Omega-Stat: An Environment for Implementing Intelligent Modeling Strategies

E. James Harner and Hanga C. Galfalvy
West Virginia University

Key Words: Modeling system, modeling strategies, tree structure, semantic map, dynamic graphics, expert system, Lisp programming environments.

Abstract

Omega-Stat is a new statistical environment built on Lisp-Stat, an object-oriented statistical programming environment. It contains extensible, reusable-component libraries for performing data management, multivariate analyses, modeling, and dynamic graphics. The point-and-click user interface allows instant access to all objects, including analysis and graphics objects comprising a semantic map. Knowledge, and methods for accessing this knowledge, are embedded within model objects and edge objects linking these models. This will allow the modeling process to be studied by following the analysis trails of expert analysts. The objective is to provide an "expert consultant" that is accessible as part of the man/machine interaction. Modeling strategies can then be built into Omega-Stat by using prior knowledge and data-analytic heuristics to guide the process of constructing the model tree and the iterative search for an "optimal" model.

Introduction

Omega-Stat is an integrated, user-friendly, extensible, object-oriented environment for data analysis, modeling, and dynamic graphics based on Lisp-Stat. Early versions of Omega-Stat are described by Harner (1990, 1991). Lisp-Stat is a prototype-based, object-oriented statistical programming environment developed by Tierney (1990). This paper presents the modeling component being developed within Omega-Stat. Modeling is completely integrated with the data management, dynamic graphics, and multivariate components of Omega-Stat.

The modeling environment builds on the ideas of many researchers, but those of Thisted (1986) and Oldford and Peters (1986) are most relevant. Specifically, the modeling process generates a tree structure with nodes representing expanded dataset models. A brief summary of the underlying statistical objects in Omega-Stat is presented first to provide the backdrop for the modeling system. A complete description is given in Galfalvy (1994). The modeling environment is described next, including both the underlying objects and the strategies for analysis.

Basic Statistical Objects: Variables and Datasets

A dataset—a collection of variables with value-lists of equal length—is the basic data structure for performing statistical analyses in Omega-Stat. Most analyses begin by selecting variables from a dataset according to their role. These variables are the arguments to a constructor function which returns a child dataset with a pointer to its parent. For example, a single response variable, with role *Y*, and one or more explanatory variables, each with role *X*, are selected prior to choosing **Model** from the **Data Browser** menu (see Figure 1). Analysis or graphical messages can also be sent to an individual variable. In this case, the constructor function returns a dataset, but the parent is a variable.

Both dataset and variable objects contain metadata (Hand, 1993), i.e., properties and descriptors about the variables and datasets themselves. A variable object has slots for its name, label, type (numeric or categoric), role (*Label*, *X*, *Y*, *Z*), admissible values, order (if categoric), and its distribution which, along with its values, define a variable. These can be defined initially, specified later (except for the variable values), and changed at any time. During initialization of numeric variables, information on discreteness, sparseness, and other descriptive properties are computed and saved. Similar considerations apply to categoric variables. During initialization, categoric variables save information on the actual and formal levels, level frequencies, con-

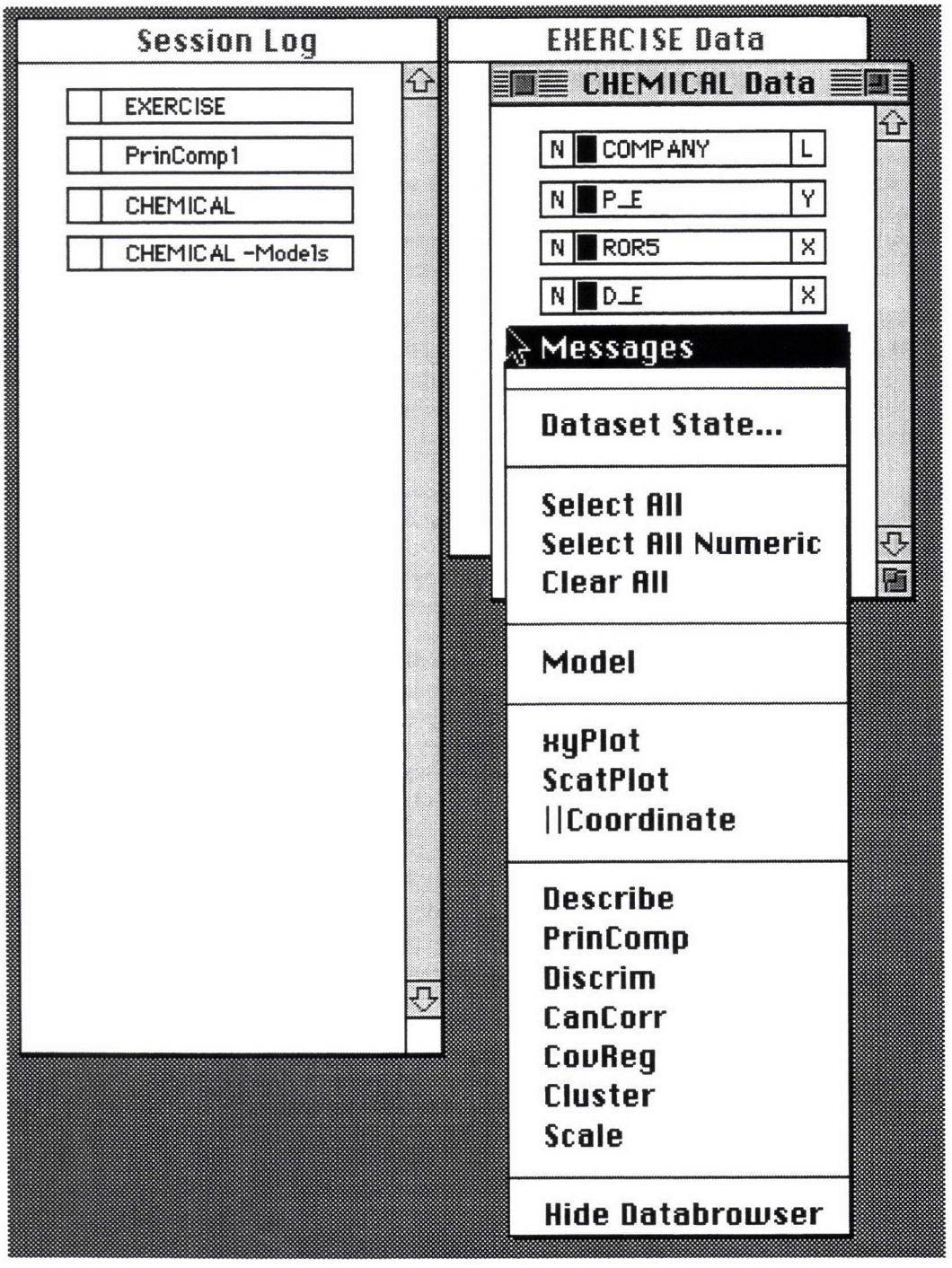

FIGURE 1. Session Log with dataset object subviews, Data Browsers, and a Data Browser Menu.

trasts, etc. At any time, users can also have other numerical information computed and automatically saved within variable objects, e.g., information on symmetry.

Metadata for datasets contain information on the nesting and crossing relationships among categoric variables, variable groupings, and special constraints, such as that imposed by composition data. As analysis proceeds, additional information is stored within datasets or their descendants. For example, the results of generalized singular value decompositions on groupings of numerical variables give information on col-

linearity, missing values, and multivariate outliers; on groupings of categorical variables, similar decompositions give information on dependency relationships and "cell outliers." The results of the former should be considered prior to and during regression analyses, whereas the latter is relevant to log-linear and analysis of variance models.

This metadata is used for data validation initially; subsequently, it is used to constrain the admissible analyses. Furthermore, this *a priori* specified (and computed) metadata is used along with diagnostics in assessing the efficacy of statistical models and in guiding statistical model selection. For example, models with count data as the dependent variable suggest examining the mean-variance relationship to determine if the Poisson, negative binomial, or some other probability model is appropriate.

Every statistical object currently created by a constructor function, e.g., a principal component, discriminant, or model instance, is an extended dataset, which has dependency relationships to its higher-level datasets. A dataset is said to be extended in the sense of Thisted (1986), i.e., it is a *data-analytic artifact* which has pointers to the original statistical and state variables, model-derived variable expressions, indicator variables, numerical descriptors, and dependency views.

Datasets, whether specified by the user or generated by a constructor function, are visualized iconically in the **Session Log** (Figure 1). **Session-log-proto** has a slot containing a tree with nodes containing pointers to the actual datasets and linkages representing parent-child relationships. This dependency tree is the basis for updating child dataset objects when a parent object is changed. Similar dependency relationships (and updating considerations) exist between dataset model objects and their views.

Within a child dataset, the underlying variables from a parent dataset are never replicated and derived variables are stored as expressions. This economy is necessary since numerous dependent datasets are created during the analysis process. For example, a principal component dataset has a slot containing pointers to the variables underlying the model and to the constructed principal variables (linear expressions with the right singular vectors as coefficients) and a slot for the principal variable scalings (the singular values), but not for the canonical basis of the principal variables (the left singular vectors).

The principal variable objects are parameterized by a power (α) of the principal variable scalings, for $0 \leq \alpha \leq 1$. This allows any plot involving the principal variables, e.g., a high-dimensional dynamic projection biplot, to be animated on α, which in the case of a biplot smoothly transforms the representation in terms of the principal component scores and principal variable coefficients ($\alpha = 1$) to one approximating Mahalanobis interpoint distances and the original covariance structure ($\alpha = 0$).

Statistical dataset objects are abstract data types with the following data access methods: *recognizers* for determining dataset types; *selectors* for extracting or modifying object slots, and *constructors* for creating dataset instances. Datasets, as abstract objects, exist independently of their views. Generally, datasets have multiple views: iconic, data browser, attribute (slot), textual, and graphical.

The standard representation of a dataset is its iconic view. Dataset icons are actually subviews of the **Session Log**, which shows the hierarchical structure of related datasets and provides an analysis history mechanism. The controller for this subview is a popup menu for sending messages to the dataset object.

The **Data Browser** view has variable objects as subviews (Figure 1). The controller for variable icon subviews is a popup menu for sending messages to a variable, e.g., to visualize the variable values in a quantile plot. The messages depend on the type of the variable object: numeric or categoric variables. The controller for the **Data Browser** is a popup menu for sending messages to collections of variables. Depending on the message, variable roles need to be specified, e.g., as response (Y), explanatory (X), or nuisance (Z) variables. Dialog views of variables and datasets show their state information.

Datasets generally have one textual view, which summaries the numerical information about the dataset, and many graphical views. Graphical-view constructors actually create a new dataset object which in turn is visualized. Many views are available or are being developed. These include various types of quantile, condi-

tioning, and multivariate plots. For example, **xyPlot** provides "guided tours" (Hurley and Buja, 1990) of user-specified subspaces constrained by the roles of the variables.

Graphical views have two types of controllers (input devices): methods built into the view itself or object instances of controller prototypes. The latter controllers are reusable components which are available to any graphical view object. Although views and controllers have explicit links to each other and to their dataset object, datasets do not have explicit links to their views and controllers. However, implicit links exist through the dependency trees of **session-log-proto**, since views (and child datasets) must be given the opportunity to reflect changes made to their underlying dataset object. Protocols determine how views are updated.

Datasets also contain state variables which determine how individual observations are represented in dataset views. These "variables" have point labels, symbols, colors, and states (invisible, hilited, etc.) as values. State variables are defined in the top-level dataset and are "inherited" by child datasets. This forms the basis of the strong linking among dataset views, e.g., dynamic hiliting when the mouse is in brushing mode. Datasets can also contain user-defined logical (indicator) variables to flag outliers, identify groups, etc., and are specific to a dataset. A special logical variable defines a mask and is used to select, generally by hilighting points in a plot, a subset of the parent dataset for analysis.

Omega-Stat currently supports two classes of datasets—those derived from **multivariate- proto** and those from **model-proto**. The remainder of this paper focuses on dataset objects derived from **model-proto**. The principal difference between multivariate and model dataset objects is in the depth of their dependency trees. Multivariate datasets typically are shallow since multivariate analyses are often descriptive and generally do not involve search strategies for "optimal" models.

Model Objects

All prototypes and model objects inherit from **model-proto**, which has no instances. **Model-proto** is a basic or foundation prototype; it contributes functionality and attributes that all models share. Currently, two prototypes inherit from **model-proto**; these are **linear-model-proto** and **nonlinear-model-proto** (see Figure 2). Each of these is "specialized" by prototypes allowing generalized linear and nonlinear models, respectively.

As described above, model objects are actually datasets which have report and graphical views. However, unlike multivariate dataset objects, only the foundation dataset is recorded in the **Session Log**. All other "descendant" datasets are visualized in a **Model Browser** which gives an overall view of the model dependency tree and the inheritance links to prototypes. This approach was taken to eliminate complexity in the **Session Log**, since model dependency trees can grow rapidly and become deep. Furthermore, **session-log-proto** is too general and does not contain sufficient structure for developing intelligent modeling strategies.

The **Model Browser** is a powerful tool for communicating with model dataset objects, which are represented by iconic subviews. The controller for these model object subviews is a popup menu (Figure 2). It contains messages for producing summary and diagnostic views of the current model object and for creating new model objects with dependency links to the current object.

The modeling portion of Omega-Stat is an implementation of Thisted's paradigm for data analysis (Thisted, 1986), i.e., the model objects are extended datasets which are viewed as nodes in a dependency tree. Furthermore, edge subviews connecting the nodes form the basis of a semantic map. The edge object contains information on the heuristics used to generate the next model object (node). The controller for the edge subview is a popup menu which provides the mechanism for the data analyst to record modeling explanations.

Fitted models are *data-analytic artifacts* as discussed above. Depending on the parent prototype of a model, the extended dataset contains expressions for fitted values, residuals, leverages, etc. Perhaps most importantly, it contains symbolic representations of the model. The model formula *scanner* and *parser*, as well as a *constructor* for the model matrix, was developed by Hasebe (1994). Both the syntax and semantics of model formula are based on the approach taken in S (Chambers and Hastie, 1992), except that both infix and prefix notations are permitted here.

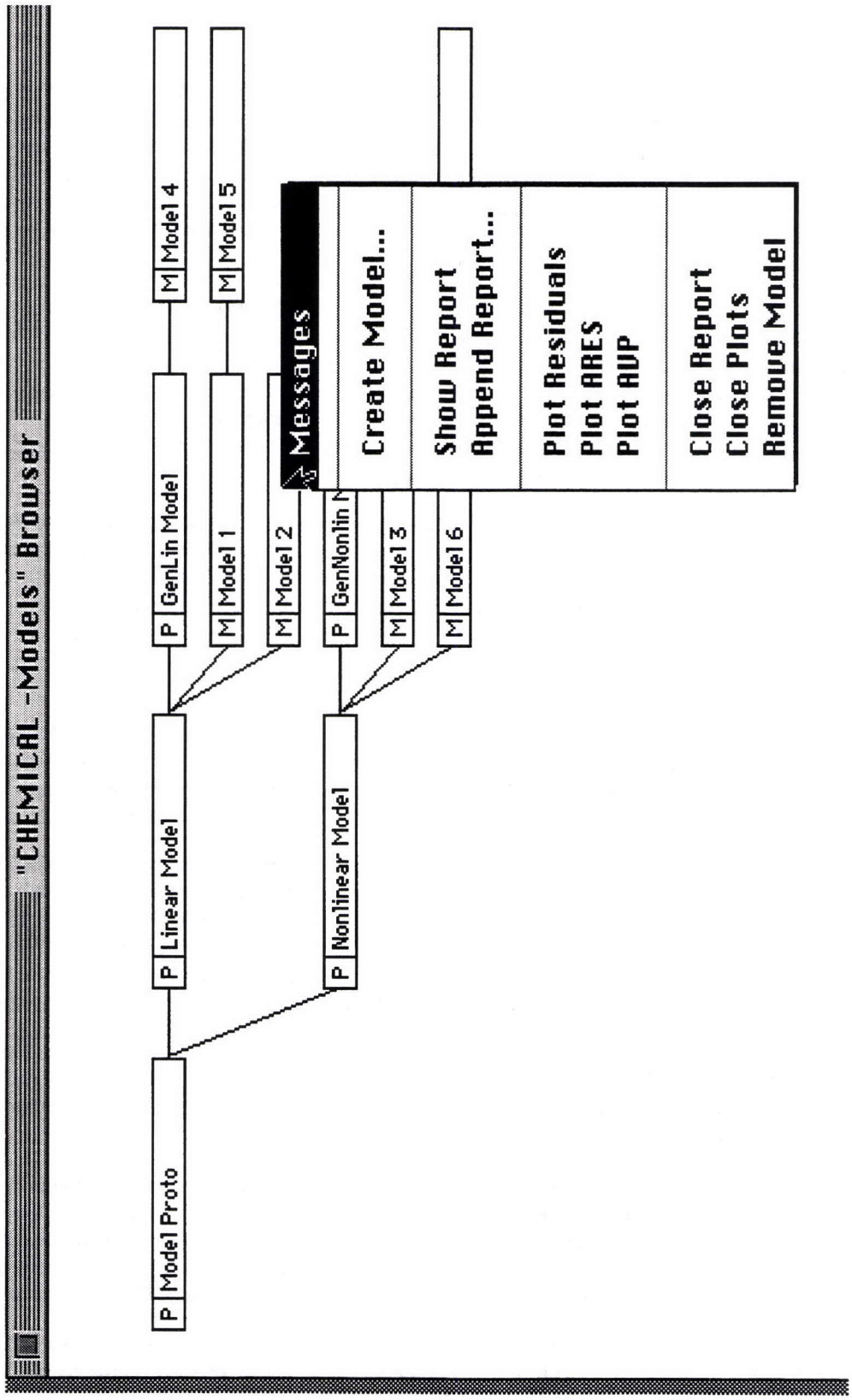

FIGURE 2. Model Browser with its model dependency tree and a model popup menu.

Modeling Strategies

Modeling entails node exploration and node transformation strategies. Messages selected from the controller of a model iconic subview either create another model (a node transformation) or produce summary and diagnostic views (node explorations).

Node explorations strategies, which are directed at a specific model instance, consist of both low-level and intermediate-level methods. Low-level diagnostics use "primitives" already available in the extended model dataset, e.g., a residual expression, to produce views which are generally graphical. Intermediate-level explorations, such as collinearity diagnostics, build on the "primitives" or require new computations to generate views, which are often both textual and graphical. Recall, however, that graphical views are themselves based on an extended datasets; thus, each specific model object not only has model dependents, but also dataset dependents associated with diagnostic graphical views. The dataset graphical views of a model can be visualized as subviews in its model diagnostic browser.

Most of the diagnostics for model checking are based on methods owned by **model-proto**. These form a group of diagnostics which in their basic form apply directly to linear models, but weighted versions apply to generalized linear and nonlinear models. These include case-deletion diagnostics. Generalized singular value decompositions, with appropriate row and column weights, allow collinearity and other diagnostics to be made.

The inheritance structure of the model prototypes will allow approximate diagnostic methods, based on linearizing higher-level models, to be replaced by more precise methods as they become available. This is the justification for having **generalized-linear-model-proto**, which contains linear models as a subset, inherit from **linear-model-proto** rather than the reverse. Each prototype in the inheritance chain has additional diagnostics; for example, checks on the link and variance function are added for generalized linear models.

The information gleaned from the model diagnostics are saved in slots in each of the model objects. This includes numerical or categorical information summarizing plots and other diagnostics. For example, assessments of trend using smoothing algorithms for added variable plots or explorations using animation in *ARES* plots indicate whether an omitted variable should be included, and in what way, as an explanatory variable (Cook and Weisberg, 1994). Currently, this diagnostic information is added to the model object by the analyst, but automated procedures are planned.

Once diagnostics are "completed" on a model object, the modeler shifts attention to selecting the next model specification, i.e., finding a node transformation based on heuristics applied to the knowledge gained from exploring models in the dependency tree. Node-transformation strategies are termed high-level and also depend on protocols for evolving the dependency tree. For example, a model with minor changes relative to the currently selected model (e.g., deleted outliers) is created as a direct descendant of the current model, whereas a model with major changes (e.g., a different link function) is created as a descendant of the appropriate prototype. The modeling process is helped by the flexibility of the symbolic representation of the model formula and the ease of changing state information. Furthermore, the analyst can move around the dependency tree to reexamine or perform more diagnostics on previously generated models.

Modeling is often done in small steps, but experts move purposely as they select the next transformation and narrow down the multitude of possible models to a manageable number. As has been mentioned previously, the modeler is encouraged to explain why a parent or prototype gave birth to an offspring. This process creates a semantic map which provides insight into the modeling process in general and an understanding of the generated models in particular.

Future Work

The current dataset model is limited in two important areas. First, regular data (e.g., time series and spatial data), sparse data (e.g., species abundances), and other data structure are not implemented. Although these data types can be represented in datasets, it is not efficient. Also, datasets containing categorical variables have redundancies, which would not be present in multi-dimensional arrays indexed by the categories of

these variables. More general data structures are possible within the Lisp-Stat object system, but the currently implemented statistical operations require a dataset (or variable) structure. As spatial and time series models and scientific visualization plots are added, other data structures will be developed.

Secondly, statistical database objects are not implemented (e.g., see Michalewicz, 1991). Datasets are arranged hierarchically in **session-log-proto** and in **model-proto**, but more general dependency relationships and dataset constructor operators are needed. This added functionality will define the specifications for an object-oriented statistical database system, which is essential for large-scale research projects.

The search strategy described thus far is informal, but it provides a method of developing man/machine strategies for modeling. After examining the semantic maps of expert (and perhaps non-expert) analysts' attempts at modeling targeted datasets, a rule-based system will be developed encompassing search strategies for reaching one or more acceptable models. The system would need to incorporate prior substantive knowledge of the problem domain, use metadata, interpret diagnostics (created by man or machine) of the models in the tree, represent this knowledge, and apply heuristics from this rule-based system. The objective is for the machine-resident "expert consultant" to suggest, but not enforce, the next node transformation. At each step, the search process is repeated until an "optimal" model is attained.

References

Chambers, John M. and Hastie, Trevor J. (1992), *Statistical Models in S*, Pacific Grove, CA: Wadsworth.

Cook, R. Dennis and Weisberg, Sanford (1994), *An Introduction of Regression Graphics*, New York, NY: John Wiley & Sons.

Galfalvy, Hanga C. (1994), An Object Environment Based on XLISP-STAT for Multivariate Analysis and Dynamic Graphics, Master's Project Report, Department of Statistics and Computer Science, West Virginia University.

Hand, D. J. (1993), "Measurement Scales as Metadata," in *Artificial Intelligence Frontiers in Statistics*, D. J. Hand (ed.), London: Chapman & Hall, 54-64.

Harner, E. James (1990), "Interactively Developing Models Based on the Exponential Family," in *Computing Science and Statistics: Proceedings of the 21th Symposium on the Interface*, Kenneth Berk and Linda Malone (eds.), 21: 110-115.

Harner, E. James (1991), "An Exploratory Data Analysis and Modeling System Based on Lisp-Stat,"*1991 Proceedings of the Statistical ComputingSection, American Statistical Association*: 70-78.

Hasebe, Yoshinori (1994), A Model Parser and Constructor for Generalized Linear Models, Master's Thesis, Department of Statistics and Computer Science, West Virginia University.

Hurley, Catherine and Buja, Andreas (1990), "Analyzing High-Dimensional Data with Motion Graphics," *SIAM J. Sci. Stat. Comput.* 11, No. 6, 1193-2111.

Michalewicz, Zbigniew (1991), *Statistical and Scientific Databases*, New York, NY: Ellis Horwood.

Oldford, R. Wayne and Peters, Stephen C. (1986), "Implementing and Study of Statistical Strategy," in *Artificial Intelligence & Statistics,* William A. Gale (ed.), Reading, MA: Addison Wesley, 335-353.

Thisted, Ronald A (1986), "Representing Statistical Knowledge for Expert Data Analysis Systems," in *Artificial Intelligence & Statistics*, William A. Gale (ed.), Reading, MA: Addison Wesley, 267-284.

Tierney, L. (1990), *LISP-STAT: An Object-Oriented Environment for Statistical Computing and Dynamic Graphics*, New York, NY: John Wiley & Sons.