Distributed Computing with Data: A CORBA-Based Approach

John M. Chambers, Mark H. Hansen,
David A. James, Duncan Temple Lang

jmc,cocteau,dj,duncan@research.bell-labs.com

Bell Labs, Lucent Technologies
Murray Hill, NJ 07974

Abstract

Statistical computing is part of a more general process, which can be called *computing with data*. Besides traditional statistical analysis, this involves acquiring, organizing, and visualizing data, often in large, structured datasets organized in database management systems and used for purposes beyond analysis. An important challenge for statistical computing (and statistics in general) is to increase the scope of our involvement in this diverse environment. At the same time, the computing environment itself is becoming more diverse in all respects: data and users are widely spread and using many different systems.

We describe research looking towards the next generation of software for such applications, centered on the idea of *distributed computing with data*. By this we mean distributed in two fundamentally different, but related, senses. First, the data and the tasks users apply to the data are distributed geographically, over a heterogeneous network of computers and operating systems. Second, the programming environment we envision is distributed over a variety of languages and other software.

We describe research towards a programming environment suitable for distributed computing with data. As a key to this environment, we propose to take advantage of the CORBA standard for distributed, object-oriented computation. This paper describes the background for our approach, the reasoning for the CORBA proposal, and some initial experiments in the new approach.

1 Introduction

Those of us who create software for computing with data—for the organization, visualization, and analysis of data—face an important opportunity and challenge. The *potential* scope of the use of our software has expanded greatly. How can we ensure that the actual scope will increase comparably in response?

The scope of computing with data has undergone (and continues to undergo) expansion on several dimensions, including:

- *computing power*,
- *number of services and overall size of applications*,
- *distribution* (the Internet and the World Wide Web), and
- *range of users and application developers*.

The combined effect is a vast potential for the use of data in all kinds of human endeavor.

The opportunity is to have the use of our software expand correspondingly: to use the enhanced computer power; to adapt to the new applications; to spread out over the Web; to attract and serve the new users.

Like any opportunity, this is also a challenge. If we fail to respond, other software will fill the niches opening up. Our software and, more importantly, the ideas that it implements will be increasingly relegated to the margins of this growing activity.

How can we respond? Clearly, so broad an opportunity will stimulate many responses and a good number of these will be valuable. The key limitation is the amount of time, care, and mental energy required from skilled practitioners. Many of the obvious reactions to the challenge involve too much work, too much duplication of effort, and too many *ad-hoc* solutions to fundamental problems.

This paper proposes an approach to the opportunity. The approach is the basis for current research in computing with data in Statistics Research at Bell Labs. More generally, though, the approach benefits greatly from co-operation among many groups. The more we can all find common ground the greater the general benefits. This paper outlines

the general basis for the approach and some areas where cooperation can be beneficial.

The general response needs to be in terms of what we will call *distributed computing with data*: an environment for programming and for using software that is distributed in four senses:

1. geographically, e.g., over the Web;
2. over different applications;
3. over different types of user interaction;
4. over different computer hardware and operating systems.

For example, the environment needs to accommodate client access to a wide variety of services, using the Internet and the Web seamlessly. The environment needs to allow applications to interface easily: re-implementing each facility in each system is a luxury we cannot afford. The environment needs to accommodate the increased range of users by providing graphical and other user interfaces suitable to less specialized users, but at the same time it has to provide powerful languages and computational systems to create the tools behind such user interfaces. Finally, the environment needs to take advantage of the increased variety of computers and of the potential for inter-connecting such a varied range of computers to accomplish difficult and specialized tasks.

There are a number of ways to approach the design of such an environment. Some related work has already been done, within our discipline and elsewhere. We believe one particular approach is compellingly attractive. It will be the basis of the major research project at Bell Labs in this area in the near future. We also propose the approach to the community in general, since it is an intrinsically open, cooperative, and all-inclusive attack on the problems identified above.

We propose the use of the Common Object Request Broker Architecture (CORBA) and in particular of the Interface Definition Language (IDL) associated with it, as the basis for distributed computing with data. The rest of this paper describes what this proposal means, argues for the advantages of this approach, and outlines with an example how the approach provides the key tools for distributed computing with data.

2 Examples and Previous Approaches

We will use one particular example that illustrates some essential requirements of many applications, and thus it is likely that readers have had to address very similar issues. Once the example has been displayed in Figure 1, most readers should be able to relate it to their own experiences just by relabelling the elements of the figure.

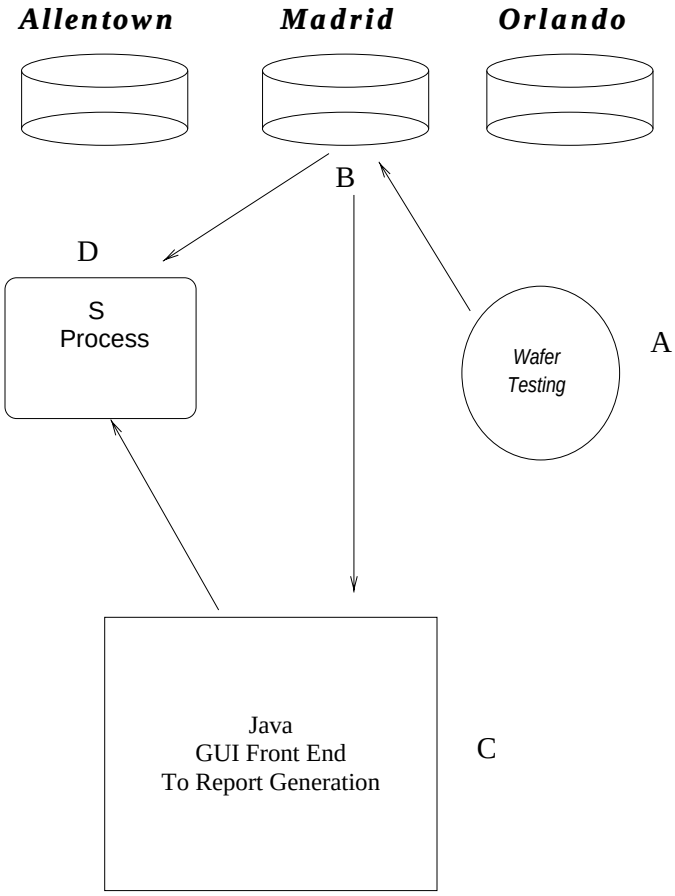

Figure 1: *Using distributed computing, via Java applets, S, and database software, to view test results from chip manufacture. Arrows indicate data transfer.*

2.1 Analysis of Manufacturing Data

The example involves the organization, visualization, and analysis of test results obtained at the end of the manufacturing of electronic chips and similar components (these components are manufactured on silicon disks called *wafers* in batches called *lots*). Specialized hardware (probe testers) examine each of the chips on each manufactured wafer for functional acceptability. Each test results in a code for success or failure (of various types). Every wafer manufactured at any of the factories of a company such as Lucent Technologies is subjected to these tests (as potential users would likely hope!), and the data from the tests is recorded in databases maintained at individual factory sites (in Lucent's case for example, in Allentown, Pennsylvania; Orlando, Florida; or Madrid, Spain, among other places).

The test results contain a lot of information; understanding it is critical for managing the manufacturing process. Spatial patterns of failures, variations over time, and the relation to various factors in the manufacturing process are all important and difficult aspects of using the data.

Techniques and software for visualizing and modeling the data have made important contributions in recent years. In particular, plots based on computations in S provided visualization of the test results that allowed engineers to see patterns much more readily and quickly. Interactive, non-programming interfaces to this visualization added to the ease of use. The interactive interfaces were then made available over the company intranet linking the users at widely separated sites.

Figure 1 illustrates the basic architecture: the probe test equipment at each location (A) generates data and stores the data in databases locally. The databases (B) are managed by database management software, often a standard system used for all the manufacturing data at the factory. To visualize the test results, the data is selected from the database, read into the statistical or visualization system and displayed. A graphical user interface allows users to choose different display styles and to select particular aspects of the data, as well as providing mechanisms for selecting the dataset of interest. The facilities can be provided on the Web by using suitable software to implement the displays and by providing user access through such mechanisms as CGI (Common Gateway Interface) scripts or Java applets.

2.2 Key Features of Distributed Computing

The example illustrates the key features of many modern applications: data are generated, often in quantity, for important purposes not under control of the statistical software; the data stored in Database Management Systems (DBMS) may be diverse in either geographical location or in the schema used; users are likewise widely distributed, using varied computing equipment, and not usually prepared to deal directly with general software for data analysis; oftentimes security considerations need to be addressed to ensure that proprietary information can only be accessed by authorized clients and users. With all these challenges, understanding the data by the appropriate people can be a very important contribution to the organization.

The software to support this example has evolved through several stages and a number of languages. The "core" software for visualization and modeling such data is the S-Wafers library (Hansen and James, [3]), which provides functionality in S to organize, analyze, and plot the data. This software, like most applications, needs to operate in a context where data flow into and from the system, with users directing the task to be carried out through some kind of interface, possibly graphical for those uninitiated in the statistical system itself. Visual Basic [2] and Java front-ends, for example, were both used to provide graphical interfaces to S-wafers in which communication among S and the databases was done through text files. To take advantage of the company Web, CGI scripts provided users with access to the data and to the plots.

In Figure 1, a user brings up a Java applet (C) and selects a database (B), which sends a list of available lot names to the applet for the user to analyze; the user selects some lots by name, and the applet then sends these selected lot names to the S process (D); the S process obtains the data corresponding to these lots from the database (B), and finally generates suitable plots.

In a precursor to the current project, Chambers, Hansen, and Mattingly [4] developed a general interface between S and Java, for which the wafers application provided a demonstration example. This architecture provided a general interface, unlike the earlier special-purpose interfaces (e.g., the Visual Basic Wafers-GUI described in [2]). S functions and Java applications or applets communicated by exchanging objects. Both languages implemented an interface to a general object format (using the string-based data dump format in S, [1, Chapter 5]). The computational results from S were communicated to the Java user interface as objects, and evaluation requests from Java were similarly passed to S as (string) objects. An additional Java application managed communication between the systems.

These approaches, and many similar ones in other examples, provide valuable software to users, but they are all to varying degrees unsatisfactory as the architecture for large-scale distributed computing with data. The simple appearance of Figure 1 hides a number of problems that make distributed programming difficult and can hurt the reliability of the resulting application.

2.3 Issues in Distributed Computing

There are potentially as many programming languages/systems in Figure 1 as there are boxes A to D. Each time one system acts as a client of another system, the client must know how to invoke a request from the particular server and how to interpret the data, if any, that the server supplies in response to the request. When the systems are, in addition, distributed over the Web, the client must know how to identify the server (for example, as a particular CGI interface on a particular site on the Web); it must then send its request to that particular server. For the application to be reliable, each client needs to ensure that the supposed server is available, that the request was communicated successfully, and that the server satisfied the request.

For the application developer, this situation results in prob-

lems such as the following.

- The communication process between systems is low-level, usually in the form of "flat files", streams of unstructured text, or special-purpose formats designed for the particular example. There is little potential for reuse of the communication structure and little general data structure to build on.
- The writer of an application needs to know enough of each server system's language and capabilities to transmit a request and interpret the result. In Figure 1, the writer of the test probe software needs to know how to create tables and supply data in the database system, while the writer of the S application needs to know how to identify and extract test data, and how to turn the result into an S dataset. Similarly for any other communication required.
- At best, the implementers of each piece application have to understand the details of communication management and worry about restarting computations, determining the cause of communication failure, and ensuring integrity of the transmitted data. At worst, no one has time to worry about such issues, leaving the user of the application unprotected.
- Security issues are either inadequately addressed or simply ignored.

Recognition of some of these problems stimulated the work on the S/Java interface. Experience with the initial versions of this software convinced us that distributed computing with data was indeed a viable and important next step. At the same time, it was clear that our attempts were solving, in a rather minimal way, general problems of communication and distributed computation. In addition, extending the design to multiple systems and convincing owners of other languages and systems to adopt such an approach were likely to be daunting projects.

What was needed was a general, powerful, and high-level approach that we could adapt for our needs rather than having to design and implement ourselves.

3 CORBA

We will not attempt a general discussion of CORBA here (see one of the many references, such as [5, 6], and the papers on the Web, such as those at `http://www.omg.org`). Instead we will mention some of the key aspects of CORBA for computing with data, and address a few of the questions about using it.

The Common Object Request Broker Architecture or CORBA is best understood in the context of object-oriented applications (although CORBA specifically provides interfaces to non object-oriented systems through the so-called delegation interfaces), and the 3-tier distributed model. This model defines a middle tier between clients and servers allowing for one common interface that all applications use for object location, method invocation, data transfer, plus many other services (this middle layer is referred to as the Object Request Broker or ORB).

As its name implies, CORBA is not a system, language or piece of software, but rather an architecture and a set of specifications. The overall goal is a standard for client/server computations, using objects distributed over a heterogeneous network of computer hardware and software. Implementations of the architecture enable application software to make requests of other software (acting as a client) and/or to provide methods that may be invoked by other software (acting as a server).

Reduced to a (very) skeletal description, here is the model CORBA provides for computations. Any computation offered by a server is an *object*; these objects provide their services as *methods* (or *operations* as CORBA refers to them). A client gets some computation by getting a reference to the object in the server; then, it invokes whatever methods it wants on that object.

For a number of reasons, this model greatly improves the implementation environment for distributed computing with data.

1. *Communication is invisible.* All the method invocations behave as if they were local computations. In particular, the client can ignore any question of where the service is located (this is oftentimes referred as *location transparent*).
2. *Programming is language-neutral.* Programmers developing clients in a particular language that provides access to CORBA do not need to know the details of what language was used to implement some object whose methods they want to use, e.g. the S syntax.
3. *Class definitions are explicit and general.* The data objects used and provided by servers and clients are explicitly defined using a language for class definition (*interface* in CORBA terminology) that matches well to such languages as Java, C++ and the most recent version of S [1].
4. *Everything is dynamic and self-describing.* An important feature of CORBA is that it provides for dynamically querying (and specifying) essentially all the information about classes and methods.

The first two items by themselves allow for direct use of CORBA to improve programming for distributed computing

with data; in Section 4 we will indicate how this would apply to the example in Figure 1. More general and more exciting possibilities arise from the last two items. Some possibilities are outlined in Section 5.

With all this, CORBA is definitely no free lunch, although there are some essentially free dishes such as the communication services. Support for CORBA is large and growing (Netscape Navigator is bundled with an ORB and the upcoming Sun's Java Developer's Kit 1.2 provides an IDL interface), but it has competitors, e.g., Microsoft's DCOM (Distributed Component Object Model), and others. And no architecture can magically solve intrinsically hard problems. Integrating deeply inconsistent views of data (for example, some aspects of statistical software with some aspects of relational databases) remains hard whatever the context. Organizing very large computational tasks in a distributed environment is hard because the underlying information content and the algorithms to produce don't easily convert to a distributed environment in most cases. Providing easy but flexible and extensible interfaces to important computations with data requires much care and considerable inspiration.

In all these hard problems, however, the importance of choosing a better underlying architecture, such as we believe CORBA provides, is that we are left with more time to work on the really hard or important tasks, with less time wasted re-solving old problems, duplicating other systems' software, or worrying about the details of distributed computing.

4 Using CORBA

As we noted before, Figure 1 makes the interaction among its component systems look simpler than conventional implementation techniques allow. The arrows in the figure generally involve sending some data from one part of the system to another. We would like to think that we could create a "reasonable" definition of the data involved and then just make the request. The test data would be stored in the database; the statistical system would get it from the database and do some computing on it; the GUI front-end gets information from the statistical server or the database server and provides the user with corresponding choices for plots and reports.

The first, and essential contribution of CORBA is to bring the actual programming environment closer to this reasonable view. Two components, such as the statistical (client) process and the database server, need only understand the data involved in terms of the essential structure, which each can understand in its own language. The structure and the available methods are declared in the Interface Definition Language. For example, all of the components in Figure 1 deal with wafer test data (except perhaps the GUI), so they need to agree on what information such data contains. The agreed-on answer just needs to be defined once, say as an IDL definition of the class, say `ProbeTest`.

```
typedef sequence<string> StringVariable;

interface ProbeTest {
    attribute StringVariable results;
    attribute string code;
};
```

This definition is one of many possible; it corresponds roughly to the example on page 46 of reference [1]. The test results are a vector of strings, one for each of the sites on each wafer in the lot. The string `code` identifies what kind of devices are being manufactured here, which in turn allows the software to find out other information for plotting. If this was agreed to be the useful way of dealing with the information, then the database software for the application would likely advertise a method, say:

```
ProbeTest getLot(in string lotId);
```

This takes a string identifying the particular wafer lot of interest, and returns a `ProbeTest` object. Similarly, a class with a `getLotIds` method provides a list of lot names stored in a given database to any client; this is the method that the Java GUI applet invokes as soon as the user selects a database (see Figure 2).

The helpful aspects of this approach are that this description can be automatically coerced into an object in Java, S, or other systems, and that the actual transmission of data implied between the components is transparent to the application programmer. The statistical system, acting as a client, invokes the method `getLot` (perhaps via C++ bindings for CORBA), just as if it were a local computation. The object reference returned would then be coerced into a corresponding object in the statistical system. Figure 2 shows the same general application as Figure 1, but this time using CORBA to supply the interactions. The arrows now indicate method invocation, and the actual programming is now much nearer to the simplicity suggested by the picture; in fact, the programming actually is *less* influenced by the details of the systems than the figure suggests. The communication of a request, for example, from the GUI appears as a local method invocation in the Java code, independent of the location or even the language of the server.

Now that the definition of the data and methods is based on a standard interface specification, it is no longer required any special coding for each pair of systems involved, and applications can invoke and respond to requests from *any* other application that supports the CORBA interfaces, independent of the client's and server's language implementation, or operating system they run on. For example, database software often supplies some summary calculations. These are likely to be less flexible than those of a good statistical system, but might be faster on large datasets, so why not invoke them directly

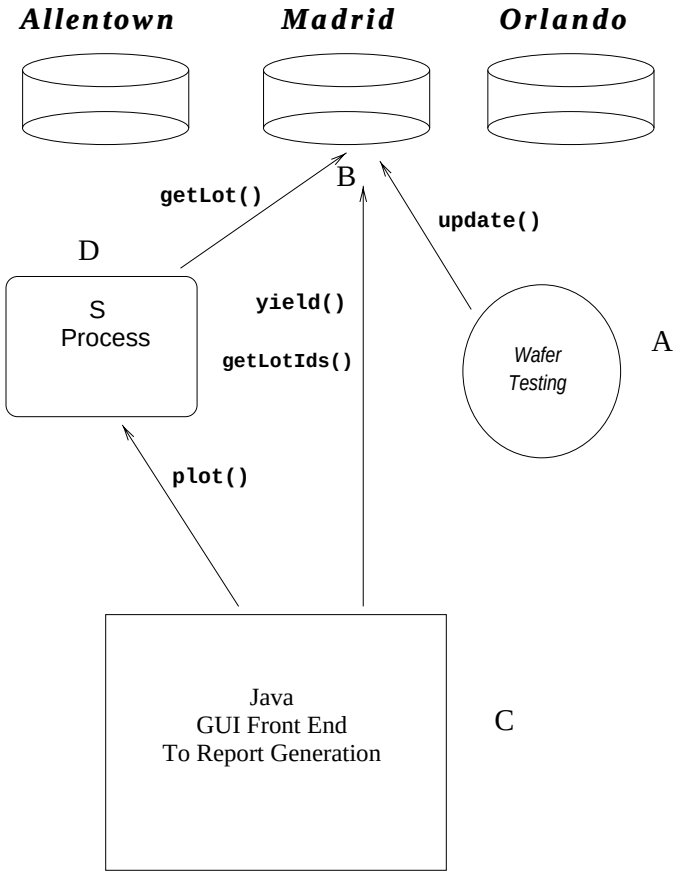

Figure 2: *Viewing test results, with CORBA providing the interaction. Arrows indicate invocation of the corresponding method, pointing towards the server providing the method.*

from the GUI? In Figure 2, the GUI invokes a `yield` method to get the average of successful tests; this method happens to be computed directly by the database interface (perhaps using JDBC, Java's Database Connectivity, to do the actual database access).

It needs to be emphasized that this scenario is a very mild, vanilla use of CORBA. It supposes a minimal up-front investment in developing higher-level tools, working directly from the standard facilities in CORBA. Even so, the communication requirements are drastically reduced, and the explicit definition of the classes and methods involved improves the clarity and understandability of the programming. In the next section, we contemplate some more interesting directions. A demonstration system along the lines discussed here was implemented at Bell Labs; the system is discussed on the Web, at `http://cm.bell-labs.com/stat/project/CorbaDemo`.

5 Future Work

Some initiatives in the area of statistical software that can contribute to more effective use of CORBA-based distributed computing are outlined briefly in this section. The development of commonly understood interfaces (classes) of data among different statistical systems will provide well-defined structure for distributed use of multiple systems. The definition of methods to deal with database software in terms of such classes of statistical data will give us a start on an improved connection between statistical and database software. Making CORBA programming directly available from statistical languages will simplify future programming with data in a distributed environment.

Besides these directions, all relatively specific to statistical software, we anticipate that new general programming tools can be created to raise the level of programming for distributed computing. We expect that future work with CORBA will benefit from the dynamic, self-describing data available for all CORBA software. Other languages and systems, such as the most recent versions of Java, and S, also provide dynamic access to metadata about the software in the language. Eventually, much of the detail of programming inter-system requests will be automated through the use of such information.

5.1 Statistical Data Definition

The Interface Definition Language provides a general mechanism for defining classes (interfaces, in CORBA terminology). These interface definitions can contain attributes (similar to slots in other languages) and operations (the signatures for methods defined on objects from the class), as well as exceptions for error handling.

Mutual understanding of some operations for a particular interface is needed for a client application to make a request of a server. The client has to provide a request in terms of an operation that the server has advertised, via an interface definition. The objects that appear as arguments to the operation or as the result of it are communicated between client and server, both of which must have enough understanding of the definition of the objects to make use of them.

For statistical systems (and other software) to offer services or to use each others services, they will need some basic interface (class) definitions on which to base communication. In the short term, one advantage of CORBA's implementation in languages such as C++ is that interfaces can be implemented for many systems (S being an example) by users; no modification of the core software is required. But much more desirable in the long term than *ad hoc* interface design would be agreement among owners of statistical systems or (even better) among professional statistical societies on a suitable set of base classes for the Statistical Data Definition.

5.2 Statistical Database Connectivity

The data used for statistical analysis increasingly derives, directly or indirectly, from information stored in database management systems, usually those built on the relational database model. Equally important is that much important data involved in science, government or industry with the *potential* for useful data analysis resides in such systems. One of the impediments to more effective statistical involvement with such data is the lack of an open, *general* interface between relational data and the typical models for observational data in statistics. As a result, extracting data from such systems is not seen sufficiently as a natural part of the cycle of statistical analysis, and such involvement as does exist lacks reusability when too tied to specific statistical or database systems.

The statistical model for observational data—in simple form that of observations on some number of variables for each of some set of observational units—has much in common with the relational model, in particular with the result of a query in the standard query language for such databases. As mentioned in Section 3, there are some intrinsically hard problems in matching the models perfectly. However, there is also a very great potential benefit for a simple, general, and extensible way of relating the common kinds of data in databases to statistical data classes, and by providing communication mechanisms in the statistical systems to select and modify data in standard database terminology, transparently to the user.

Standards exist on the database side to build on: the structured query language (SQL) for formulating queries to any conforming database system, and the JDBC as a programming interface for implementing queries and dealing with the results. By combining JDBC with CORBA and suitable statistical data definitions in IDL, we intend to provide a general base for communication between statistical software and databases. An encouraging feature is that some extremely simple interface and method definitions would be a sufficient base to get things going.

An interface for statistical data compatible with the general relational model for databases will be most effective if it, too, can become a standard accepted by statistical and computing groups. As a step towards this goal, we will examine in our research on distributed computing some simple versions of such an interface, and present them to our colleagues for comment.

5.3 CORBA and Statistical Systems

Presently, with some effort, users of a statistical system can interact with existing CORBA server objects and even provide objects within the statistical environment as servers. This is done using the usual CORBA approach of generating IDL descriptions of the server and compiling C++ or Java code to provide access to the CORBA facilities. Finally, operations are developed in the statistical system to interface to the C++ or Java code.

The prospects for integrating statistical software via CORBA, however, go potentially much farther. We believe that statistical systems can easily provide builtin CORBA support to insulate their users from such low-level CORBA programming. This is made possible by the dynamic, object-based nature of CORBA, which allows systems to discover and interpret interface definitions in IDL. When operating as a client, the statistical system can dynamically translate a call in the statistical system's language to a remote method invocation on a CORBA object. Similarly, existing data objects and/or methods in a given statistical environment can be automatically made to act as CORBA servers. The system need only translate the description of the data structures into IDL to make this possible. This and translation between CORBA and each statistical system's objects can be performed in a recursive manner by unraveling the recursive IDL description of the objects.

As an example, this facility, combined with the statistical database connectivity above, will provide a rich environment for programming seamless database access from a statistical language. In many practical applications with large datasets, such programming will eventually allow the evaluator manager in the statistical language to do key translations of user requests to, for example, apply certain computations of selection, sorting, and summary during the database query rather than later in the statistical system.

5.4 Visualization

In addition to statistical languages and database management, nearly all the examples similar to those shown here involve interactive graphics or visualization of data. We will explore existing software suitable to be used in the IDL-style approach to distributed computing, ideally collaborating with the developers of these packages. We hope to find that existing graphical tools are accessible to integration into this framework.

6 Conclusions

The distribution of computing services, geographically over the Web and computationally over different languages and operating systems, presents both an opportunity and a challenge for statistical software. We have proposed a CORBA-based approach for distributed computing with data, an approach that is neutral to programming languages (implementable in C, C++, Java, etc), operating system (Unix, Linux, Windows 95/98/NT, OpenVMS, MacOS, OS/2) computer architecture, and network infrastructure. Because of its

formal descriptive definition of data structure and because of the dynamic, self-defining nature of objects on the network, CORBA offers the potential for a high-level interoperability among systems, and a powerful distributed programming environment for users.

As one step in realizing this approach, we hope to create, together with the statistics and scientific communities, a Statistical Data Definition standard that distributed applications can rely upon for communication. We have also outlined some of our future efforts in the areas of database connectivity, statistical system use of CORBA, and visualization.

References

[1] John M. Chambers. *Programming with Data: A Guide to the S Language*. Springer-Verlag, 1998.

[2] Lorraine Denby and David A. James. A graphical user interface for spatial data analysis in integrated circuit manufacturing. In Michael Meyer and James Resenberg, editors, *Computing Science and Statistics*, volume 27. Interface Foundation of North America, 1995.

[3] Mark Hansen and David A. James. A computing environment for spatial data analysis in the microelectronics industry. *Bell Labs Techn. J.*, 1997.

[4] John Chambers, Mark Hansen, and Jonathan Mattingly. S and Java: Experimenting with data analysis on the web. *Bell Labs Research Memorandum*, 1998. Available on the web at: `http://cm.bell-labs.com/stat/doc/SJava.ps`.

[5] Alan Pope. *The CORBA Reference Guide*. Addison-Wesley, 1998.

[6] Jon Siegel. *CORBA Fundamentals and Programming*. Wiley, 1996.