$\widehat{\Omega}$- A Component-based Statistical Computing Environment

John Chambers, Duncan Temple Lang
2C-259 Bell Labs, 600 Mountain Ave., Murray Hill, NJ 07980, U.S.A
jmc@research.bell-labs.com, *duncan@research.bell-labs.com

Nous décrivons des aspects importants des systêmes à venir pour la statistique, dont les réalisations prévu en cadre du projet collaboratif $\widehat{\Omega}$.

1 A Component-based Environment

Computing in general is undergoing tremendous change due to, amongst other things, the Internet and advances in hardware. Statistical computing is subject to these and additional changes - increased volume of data, use of computationally intensive methods, dynamic reporting, etc. Languages such as *S* have made significant strides in modernizing statistical computing. Influenced by developments from computer science, these environments provided a vehicle tailored for statistical computing into which new developments could be easily integrated. It is probable that the next generation of concepts will be more difficult to add to these environments, designed for a different style of computing. As a result of this and the importance of computing within the statistical community, we feel that it is prudent to start developing the next-generation environments now. In the remainder of this section, we look at what features are needed and desirable in a good, modern statistical computing environment that will help us in the next decade. In the second section, we discuss a project to implement such an environment. The list of features of a modern/future environment is neither exhaustive, nor arranged in any particular order.

- The core of the environment should be usable from multiple front-ends, including traditional text-based command line and graphical user interfaces (GUIs). Also, it and its scripting languages should provide both interactive access as well as being embedable at different levels within other applications.
- The environment should support *development* of GUIs for use by different audience/user types as well as having its own GUIs.
- Extensibility at both the language/interpreter level as well as at the native core level is vital for a flexible, adaptable and long-lived system.
- Internet capabilities including the ability to read data to and from network applications is just one aspect of network awareness. We must also be able to run code easily across the network in a portable manner.
- Databases access, rather than static snapshots of data provided in ASCII files, are becoming more important and allow us to move to a dynamic analysis process.
- Of course, we require facilities for describing models and procedures for fitting them. The environment should support utilizing different implementations of these procedures to handle different volumes of data, different levels of precision (i.e. sampling), etc.

- Interactive graphics and connections with other graphical controls should be easily programmed. Additionally, we would like to be able to embed "live" plots (and computations) within documents and worksheets along with other computations. In this way, we achieve a general analysis sheet, an extension of the spreadsheet mechanism.
- The environment should provide facilities for distributing tasks across multiple processors within a process, and across distributed machines within a cluster. It should encourage research on parallel algorithms.
- Access to existing code written in *Fortran* and *C* and the current family of statistical languages is needed to avoid rewriting existing and well tested code. Providing the dynamic CORBA facilities mentioned below in the interpreted languages will permit the latter. Implementing the new environment using languages that provide an interface to compiled code will handle access to *Fortran* and *C*.
- The environment should be able to utilize services of other processes (servers) in other languages, potentially on other machines. One motivation for this is to be able to avail of new developments in other disciplines, including newly released software. This is the CORBA model and easy access from within the environment is necessary. Additionally, we want to be able to integrate new software directly into the environment without tedious wrappers.
- Performance will always be an important aspect of any environment for doing statistical analysis. High-level languages such as *S* and *Matlab* have made different trade-offs between ease of use and efficiency. An ideal environment will support the broad spectrum from ease of use to efficiency within the same framework rather than forcing the user to change between environments for prototyping and production code.

Most of these features can be implemented using a *component-based architecture*. This involves coupling data and external access via methods or services into a single unit called a component. Rather than building a new monolithic system which statisticians enter, the goal is to provide a collection of small tools which the statistician easily links together to fit her needs. The result may appear the same but the foundation is very different and significantly more flexible. This architecture supports distributed computing and parallel processing by communication between components. It also allows rapid substitution of components with the same methods to utilize different implementations. Encapsulating the data from the user interface via methods allows the different interfaces styles to be overlaid on the environment (dynamically) and for the components to be embedded.

Experience with CORBA and other systems illustrates that components should be described in terms of their services and not their implementation. This allows them to be easily substituted for different behavioral characteristics and to be language- and hardware- neutral. The same experience also shows that there are new issues to be dealt with in this architecture. These include security, reproducibility of computations, component/service discovery, configuration management, performance.

2 Realizing the Component-Based Environment

Given the list features above and related issues, we need to determine how to implement such an environment. Of course, everyone has their own favorite language. However, a natural candidate is *Java*^{TM}. It provides facilities for almost all of the desired features. It is object-oriented supporting extensibility of the base implementation of the environment's components. It also provides *reflectance* allowing interpreted access to native objects and their methods. Thread, GUI and network support are provided in the language and its core libraries. It is portable and also has unparalleled momentum in the computer world, with new libraries and tools being distributed at a phenomenal rate.

Portability comes at the expense of speed in *Java*^{TM}. Recent developments have improved performance, but there is still a real decision to be made regarding the trade-off between flexibility and performance. It is likely that one will soon be able to compile *Java*^{TM} source directly to machine code and obtain performance similar to *C*which is encouraging.

2.1 Enhancing Existing Systems

Before we start building a new system, we must allow the extensive functionality in the existing systems to be accessed as components. One of the projects we recently undertook in Bell Labs was to develop a framework for dynamically embedding CORBA facilities within an interpreted language such as $\widehat{\Omega}$, *S*/*R*and *Xlisp-Stat*. This allows methods to invoked in CORBA objects from within these languages with almost no additional work from the caller. Similarly, local objects within these languages can be used as inputs to these calls and as regular CORBA server objects. The implementation of their operations involves functions in these languages themselves, presumably already developed for local use. This is described in more detail at www.omegahat.org/CORBA/

2.2 The $\widehat{\Omega}$ Environment

The $\widehat{\Omega}$ project is an effort to create the environment described in the first section of this paper. It currently includes 15 people involved in developing the primitive components for the language, graphics and modeling. These include the owners of 3 statistical systems - *S*, *R*and *Xlisp-Stat*. Most of the others are all actively involved in the development of *R*. Other individuals are contributing to the development of specific modules rather than the overall environment. It is an open source project in an effort to collect the small statistical computing community in a united effort.

Now that we have discussed the philosophy of the environment and its design, we will briefly mention some of the specific features it offers and describe the current status. More comprehensive and up-to-date information is available at www.omegahat.org and at the presentation at the ISI meeting (and Joint Statistical Meetings in Baltimore).

2.3 The Interpreter and Language

There is currently one interactive interpreted language for the $\widehat{\Omega}$ system. The user typically invokes commands from either a text-oriented command line or a integrated GUI which provides an command field, output displayed in HTML, system information (search path, available classes, debugger, etc.) viewer. The basic interpreter(s) that evaluates these commands can also be embedded in other *Java*^{TM} applications. There can be multiple evaluators in existence at any time, potentially running concurrently in different *Java*^{TM} threads.

The language is a hybrid of the *Java*^{TM} and *S*/*R*languages. It is object-based in that every variable is a *Java*^{TM} object and methods are invoked on that object. It supports functions and extensible operator overloading (i.e. the subscripting operators [& [[, and the *get within* operator, `.`). Generic (overloaded) functions are allowed and interpreted classes provide an alternative to compiling *Java*^{TM} source. One of the most important distinctions from *S*/*R* is that $\widehat{\Omega}$ uses references to objects in the

Java^{TM}-style. Thus, when one passes an object to a function which changes its contents, the changes are reflected in the original object.

One of the more remarkable features of the $\widehat{\Omega}$ language is that *all* *Java*^{TM} classes and objects are available to the interactive user. No wrapper functions need to be created to provide an interface between the native language in which the interpreter is written and the interpreted language (compare with the `.C()` in *S*). Specifically, one can create an instance of a *Java*^{TM} class and bind it to an $\widehat{\Omega}$ variable and manipulate it directly. The methods of an object can be invoked interactively with appropriate arguments. This mechanism supports adding classes to the *Java*^{TM} classpath dynamically, be they located in directories, jar/zip files or even on remote machines accessible via the Internet. New classes can be dynamically generated and loaded within a running $\widehat{\Omega}$ session. This allows functions to be used to implement *Java*^{TM} interfaces.

The evaluation model is very similar to *R*, using lexical scoping. A top-level expression is evaluated by resolving the methods and variables from the databases in the search path attached to the interpreter. Sub-expressions are evaluated in their own frames and free variables are located by searching in the parent frame, and so on. Used in this way, the language appears more friendly than *Java*^{TM} as variables do not have to be declared and can be used to store values with very different types.

Optional type declarations are supported. The user can choose to provide a class name for a variable, function parameter or return value, and have the system detect violations of this type, aiding in debugging. This helps programming large collections of functions and packages. Also, this information can be used by the system to compile interpreted expressions and functions into *Java*^{TM} byte code. This allows the user to prototype in a dynamic interpreted language, but later benefit from the speed of compiled code without recoding in a different language. The optional type-checking supports different levels of users and different programming tasks.

The ability to access all *Java*^{TM} classes means users can chose their own primitives which may support vector operations, etc. Similarly, all the *Java*^{TM} utilities such as JDBC and XML are immediately available.

The dynamic CORBA facilities are fully integrated, allowing remote calls and variables to appear local. A task management facility is currently being developed that exploits native threads and clustered machines.

2.4 Graphics & Modeling

Work has just started on both the graphics and modeling capabilities. In the former, we are constructing primitive elements from which a multitude of different plot types can be constructed. One focus is on separating the layout of multiple plots from their content. We are investigating ways of doing this via *Java*^{TM} layout managers and constraint systems. This allows arbitrary plots to be grouped in interesting ways, of which trellis is just one example.

In addition to static plots, the graphics elements support different interactive properties. Similarly, they can be embedded in GUIs (even used as dynamic icons in buttons and HTML documents). As with all of $\widehat{\Omega}$, the goal is to produce a flexible set of components on which others can be easily built and modified to explore novel approaches to visualizing data.

On the statistical modeling front, we are starting to extend the Wilkinson & Roger's formula language used in *S*. The aim is to incorporate more of the modern model specifications including (Bayesian) hierarchical models, graph models, mixed effects model, neural networks. In other words, we want to include multi-level specifications rather than simple structural components of a model.