Designing A Distributed Information System for Collaborative Statistical Computing*

Oliver Günther† Rudolf Müller† Peter Schmidt† Hemant Bhargava‡
Ramayya Krishnan§

Abstract

The World Wide Web has been extremely successful as a tool for distributed publishing and sharing of information among large dispersed groups. This raises the question whether the distributed authoring and execution of *software modules* can be supported in a similar manner. We study this problem by first developing the requirements of a group of developers and users of statistical software at a German national research laboratory. We then propose a conceptual framework to guide the design of an information system in order to meet these requirements and report on MMM, a prototype implementation.

1 Introduction

The World Wide Web (WWW) [BCL+94], a distributed information system on the Internet, is a successful example of the use of information technology to facilitate authoring and sharing of hypermedia information among large dispersed groups. It has not only brought millions of new users to the Internet, but also made it easy and desirable to publish information content for dissemination to this worldwide audience. This success of the Web, as well as the fact that it is mostly limited to the dissemination of *static* information, raises the question whether the distributed authoring and execution of *computational software modules* can be supported in a similar manner. This paper studies this problem

- by examining the requirements of a group of researchers who need to collaborate in the implementation and use of software modules and packages for statistical data analysis at the German National Research Center on the Quantification and Simulation of Economic Processes (SFB 373),

*This research has been supported by the Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 373.
†Institut für Wirtschaftsinformatik, Humboldt-Universität zu Berlin, Spandauer Str. 1, 10178 Berlin, Germany, {guenther,rmueller,pschmidt}@wiwi.hu-berlin.de
‡Naval Postgraduate School, Monterey, CA 93943, USA, bhargava@cs.nps.navy.mil
§The Heinz School of Public Policy and Management, Carnegie Mellon University, Pittsburgh, PA 15215, USA, rk2x@cmu.edu

- by developing a conceptual framework to guide the implementation of an information system designed to meet these requirements, and
- by describing a prototype system, MMM, which implements the conceptual framework as middleware services that facilitate WWW-based interaction between users and developers of statistical software at the center.

The rest of the paper is organized as follows. In Section 2, we present an illustrative scenario of interaction with statistical methods. We use this scenario to motivate the requirements for a distributed information system that supports collaboration in statistical computing. In Section 3, we present a conceputal framework for such a system. In Section 4, we briefly describe the MMM system that implements the conceptual model as middleware services using the Ypsilon software environment [MM94].

2 Collaboration in Statistical Computing

This section presents a scenario, set in the German National Research Center SFB 373, to illustrate collaborative statistical computing. The scenario is used to motivate the requirements for an information system that can facilitate this type of resource sharing in a large dispersed group.

2.1 Background

The research center SFB 373 conducts research on the development, adaption, and application of statistical methods for empirical economics. Its members are mathematicians, statisticians, econometricians, economists, and computer scientists. One of the major objectives of the center is to promote interdisciplinary projects.

Collaboration in this setting entails both publishing one's own statistical methods, i.e., making them available to other team members, and using methods developed by other teams. A key objective of these statistical methods is data analysis: Given a set of observations, how to find an appropriate statistical model (e.g., a linear autoregressive process), and to fit it to the given data set. Interaction with such a method is typically performed in a three-step loop: (i) choose the model, (ii) estimate the model parameters, and (iii) visualize results. Statistical methods are implemented using either specialized scientific computing software, such as Mathematica, Matlab, Gauss, and SAS, or software developed in-house at the center, such as XploRe [HKT95]. Reflecting the corresponding programming paradigm, the software modules are often referred to as *scripts*.

While each of these statistical software packages supports this kind of data analysis, the heterogeneity in data formats, scripting languages, etc. presents challenges when models implemented in different systems have to be used together. This is a serious problem in a collaborative laboratory environment where different "traditions" of doing scientific computing need not only to co*exist* but to co*operate*.

```
link=Install["sqlmath"]
sybaseconnect["MMM","Password","SYBASE","boerse"]
ts=Transpose[List[First[Transpose[sybaseSQL["select deutscherendite, jahr,
monat from bank en1 order by jahr, monat"]]]]]
sybasedisconnect[]
```

Figure 1: A Mathematica script that encapsulates a query to a relational database system

2.2 A Case Study

The objective of the following short case study is to motivate the specific requirements for a system that supports the distributed publishing and execution of statistical methods. The case study is based on three scripts written by three different teams in the center. The scripts have been implemented using three different software packages: Mathematica, Matlab, and XploRe. They had to be tied together to perform the following complex task:

1. Given a relational database that contains monthly return on capital for major German companies, find the monthly return on capital of Deutsche Bank between 1957 and 1991. We assume that a Mathematica-SQL connection has been implemented, such that the query can be performed using the Mathematica method shown in Fig. 1.

2. Compute the time series of estimated residuals of the time series of the Deutsche Bank returns. This estimation uses the Matlab implementation of a function that estimates a feed-forward neural network using Levenberg-Marquardt Approximation (see [LT96]). Fig. 2 lists parts of the Matlab script.

3. Based on the result of the last computation, perform an analysis of the conditional second moments using GARCH models. An XploRe macro is available to perform this computation (Fig. 3).

What kind of computing infrastructure does this require? All three functionalities require access to computers with the appropriate software environment preinstalled. The modules shown in Fig. 1–3 have to be in place on these machines, including all functions that are referenced directly or indirectly (e.g., the function `sybaseconnect` in the Mathematica script). In addition, variables that need to be initialized before execution need to be identified and initialized appropriately. For example, before the Matlab script from Fig. 2 is executed, the variable `xraw` has to be initialized with the time series. The Matlab script generates multiple outputs. Of these, only `err_o` containing the estimated residuals after calling the Levenberg-Marquardt approximation is used as input to the XploRe script. Identifying those variables that are used in this manner to link scripts is important since their values need to be stored before the script execution terminates — for example, by adding commands that write their values into a file. In addition to saving values, data format conversion may be required. For example, the result of the SQL query (Fig. 1) is a Mathematica vector (variable `ts`), which has to be converted into a

```
% parameters used for Levenberg-Marquardt Approximation
min_grad  = 0.00001;
min_grad  = 0.00001;
mu        = 0.01;
mu_inc    = 10;
mu_dec    = 0.1;

xraw  = log(xraw);

[Ztrain,wnnew_o,epochsmin_o,wncvnew_o,n_par_o,sc_o,aic_o,hq_o,Atrain_o,
errdat_o,Acv_o, errcvdat_o,W1_0,b1_0,W2_0,b2_0,W1_1,b1_1,W2_1,b2_1,W1_2,
b1_2,W2_2,b2_2,W1_3,b1_3,W2_3,b2_3,W1_4,b1_4,W2_4,b2_4,W1_5,b1_5,W2_5,b2_5]
= nnselc(lags,xraw,S1max,trans,disp_freq,max_epoch,epochsec,sin_epoch,
err_goal,lr,initcmax,Ztrain,min_grad,mu,mu_inc,mu_dec);

err_o   = [errdat_o' errcvdat_o']';   % estimated residuals for total data set
```

Figure 2: This Matlab routine initializes several parameters and calls the function `nnselc`, which implements a Levenberg-Marquardt approximation.

format that is readable by Matlab. If the machines running, say, Mathematica and Matlab do not share a common file system, the data has to be moved within the network, e.g. by using FTP.

2.3 Requirements

Two trivial solutions come to mind in order to overcome the heterogeneity problems discussed above. First, authors could be required to make their modules available in multiple formats. Second, users could be required to translate the required programs such that they can be executed in their local hard- and software environment. Both alternatives are cumbersome and time-consuming. All this effort could be avoided if there were a system in which researchers can *publish* their modules in their original form (i.e, the one in which they had implemented them for their own purposes), and users can *use* them as if they were operating on a personal desktop-based environment. This vision led to the design of the MMM system based on the following set of requirements.

Execution services: For each statistical computing package at least one *execution service* has to be made available. Each such service has to be accessible by all members of the research center and by authorized outside users. Execution services have to be extensible by authors, i.e., authors can make available libraries with modules, which can then be used in other modules by other authors. The challenge of realizing execution services is to make

```
; This function estimates the function of conditional second
; moments of a univariate time-series, using GARCH models.
; The output is a matrix d

r = r'
n = rows(r)
t = 1957.5 + (0:n-1)/12
n = rows(r)
library("mmmlib")
d = archest (r, 1, 1)
xt = var(r)|d[2,1]*r[1:n-2]^2
st = genar (xt, d[#(1,3),1], n-1)
```

Figure 3: This XploRe macro implements a GARCH model by calling XploRe standard functions.

the specifics of interacting with different packages transparent. For example, the different ways of executing a module within a package, such as writing a batch job, or entering a command in a line-oriented command interface, have to be hidden. Independently of the package specifics, the interface of the execution service should consist of three principal steps: specify input, start execution, and access output.

Interface definition: Modules that are authored to be executable in this manner need interfaces that specify which input variables have to be provided, how they should be invoked, and which results are available when execution is complete. This presents a major challenge, since statistical methods are typically authored in declarative, interpreted languages rather than imperative, object-oriented languages (such as C++), which greatly facilitate interface definition.

Interoperability: As demonstrated in the case study, this is where users lose most of their time. Assistance is particularly necessary for (i) data transport and format conversion between all statistical software systems involved, (ii) system-specific combination of data and methods, and (iii) invocation and execution control across the network.

State maintenance: In general, statistical analysis is exploratory with multiple passes on the methods and parameters. In an interoperable setting, a user has to keep track of inputs to methods and intermediate results. Assistance needs to be provided in the form of *state maintenance services*, which store the inputs and results of intermediate computations during a session. This facilitates exploratory work with multiple methods.

Scripting: Assuming that the infrastructure is set up in a way that a single user has access to all distributed resources, this user might want to automate the execution of a computational plan by bundling together all the steps (e.g., the three operations in the case study) into a single executable script. This script could then even be authored as a "new" service.

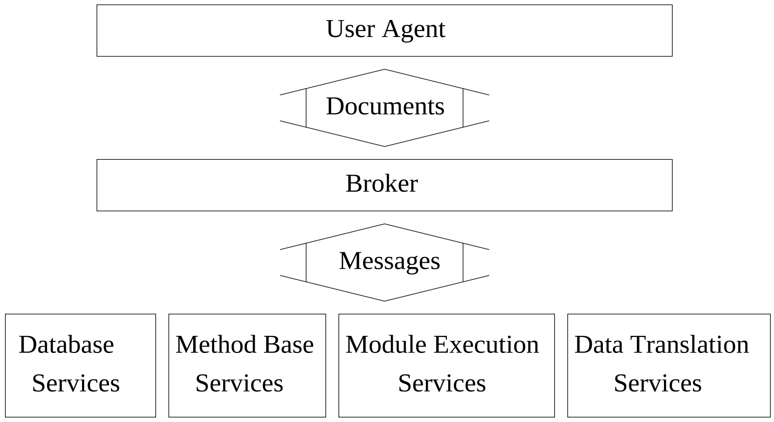

Figure 4: User Agent, Broker and Services

Internet/WWW: From a pragmatic perspective, the information system for authoring and sharing of statistical methods should have wide reach. WWW-based interfaces offer such a wide reach on multiple platforms. This cross-platform support is needed in a center such as the SFB where platforms and operating systems proliferate. Thus, interfaces for authoring and using methods should be based on Web technology.

Possible additions to this list of requirements include a meta-information system that supports users in the appropriate usage and information retrieval, or visualization and interactive graphics even when working with remote services. These features are subject to future work; they will not be discussed further in this article.

3 Conceptual Framework

Based on the requirements specified above, this section presents a conceptual framework for an information system to perform collaborative statistical computing. The principal components of our system are agents responsible for different types of services. Users of statistical modules communicate with the system via a *user agent*. This agent connects them to a *brokerage agent* or *broker*, which mediates transactions between several types of service agents and the user (see Fig. 4). We first describe the functionality of service agents, then the user agent, and finally the role of the broker.

3.1 Database Services

The analysis of data sets is fundamental to statistical computing. Database services maintain data sets for the users of the system. In statistical computing, the structure of data sets is mostly restricted to real-valued vectors and matrices. Only recent developments in statistical packages add the concept of records including alphanumeric data, such as names and dates

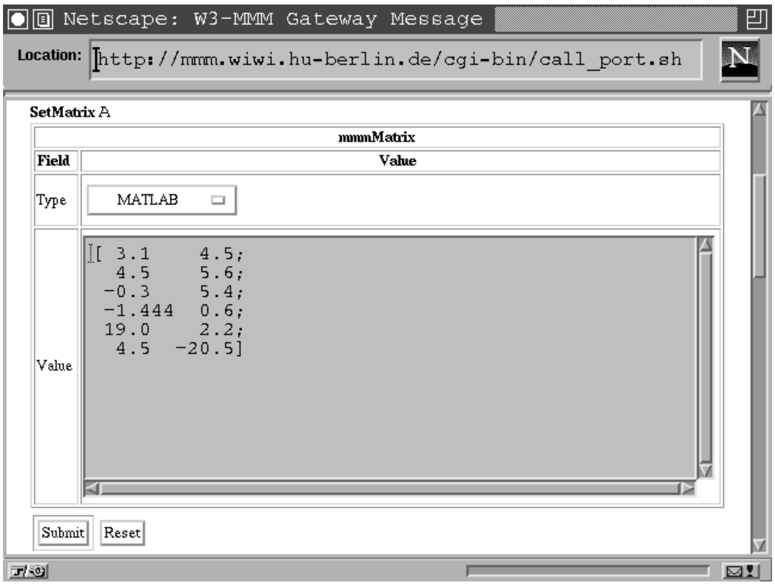

Figure 5: An HTML form to edit a matrix with the MMM user agent. The select button specifies the language. The matrix is in Matlab format, a language-specific ASCII representation.

of observations [SKKH96]. Given the declarative, interpreted nature of statistical computing languages, data sets are usually represented as formatted strings containing the numerical values in ASCII notation. Details of the format (separators, use of parentheses, etc.) vary between systems.

The interface of a database service provides operations to create, update, access, and delete data sets. A database service treats data sets as black boxes in the sense that it does not provide operations to manipulate, for instance, parts of a matrix. Such tasks have to be performed by execution services using suitable modules. Also, there are no operations specific to the format in which the data is expressed as a character string. Necessary conversions of the format are delegated to special *data translation services* (see Section 3.2).

Apart from the black box conceptualization of a data set, each data set in a database service is combined with slots for meta-information. The simplest, but most important example of such meta-information concerns the format of the data set; see Fig. 5 for an example. Further information could be added, depending on the particular application in mind.

Database services should also be able to integrate data sets from other sources. This includes simple remote access via standard protocols, such as FTP and HTTP, but also the more complex execution of database queries, possibly followed by some format translation. Alternatively, data integration could be done with the help of an execution service, as illustrated by the first module in the case study.

3.2 Data Translation Services

Data translation services perform the required conversions between different data formats. Typical formats include character string representations or relational representations coming in from the database services, as well as the package-specific "statistical" formats. Conversion has to be supported between all formats required by the module execution services of the system.

3.3 Method Base Services

In analogy to database services, method base services treat modules as black boxes. Inside those boxes are character strings that represent the implementation of some method. A method base service supports similar operations as a database service: It provides an interface to add, delete and change modules, and it provides an interface for an execution service to access a module.

As with data sets, an instance of a module provides slots for meta-information. Again, the language of a module is an important piece of meta-information. But more is required. Usage of a module requires that it can be moved to an execution service and combined with data from the database service. Results have to be moved to the database service after the execution is completed. Both operations require that meta-information about input and output of a module be provided. An analysis of several packages showed that a list of names of input and output variables, i.e., a *signature*, suffices to perform these operations (see Section 3.4). In addition, meta-information about a module may have to list other modules from the method base that have to be installed at the execution service prior to execution.

3.4 Module Execution Services

Module execution services encapsulate the execution of statistical methods by implementing connections to execution engines for the supported packages. This is done in three phases: First, the service accesses a module from a method base service and data from a database service. It concatenates data and module in a way specific to the package. Second, it instructs the engine to execute the module. Third, when execution is complete, it obtains the output data from the engine. This last step may require the addition of package-specific commands to the module in the first phase. Finally, outputs are forwarded to the database service.

The details of implementing the three phases vary considerably from package to package. In Mathematica, for example, a library can be used to create a stateful link to a Mathematica engine using a C program. With XploRe, on the other hand, no such support exists, and the implementation uses a stateless connection to an XploRe server listening on a telnet port on a UNIX machine. In the first case, variables can be initialized in a stepwise manner by appropriate function calls. In the second case, the XploRe module has to be combined with the data by string concatenation.

As mentioned in Section 3.3, the phases have to be supported by meta-information that is provided by the author when submitting a module to the method base service. Thus, names of input and output variables have to be listed. Furthermore, the module implementation must not contain any initializations of input variables, nor instructions that store results in files. Corresponding commands are generated by the execution service and depend on the way the package is connected with the service.

3.5 User Agent

Our design assumes an interaction with the system that is based on the exchange of documents, such as an HTML form (Fig. 5). Document content, such as HTML form variables, is translated into a *message*. In case of using a Web browser and HTML forms, form submission results in the invocation of a CGI application that communicates with the broker using a system-specific protocol. In this scenario, the Web browser and the CGI application together make up the user agent. Alternatively, the user agent could consist of a series of downloadable Java applets or Active-X applications.

3.6 Broker

Services cooperate with each other in order to allow providers to author modules and data sets and to allow consumers to use them. Such requests are formulated by the user agent. The *broker* mediates between the user agent and the other services. On request from a user, the broker gathers information about the operations that could be performed by the services. It translates this information into a document which is sent as response to the user interface (usually a Web browser). The user uses this document to select the next operation (see Fig. 6). Again, the broker could be a Java applet running inside the Web browser of the user, thus combining the user agent and the broker in one application.

4 The Implementation of MMM

We have presented a conceptual framework for a system to publish software modules for remote access. This section gives a brief overview of MMM, a system that implements this design by combining Web technology with distributed object technology. Web technology is used to implement user agents and to integrate resources like data repositories and computational services on telnet ports. Distributed objects are used to realize agents that provide services for authoring and executing statistical methods, and for the maintenance and format conversion of data sets. For a more detailed description of the MMM implementation see [GMS+96].

We first show how the Ypsilon system [MM94], a C++-environment for model-based software development, has been applied successfully to implement the conceptual framework. After giving a short overview of Ypsilon, we illustrate how we used this environment to implement a generic client/server topology for the exchange of formatted messages over TCP/IP. Finally, we describe the implementation of the agents and their communication protocol.

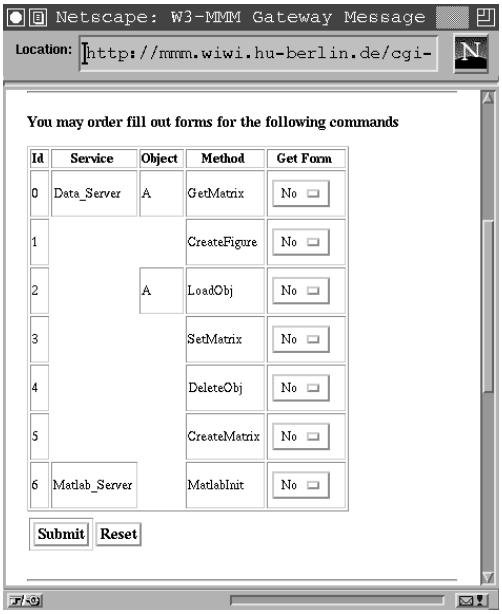

Figure 6: An HTML form to select the appropriate form for the next operation at an MMM service.

4.1 The Software Development System: Ypsilon

Ypsilon was developed as a toolbox for model-based software development. It provides an infrastructure to encapsulate services into objects implemented in C++. The original focus of Ypsilon was the implementation of libraries with efficient algorithms for project scheduling [MS95]. Ypsilon objects inherit a rich set of functionalities from Ypsilon core libraries, including graphical user interfaces, representations of instances as ASCII streams, and memory handling. This makes it a useful toolbox for rapid prototyping of C++ software systems. The system consists of

1. a generator that produces C++ class implementations from (class) models;
2. compiled C++ and X11/Motif libraries that provide generated classes with programming and user interfaces;
3. a generic runtime environment to instantiate and work with instances of Ypsilon classes.

Each Ypsilon instance has a representation as an ASCII character string that contains the complete type information. Together with the instance value, this information can be used to define the protocol between MMM services. Messages are modeled as Ypsilon classes. Sending a message translates to sending an instance of an Ypsilon class. If the same channel is used to send messages of different types, i.e., instances of different Ypsilon classes, the type information defines the type of the message. This enables the implementation of generic communication services. Jeusfeld and Bui [JB95] have proposed a similar type of data representation as a basis for interoperable decision support system components on the Internet.

Ypsilon *function models* are all those Ypsilon models (i.e., classes) which have fields `input` and `output`. A function model encapsulates an (external) function call as a C++ class. The typical usage of a function model in a C++ program is illustrated below, where `F` is a C++ class implementing a function model:

```
...
F f;
f.input() = a;
f.evaluate();
b = f.output();
...
```

The implementation of the member function `evaluate` encapsulates not only the call of the external function, but also format conversion from and to the external function's data structures. Furthermore, function models implement a standardized interface for function calls that can be used in a simple manner by other applications.

4.2 The Communication: MMM Client and Server Models

While Ypsilon function models encapsulate services, the two Ypsilon models `MmmFClient` and `MmmFServer` encapsulate the communication of services. They implement the protocol layer of MMM by using functionalities of the ACE library [Sch94]. This library contains C++ wrapper classes for interprocess communication; it runs on a broad variety of operating systems.

`MmmFClient` is a function model that encapsulates the communication with servers via Internet stream sockets. As a function model, it contains the fields `input` and `output`. Both fields are of type `GCData`, a wrapper class whose objects can be initialized dynamically with all other Ypsilon models. Together with the typed ASCII representation, this allows a generic handling of Ypsilon model instances. `MmmFClient` also has a field `server`, a record that gets initialized with a DNS name (or an IP address) and a port number. The member function `evaluate` of `MmmFClient` opens a stream socket connection to the address specified in the field `server`, takes the `GCData` object from the field `input`, and sends it to this address. The received reply is assigned to the field `output`, followed by closing the connection to the remote address.

The model `MmmFServer` contains a field `port` and a field `service`. Its task is to receive messages from `MmmFClient` objects, process the message with the field `service`, and send a reply back to the client.

Setting up a MMM service requires the implementation of an Ypsilon function model and a program that initializes the `service` component of an `MmmFServer` object with an object of the function model. Calling such a service out of a C++ program requires the initialization of an `MmmFClient` object, the initialization of its `input` with the request and the receipt of the reply from `output`.

4.3 Agents and Messages: MMM Service Models

The last section showed how services are embedded in the server model `MmmFServer`, which implements their communication. A service thus waits for an instance of a specific *message model*, processes the request encoded in that instance, and generates as result an instance of a reply model. MMM services implement all types of services discussed in Section 3.

Each service accepts a set of message models, where different messages initiate different operations at the service. These operations may change the state of the service. Certainly, it depends on the state of an object, whether some operation makes sense or not. In other words, the state of an object defines whether a message is *acceptable* [BKM95b]. For example, at the beginning of each interaction with a service, an authentication operation has to be performed. Only after valid authentication, other messages become acceptable. For example, browse meta-information, a message to the method base, becomes acceptable only after authentication.

The MMM broker implements middleware between user agent and services. It is a stateful service, whose state consists essentially of information about the interface of connected services. The connection between Web browser and broker is established by the user agent. Depending on the message from the user, the broker consults one or several services to process the user request. Each consultation is performed by using one of the acceptable messages of that service. The result from the service is a reply message that is sent back to the user agent. In addition, the broker receives from the service an update of the set of acceptable messages for that service. As a part of the reply to the user, the broker offers an HTML form by which the user may choose one of the acceptable messages of any of the services (Fig. 6).

5 Discussion

In this paper we proposed a conceptual framework for collaborative statistical computing. We also described the MMM system, a prototype implementation to make software modules accessible on the World Wide Web. The key features of MMM are:

- It implements stateful services, motivated by the exploratory nature of statistical data analysis.
- By using middleware services, it enables interoperability between proprietary statistical computing packages.
- It provides publishing support by helping authors in interface definition.

The combination of these features distinguishes MMM from most other work that is concerned with the collaborative usage of software modules in distributed environments. The first milestone towards enabling access to computational services on the Web was the Common Gateway Interface (CGI) [McC94]. CGI defines a standard of passing data from a Web browser to an application program. While thousands of CGI applications are now available on the Web (including many with scientific software), each of them provides a singular solution. There is no support to connect several such services into a pipeline, as is required, for example, in the case study presented above. Java applets also lack comfortable support for this kind of interoperability. Due to security considerations, browsers prevent Java applets that were downloaded from some site A to download other applets from some other site B. While this constraint may disappear in time, as the related security problems can be solved in a more sophisticated manner, Java has another major disadvantage: It requires the reimplementation of software already available. Switching to an object-oriented, imperative programming language like Java, however, is simply not a valid alternative for our typical users, who make considerable use of the mathematical expressiveness and richness of mathematical libraries in scripting languages like Mathematica or Matlab.

There is considerable overlap between the MMM concepts and middleware architectures [Ber96], in particular concerning the idea of the object request broker [OHE96]. While these technologies tend to be strong in providing reliable services in distributed computing (e.g., by enforcing transaction management), they do not emphasize support for publishing as much as MMM does. Interface definition languages support only imperative, object-oriented languages (e.g. C ++). So it may be a while before the *intergalactic object bus* [OHE96] will stop at our type of statistical computing services. However, we expect that parts of the MMM functionality that are currently implemented in Ypsilon can soon be replaced by ORB implementations following standards like CORBA or COM/OLE. These implementations were simply not available at the time when MMM started.

Another enhanced system for Internet access to software modules has recently been presented by Becker [Bec96]. Other than MMM, however, the system follows a functional design. Values are filtered through stateless services that encapsulate solvers for combinatorial optimization algorithms. The system lets a user state computational plans in the system's own scripting language. From the electronic commerce point of view, Bhargava et al. have surveyed emerging electronic markets for accessing software modules [BKM96a] and investigated business transactions [BKM96b]. We believe that the results of the MMM project could guide the creation of a new generation of electronic markets for statistical methods, as they are currently investigated for decision support technologies [BKM95a, BKC+96].

References

[BCL+94] T. Berners-Lee, R. Cailliau, A. Luotonen, H.F. Nielsen, and A. Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, 1994.

[Bec96] P. Becker. An embeddable and extendable language for large-scale programming on the Internet. In *Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS'96)*, 1996.

[Ber96] P. Bernstein. Middleware: A model for distributed system services. *Communications of the ACM*, 39(2):86–98, 1996.

[BKC+96] H.K. Bhargava, R. Krishnan, M. Casey, D. Kaplan, S. Roehrig, and R. Müller. Model management in electronic markets for decision technologies: A software agent approach. SFB Discussion Paper, Sonderforschungsbereich 373, Humboldt-Universität zu Berlin, 1996.

[BKM95a] H.K. Bhargava, A.S. King, and D.S. McQuay. DecisionNet: An architecture for modeling and decision support over the World Wide Web. In Tung X. Bui, editor, *Proceedings of the Third International Society for Decision Support Systems Conference, Vol. II*, pages 541–550, Hong Kong, 1995. International Society for DSS.

[BKM95b] H.K. Bhargava, R. Krishnan, and R. Müller. On parameterized transaction models for agents in electronic markets for decision technologies. In Sudha Ram and M. Jarke, editors, *Proceedings of the Fifth Workshop on Information Technologies and Systems, Amsterdam, Holland, December 1995*, 1995.

[BKM96a] H.K. Bhargava, R. Krishnan, and R. Müller. Decision support on demand: Emerging electronic markets for decision technologies. *Decision Support Systems*, 1996. to appear.

[BKM96b] H.K. Bhargava, R. Krishnan, and R. Müller. Electronic commerce in decision technologies: A business cycle analysis. SFB Discussion Paper, Sonderforschungsbereich 373, Humboldt-Universität zu Berlin, 1996.

[GMS+96] O. Günther, R. Müller, P. Schmidt, H. Bhargava, and R. Krishnan. MMM: A WWW-based model management system for using software modules remotely. Discussion paper 32-1996, Sonderforschungsbereich 373, Humboldt-Universität zu Berlin, 1996.

[HKT95] W. Härdle, S. Klinke, and B. A. Turlach, editors. *XploRe: An interactive statistical computing environment.* Springer-Verlag, Berlin, 1995.

[JB95] M. Jeusfeld and Tung Bui. Interoperable decision support system components on the Internet. In Sudha Ram and M. Jarke, editors, *Proceedings of the Fifth Workshop on Information Technologies and Systems, Amsterdam, Holland, December 1995*, pages 56–67. RWTH Aachen, Fachgruppe Informatik, 1995.

[LT96] H. Lütkepohl and R. Tschernig. Nichtparametrische Verfahren zur Analyse und Prognose von Finanzmarktdaten. In G. Bol and G. Nakhaeizadeh und K.-H. Vollmer, editors, *Finanzmarktanalyse und -prognose mit innovativen quantitativen Verfahren.* Physica-Verlag, Heidelberg, 1996.

[McC94] Rob McCool. *The Common Gateway Interface.* http://hoohoo.ncsa.uiuc.edu/-cgi/overview.html, 1994.

[MM94] D. Möller and R. Müller. A concept for the representation of data and algorithms. In N. Dean and G. Shannon, editors, *Computational Support for Discrete Mathematics, DIMACS Workshop March 12-14, 1992.* AMS, 1994.

[MS95] R. Müller and D. Solte. How to make OR results available: a proposal for project scheduling. In W. Gaul, F. J. Radermacher, and D. Solte, editors, *Data, Expert Knowledge and Decisions*, volume 55 of *Annals of Operations Research*, pages 439 – 452. J.C. Baltzer Science Publishers, 1995.

[OHE96] R. Orfali, D. Harkey, and J. Edwards. *The Essential Distributed Objects Survival Guide*. John Wiley & Sons, Inc., New York, 1996.

[Sch94] D. C. Schmidt. The ADAPTIVE Communication Environment: Object-Oriented Network Programming Components for Developing Client/Server Applications. In *Proceedings of the* 12^{th} *Annual Sun Users Group Conference*, pages 214–225, San Francisco, CA, June 1994. SUG.

[SKKH96] Swetlana Schmelzer, Thomas Kötter, Sigbert Klinke, and Wolfgang Härdle. A new generation of a statistical computing environment on the net. In Albert Prat, editor, *Proceedings in Computational Statistics: 12th COMPSTAT Symposium held in Barcelona, Spain, 1996*, pages 135–148. Physica-Verlag, 1996.