genDoc` utility, currently in a highly experimental condition, is intended to provide hyperlinked documentation of S4 classes and methods with high fidelity to current code images. This is in contrast to prompt-based documentation which may mislead on crucial architectural information like arguments or signatures if code has changed without documentation updates. When invoked on a given S4 class, `genDoc` examines all its slots and methods in real time and produces HTML as illustrated in Figure 2. Optional supplementary arguments to `genDoc` provide additional textual details to be added to the HTML; the protocol for merging text and `genDoc` crawl results is such that disparities between supplementary doc and true class structure are resolved in favor of the true class structure. Because the GEE redesign in terms of S4 classes has been postponed to allow simultaneous targeting of Splus and R, no appreciable examples of genDoc application to GEE are currently available. Figure 2 is an application to the polynomial class of Venables and Ripley [12].

4.2 Data structure protocols for longitudinal and clustered data

Figure 3 is a document type definition (DTD) for data obtained in coordinate systems. The DTD allows association of series of different lengths, obtained on different coordinate sequences, on the bearer of a unique key. The DTD requires that units be provided for both coordinate information and for data values. An example of marked-up data is provided at the bottom of the Figure.

Omegahat provides tools for parsing XML and for flexibly manipulating content in R/S. A small Splus program called dumpVals converts XML conforming to the Coordz DTD into lists of coordinate-value series nested within individual-level data records. The passage from lists of this type to data frames supporting modeling remains to be specified.

5 Language issues

5.1 The patterned matrix class in pure S YAGS

When prototyping a mathematical algorithm it is desirable to use a compact language that closely emulates the standard mathematical notation expressing the algorithm. In pure S YAGS, solve(sum(t(D.i) %*% Rinv.i %*% D.i)) faithfully implements $(\sum_i D_i^t R_i^{-1} D_i)^{-1}$ for a general series of cluster sizes n_i. (Relative to the definitions given in section 2, D has been slightly redefined to absorb the variance function and expose the weight as an inverse correlation matrix.) Here solve, sum and %*% have been overloaded to deal appropriately with block structure and block diagonal matrices which are represented as lists. The construction of lists representing patterned matrices is carried out flexibly using fill:

```
Rinv.i <- fill(wcorigen, cor.els, c("pmat", "block", "diag"))
```

where cor.els is a list of N elements, the ith providing information required to evaluate R_i^{-1} using the function wcorigen. For example, if R is an exchangeable correlation structure, then cor.els need only include the current value of α and the cluster sizes n_i, wcorigen computes two rational functions of α and n_i and inserts the values appropriately into the output matrix. If R is continuous time AR(1), cor.els must include actual observation times for all elements, and in general wcorigen will involve an explicit matrix inversion. The final argument to fill is an S3 class tag identifying the matrix pattern to be returned.

5.2 A speculative design in Java

Here I will not focus on GEE but will instead consider larger-scale architecture for statistical modeling using Java. The programming described here was carried out without knowledge of the existence of Omegahat, and follows some of the simpler proposals of Felleisen and Friedman [4].

The key concept underlying this approach is the visitor design pattern, also known as the double dispatch protocol. Statistical procedures are viewed as visitors to variables who seek their services. Variables are typed according to properties that determine what sort of modeling will be suitable. The base abstract class is `VblD`, extended by `CenVbl`, for example. `CenVbl` has two fields, `datahi` and `datalo` representing the upper and lower limits within which each data point is known to lie. The `Vbl` classes have a single method, `accept`, with a parameter naming the procedure class to be invoked.

Statistical procedures implement the `OpOnVblI` interface, which stipulates that any procedure must cater specifically for *each* of the types extending `VblD`. A procedure that fails to cater for censored data, for example, cannot implement the `OpOnVblI` interface. The procedure that is actually selected depends on the type of the response variable and also potentially on features of arguments to the procedure. For example, the actual procedure carried out when `RegrOnV` is invoked may depend on the class of the regressors.

This design imposes an asymmetry between statistical data representations and analysis procedures. Data classes do not require any modification when new procedures are introduced, as they possess but one generic method, `accept`. Procedures on the other hand are required to be explicitly prepared to deal with data from any of the recognized statistical data types. This asymmetry seems appropriate. Data representations should focus on efficiently encoding all substantively relevant features of the conditions of measurement. Statistical procedures should be sensitive to conditions of measurement of the data to which they are to be applied.

R with SJava-0.62-2 [7] is a convenient platform for experimenting with this Java design. With `ExtdMatrix`, a class lightly extended from `Jama.Matrix`, proceed as follows. (The `VblD`, `ExtdMatrix` and allied classes are available from the author upon request.)

```
# currently reading data from disk!
X <- .Java(.JNew("ExtdMatrix"), "LoadFrom", "X")
Y <- .Java(.JNew("ExtdMatrix"), "LoadFrom", "Y")
CY <- .JNew("CmpltVbl",Y)
RegOnX <- .JNew("RegrOnV", X)
Fit <- .Java(CY, "accept", RegOnX)
gcoef(Fit) # gets est. coefficient vector
```

Table 1 gives some run time comparisons for ordinary linear regression carried out in R `lsfit` and in the design described here. Numerical compatibility of results checked out. As the problem size grows, the Java implementation engenders a slowdown of a factor of 5 to 10. For a completely naive implementation of a speculative protocol, this may not be so bad. It is of interest to compare R to R/Java in more primitive calculations. For $N = 100$, $p = 20$ $X^t X$ and its inverse was obtained using R/Jama matrices and in pure R. R alone is about 20 times faster for the cross-product, and about 10 times faster for the inverse cross-product. Profiling the Java computations should be helpful at choosing an appropriate design of this type.

5.3 S/C++

The current distribution of YAGS [2] (some extensions described here not yet released) involves a C++ engine and will ultimately replace the statlib S/C implementation. The primary design challenge is achieving good performance without sacrificing ease of remote extensibility. Two examples of the kinds of extensibility I wish to support are

- auxiliary estimating equations for α, and
- correlation estimation based upon martingale residuals for analysis of censored outcomes.

The latter problem appears to require a switch-selectable, built-in option. The former problem is difficult to solve in a truly user-friendly way.

The current approach to incorporating and solving auxiliary estimating equations in YAGS involves a protocol for C++ functions based on `matrix` objects defined in a separate library. The function arguments are

```
matrix PRin, matrix ID, matrix TIMin, double phi,
    int p, matrix alpin
```

encoding the overall vector of Pearson residuals, the cluster discriminator, coordinate information, the scale parameter, the regression dimension, and a current value α_c of α (possibly obtained through earlier iterations). The function returns a 3×1 `matrix` object consisting of the value of the estimating function $u(\alpha_c) = \sum u_i(\alpha_c)$, its derivative u', and $\sum u_i u_i^t$. The command `matrix* e = split(PRin, ID);` yields an array of C++ matrices encoding cluster-specific residual vectors. Additional numerical facilities are provided in the associated C++ library for use in the evaluation of user-supplied $u(\alpha)$. For example, the operator $\partial/\partial\alpha \mathrm{chol} R^{-1}$ is of use in constructing unbiased estimating equations for various structures. The required derivative is analytically trivial for the AR(1) structure; for other structures the derivative can be obtained numerically using the library. Compliant functions are solved generically during the GEE iteration process, using e.g., a secant algorithm for scalar α.

6 Summary

Distributed statistical software involves both the dissemination of closed algorithms for use "as is" and the creation of toolkits for extension and recombination by other users, sometimes in environments that are unfamiliar or unforeseen. I have reviewed options facing developers of documentation, data structure, and statistical inference toolkits in the context of GEE, a popular methodology whose limits and potentials are current research questions. Navigation of the option space is a challenging but worthwhile task for statistical software developers. Developers, researchers and users can all expect to benefit when good design choices are made.

References

[1] Vincent Carey. gee, `http://lib.stat.cmu.edu /s/gee`, `http://cran.r-project.org/src/contrib/packages.html#gee`. *WWW*, 1990.

[2] Vincent Carey. Yagsv2, `http://lib.stat.cmu.edu` `http://www.biostat.harvard.edu/˜ carey/ydown.html`. *WWW*, 1999.

[3] Martin Crowder. On the use of a working correlation matrix in using generalised linear models for repeated measures. *Biometrika*, 82:407–410, 1995.

[4] Matthias Felleisen and Daniel Friedman. *A Little Java, A Few Patterns*. MIT, 1998.

[5] Ulrike Groemping. Gee iml macro 2.03 `http://ifr69.vjf.inserm.fr /ũ472/equipes/biostatistique/sdilgee5.txt`. *WWW*, 1994.

[6] Donald E. Knuth. *Literate Programming*. CSLI Stanford, 1992.

[7] Duncan Temple Lang. The omega project: new possibilities for statistical computing. *JCGS*, 9:423–451, 2000.

[8] Kung-Yee Liang and Scott L. Zeger. Longitudinal data analysis using generalized linear models. *Biometrika*, 73:13–22, 1986.

[9] J.K. Lindsey. Data objects and model formulae for the future: some proposals. `http://alpha.luc.ac.be˜ lucp0753/msmodel.ps`, 1999.

[10] Thomas Lumley. Xlispstat tools for building gee models. *Journal of Statistical Software*, 1, 1996.

[11] Norman Ramsey. Literate programming: weaving a language-independent web. *CACM*, 32:1051–1055, 1989.

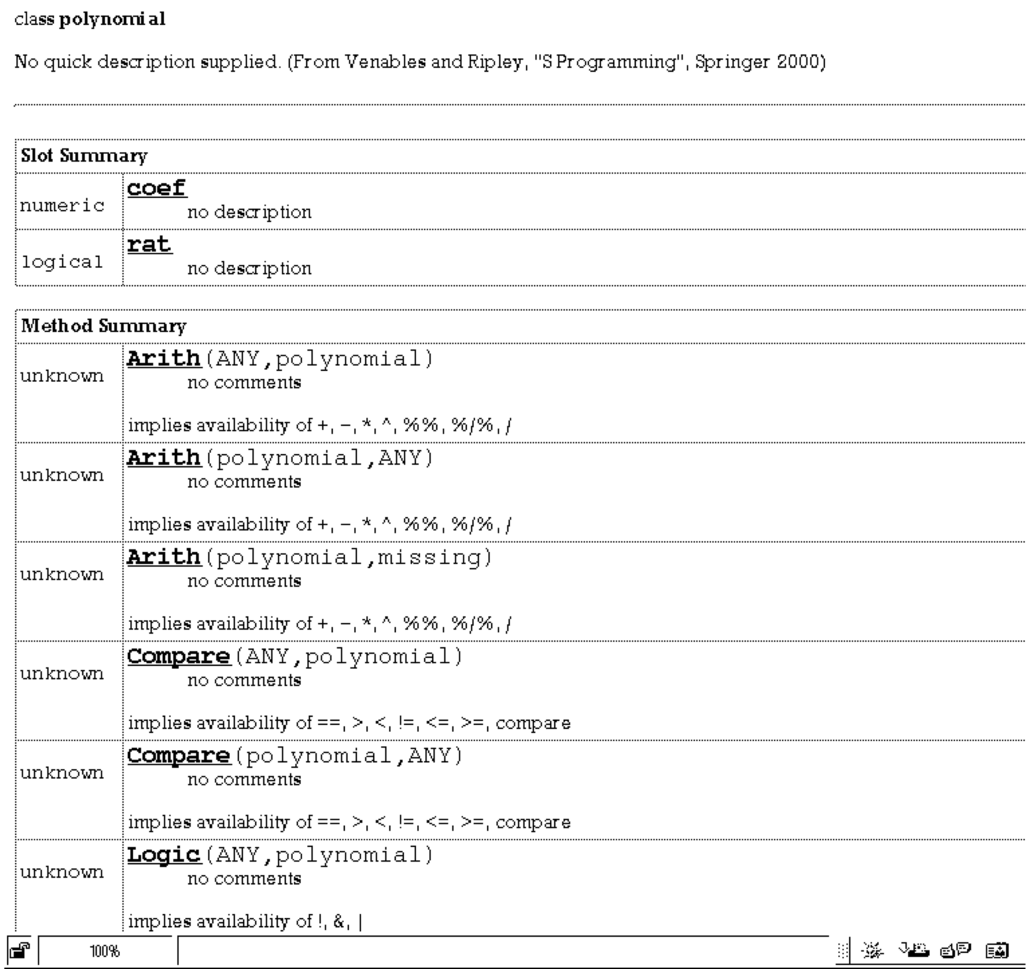
[12] W. N. Venables and B. D. Ripley. *S Programming*. Springer, 2000.

Table 1: User CPU times in seconds for 2000 runs of linear regression with an $N \times p$ design matrix in R.

			Java
N	p	lsfit	“accept”
20	5	9.9	7.8
100	5	12.8	17.5
100	20	18.1	143.3
1000	5	34.2	128.0
1000	20	119.1	1020.0

Figure 1: Illustration of XML source for `geeweb.xml`

```
<section>
 <title>
  Splus interface to yags
 </title>

<p/>
The main .q files are yags.q and yags.CC.q, the latter
isolated as the interface to the C++ engine.

<code modname="yags.q" lang="S">
if (version$major > 4)
{
setOldClass(c("pmat","block","vstack"))
setOldClass("yags")
}
  <fragmentRef id="the modeling wrapper"/>
  <fragmentRef id="the main fitter"/>
  <fragmentRef id="scale estimator examples"/>
  <fragmentRef id="correlation estimator examples"/>
  <fragmentRef id="correlation estimator estimating equation examples"/>
  <fragmentRef id="working correlation inverse examples"/>
  <fragmentRef id="extended GLM family support"/>
  <fragmentRef id="the pmat class"/>
  <fragmentRef id="programming miscellanea"/>
  <fragmentRef id="printing utilities"/>
  <fragmentRef id="test data"/>
</code>

<subsection>
<title>
GEE model formula interface for yags
</title>

<code lang="S">
<fragment id="the modeling wrapper">
yags &sgets; function(formula, id, weights = NULL, cor.met = NULL,
       family = gaussian, alpfun ...
    <fragmentRef id="formula processing"/>
...
</fragment>

Tangling with Xalan:
testXSLT -in ydemo.xml -xsl codeOnly.xsl -param modnm "'yags.q'" > ydemo.q

Weaving with Xalan:
testXSLT -in ydemo.xml -xsl article.xsl > ydemo.html
```



class **polynomial**

No quick description supplied. (From Venables and Ripley, "S Programming", Springer 2000)

Slot Summary	
`numeric`	**`coef`** no description
`logical`	**`rat`** no description

Method Summary	
unknown	**`Arith`**`(ANY,polynomial)` no comments implies availability of +, −, *, ^, %%, %/%, /
unknown	**`Arith`**`(polynomial,ANY)` no comments implies availability of +, −, *, ^, %%, %/%, /
unknown	**`Arith`**`(polynomial,missing)` no comments implies availability of +, −, *, ^, %%, %/%, /
unknown	**`Compare`**`(ANY,polynomial)` no comments implies availability of ==, >, <, !=, <=, >=, compare
unknown	**`Compare`**`(polynomial,ANY)` no comments implies availability of ==, >, <, !=, <=, >=, compare
unknown	**`Logic`**`(ANY,polynomial)` no comments implies availability of !, &, |

Figure 2: Output of `genDoc("polynomial",fraglist)`.

Figure 3: A DTD for coordinatized data.

```
<!-- Coordinatized Data DTD: CoordzDat.dtd -->
<!ENTITY % cdesc 'coordDesc (age|IdioTemporal|IdioSpatial|
                          IdioSpatiotemporal|
                          GMT|GPS|pedigree|otherC) #REQUIRED'>
<!ENTITY % vdesc 'valDesc (numeric|char|interval|otherV) #REQUIRED'>
<!ATTLIST obsSeries
    name CDATA #REQUIRED
    %cdesc
    coordUnits CDATA #REQUIRED
    %vdesc
    valUnits CDATA #REQUIRED>
<!ELEMENT CoordzDat (record+)>
<!ELEMENT record (key,obsSeries+)>
<!ELEMENT key (#PCDATA)>
<!ELEMENT obsSeries (cvPair+)>
<!ELEMENT cvPair (coord,val)>
<!ELEMENT coord (#PCDATA)>
<!ELEMENT val (#PCDATA)>
<!-- -->


Example datum: weight in pounds at 11 months

  <obsSeries name="weight" coordDesc="age"
         coordUnits="months" valDesc="numeric" valUnits="pounds">
   <cvPair>
    <coord>
    11
    </coord>
    <val>
    23
    </val>
   </cvPair>
```

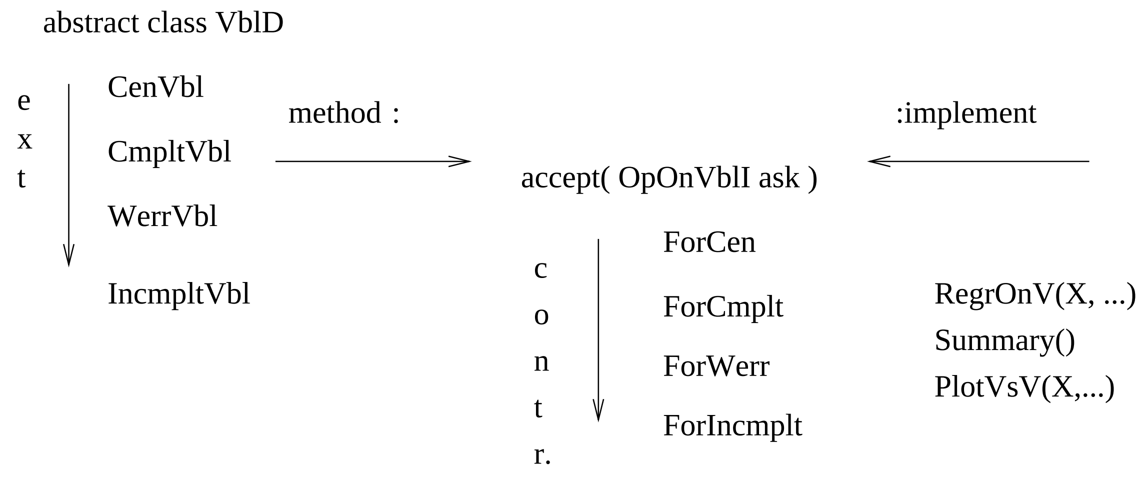


Figure 4: Some Java classes/interfaces for statistical modeling. `CenVbl`, etc. extend the abstract class `VblD`, and the interface `OpOnVblI` specifies the contract that all `VblD`-applicable
operations must have explicit methods for censored, incomplete, etc. data types.