MMM: A WWW-Based Method Management System for Using Software Modules Remotely*

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

Abstract

The World Wide Web has been highly successful as a tool for the distributed publishing and sharing of online documents 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 an information system design that meets these requirements and report on MMM, a prototype implementation.

1 Introduction

The World Wide Web (WWW) [BCL+94], a distributed hypermedia information system on the Internet, demonstrates successfully how current technology can support the sharing of information among large dispersed groups. It not only brought millions of new users to the Internet, but also made it easy and desirable to publish online documents for dissemination to this worldwide audience. This success of the Web, combined with 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, SFB 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 proposing an information system design to meet these requirements, and
- by describing the implementation of a prototype.

The resulting system, called MMM (Method ManageMent system), is a collection of middleware services [Ber96] that facilitate WWW-based interaction between software users and developers. While statistical computing has been our main application focus so far, the basic technologies presented in this paper apply to the distributed authoring and utilization of software modules in general.

The rest of the paper is organized as follows. In Section 2, we present a typical scenario of interaction with statistical methods at the center. We use this scenario to motivate the requirements for a distributed information system that supports collaboration and resource sharing in similar environments. In Sections 3 and 4, respectively, we describe the design and realization of MMM. This includes a brief overview of the Ypsilon software development environment [MM94], which we based our implementation on. Section 5 surveys some related work and gives an outlook on future research.

2 Collaborative Statistical Computing

2.1 Background

The German National 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 methods is typically performed in a three-step loop: (i) choose a model, (ii) estimate the model parameters, and (iii) visualize results. Statistical methods are implemented using either specialized scientific computing software, such as Gauss, Mathematica, Matlab, SAS, S_Plus, and SPSS, or software developed in-house at the center, such as XploRe [HKT95]. In reflection of the corresponding programming paradigm, the software modules are often referred to as *scripts*.

While each of these statistical software packages supports the kind of data analysis described above, the heterogeneity in data formats, scripting languages, etc. presents a major challenge whenever models implemented in different systems have to be used in combination. This is a serious problem in a collaborative 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","stockxc"]
ts=Transpose[List[First[Transpose[sybaseSQL["select ret_cap, year, month
from ret_cap_month where company = "Deutsche Bank AG"
and year >= 1974 and year <= 1991 order by year, month"]]]]]
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 AG between 1974 and 1991. For security reasons there is no direct access to the database, but there is a Mathematica-SQL connection that allows the embedded execution of certain types of queries (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 [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 [Bol86]. An XploRe macro is available to perform this computation (Fig. 3).

What kind of computing infrastructure does all this require? On the one hand, the three software modules shown in Fig. 1–3 have to be executable on some set of networked computers. Before execution, all functions referenced directly or indirectly need to be made available, and variables may have to be initialized. The Mathematica script in Fig. 1, for example, requires access to the function `sybaseconnect`. Before the Matlab script in Fig. 2 can be executed, the variable `xraw` has to be initialized with an appropriate time series. The Matlab script generates several outputs, including `err_o`, which contains the estimated residuals after calling the Levenberg-Marquardt approximation. Only this variable is used as input to the XploRe script. Identifying such variables that are used to link scripts is important since their values need to be stored before the generating script terminates. Data format conversions may also be required. The result of the SQL query (Fig. 1), for example, is a Mathematica vector (variable `ts`), which has to be converted into a format that is readable by Matlab. If the machines running,

```
% 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.

say, Mathematica and Matlab do not share a common file system, the data has to be moved across 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 make their modules available in multiple formats and on multiple machines. Second, users could translate the required programs themselves such that they can be executed in their local hardware and software environments. Both alternatives are cumbersome and time-consuming. All this effort could be avoided if there were a system in which researchers can *publish* software modules in their original form (i.e, the one in which they had implemented them for their own purposes), and users can *execute* them as if they were operating on a personal desktop 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 there should be at least one *execution service* available that makes the specifics of interacting with other 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 support the three basic operations: specify input, start execution, and access output. Each execution service should be accessible by the members of the working group and

```
; 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.

by authorized outside users. Execution services have to be extensible, such that authors can make available libraries with modules, which can then be used in other modules by other authors.

Interface definition: Modules that are to be executable by such execution services 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. The system needs to support authors such that this information can be provided in a user-friendly manner. Generally speaking, it is crucial for the success of such an approach that new software modules can be checked into the system with reasonable effort. 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 would greatly facilitate the required interface definition.

Interoperability: As demonstrated in the case study, this is where users lose most of their time. Assistance is particularly desirable for (i) data transfer 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 most research environments, statistical analysis is exploratory, involving 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 inputs and results of intermediate computations during a session.

Scripting: Assuming that the infrastructure is set up in a way that a single user has access to a variety of distributed resources, this user may want to automate the execution of a

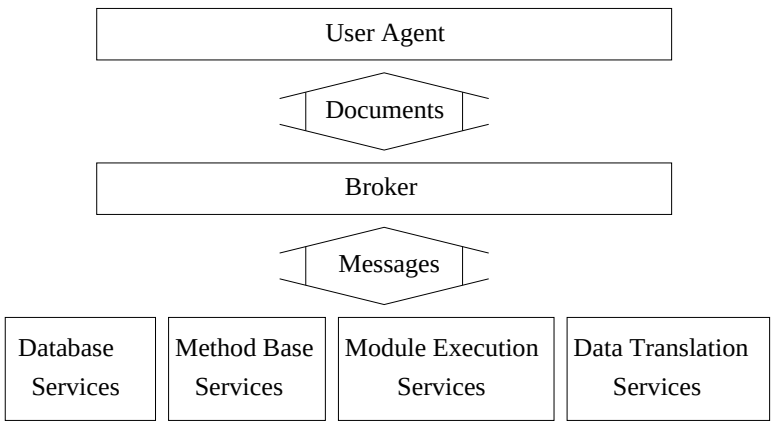

Figure 4: User Agent, Broker and Services

computational plan by combining all the steps (e.g., the three operations in the case study) into a single executable script (which could then be published in turn).

Internet/WWW: WWW-based interfaces reach a wide range of potential users on multiple hardware and software platforms. This kind of cross-platform support is needed in a center like the SFB where platforms and operating systems proliferate. Interfaces for authoring and using methods therefore need to be based on Web technology.

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

3 The Information System Design

Based on the requirements specified above, this section presents the design of an information system to support collaborative statistical computing. Section 3.1 describes the principal system components, and Section 3.2 discusses the extent to which our approach meets the stated requirements.

3.1 Principal Components

Our design is based on *agents* that are 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 (Fig. 4). We first describe the functionality of a variety of service agents, then the user agent, and finally the role of the broker.

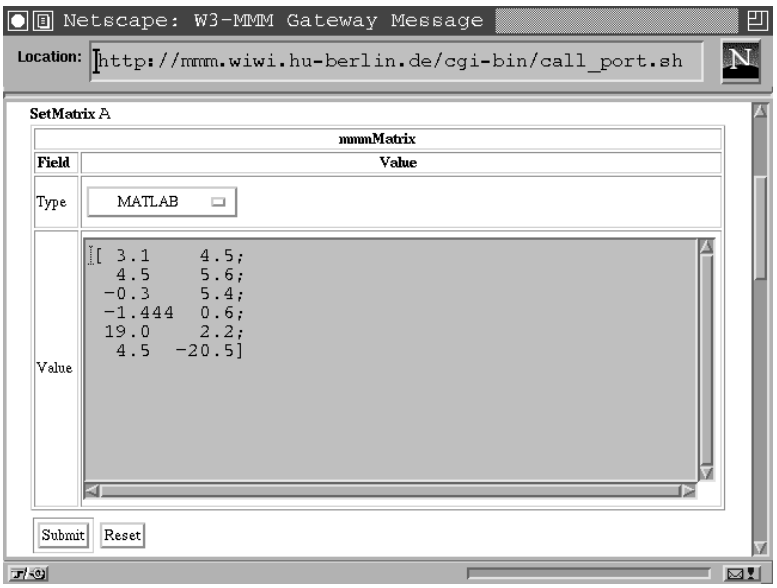

Figure 5: An HTML form to edit a matrix with the MMM user agent. The select button specifies the language. The matrix is in Matlab format, a language-specific ASCII representation.

3.1.1 Database Services

The analysis of data sets is fundamental to statistical computing. At this point, real-valued vectors and matrices are the most common data types by far. Alphanumeric data (e.g. for representing names or dates of observations) as well as complex types (such as records) are only gradually becoming more popular [SKKH96]. Data sets are usually represented and exchanged as formatted ASCII strings. Formatting details, such as separators or use of parentheses, vary between systems.

Database services maintain data sets for the users of the system. The interface of a database service provides operations to create, update, access, and delete a data set. A database service treats a data set as a black box. It would, for example, not provide any operations to manipulate parts of a matrix; such tasks have to be performed by execution services using suitable modules. There are also no operations that are specific to the data representation format. If format conversions are necessary, they are delegated to special *data translation services* (see Section 3.1.2). Notwithstanding this black box concept, data providers can enhance a data set by selected meta-information (such as the chosen format, see Fig. 5).

Database services are also able to integrate data sets from other sources. This includes simple remote access via standard protocols, such as FTP and HTTP, but also the direct SQL-based access to external relational databases, possibly followed by some format translation. Alternatively, data integration could be performed using an execution service, as illustrated by the first module in the case study.

3.1.2 Data Translation Services

Data translation services perform the required conversions between different data formats. Typical formats include relational representations produced by the database services or package-specific "statistical" formats. Conversion has to be supported between all formats required by the module execution services of the system (see Section 3.1.4). The interface of a data translation service is straightforward: A request sends a data set in some format together with the specification of a target format and obtains the converted data as a result.

3.1.3 Method Base Services

In analogy to the treatment of data sets by database services, method base services treat modules as black boxes. Inside those boxes are ASCII 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 a module, and it provides an interface for an execution service to access a module.

As with data sets, one can also store some meta-information with each module instance (Fig. 6). The language the module is written in is an obvious example, but other items may also be essential. One may, for example, want to provide meta-information about which other modules have to be installed at the execution service prior to execution. Moreover, in order to execute a module, one has to move it to an execution service and combine it with data from a database service. The results of the computation have to be returned to the database service after execution has completed. Both operations require meta-information about the input and output behavior of a module. An analysis of several packages showed that a list of names of input and output variables, a so-called *signature*, is usually sufficient.

3.1.4 Module Execution Services

Module execution services encapsulate the execution of statistical methods by implementing connections to selected execution services. 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 execution server to execute the module. Third, when execution is complete, it obtains the output data from the execution server. 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 to 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.

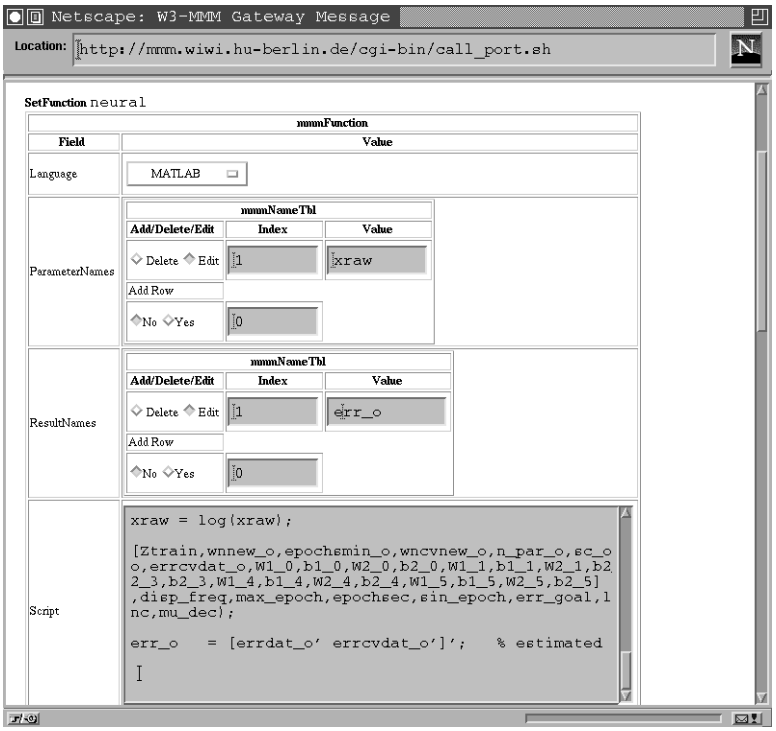

Figure 6: An HTML form to publish a module at a method base service. The top part specifies some meta-information (language and signature), the bottom part contains the script.

As mentioned above, the three phases have to be supported by meta-information that is provided by the author when submitting a module to the method base service. Names of input and output variables have to be listed. Moreover, 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.

3.1.5 User Agent

Our design assumes an interaction mode 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 *messages*. If one uses 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 case, the user agent simply consists of the Web browser and the CGI application. Alternatively, the user agent could consist of a series of downloadable Java applets or ActiveX applications.

3.1.6 Broker

Services cooperate with each other in order to allow providers to publish modules and data sets and to allow consumers to use them. The corresponding 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 currently be performed by the different services. It compiles this information into a document which is sent to the user agent. Based on this information, the user then selects the next operation, and so on (Fig. 7). Under certain conditions, 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.

3.2 Analysis

To what extent does the design specified above fullfil the requirements of Section 2.3? In this section, we will analyze this question using examples from the case study.

Execution Services: In principle, our proposed execution services can properly execute the modules from the case study. Care has to be taken, however, that the execution services have access to all required submodules. All three scripts in the case study call library functions that were made available by some statistical computing package. In the Matlab example, these functions even contained the core functionality of the module. The Matlab execution service therefore has to be given access to the function (e.g. by storing the executable in a directory that is accessible by the service).

Interface definition: This problem is addressed by the meta-information facility that allows authors to describe software modules submitted to the method base service. We argued

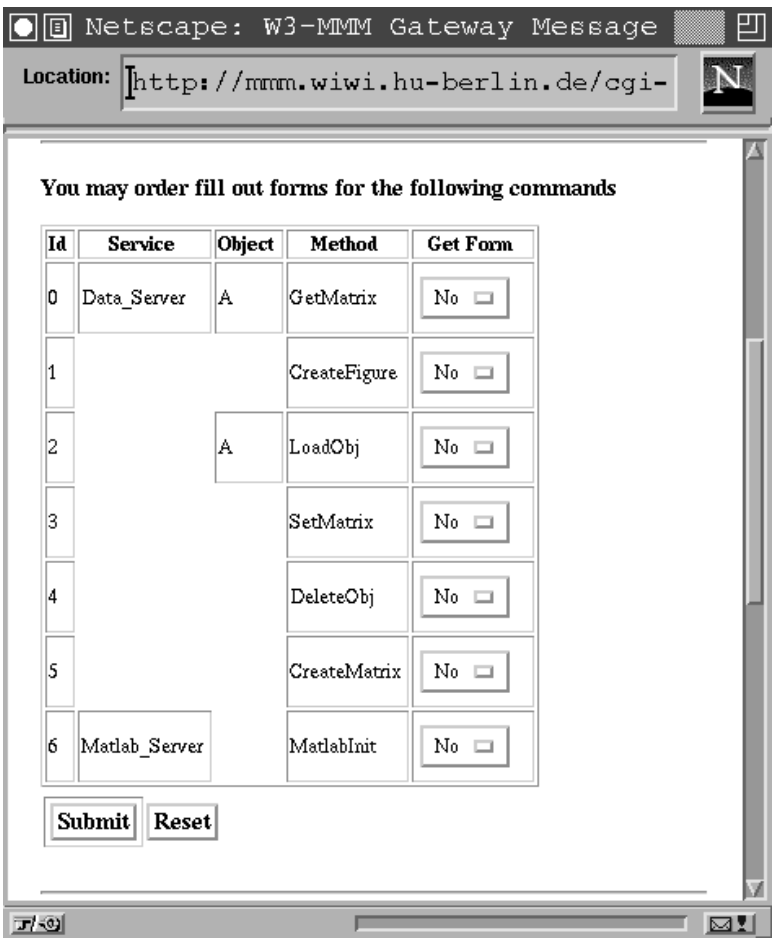

Figure 7: An HTML form to order the appropriate form for the next operation.

above that a signature is usually sufficient, provided the participating data sets are syntactically and semantically correct. Because data sets are usually entered or computed using some standard statistical package, syntactic correctness often reduces to correctness of dimensions. Semantic correctness is a more complex issue, especially because interactions with a module are often only feasible in a specific order. While not yet supported by our execution services, parameterized transaction models would allow such more complex interface definitions [BKM95b]. At this point, however, checking in a new software module is still a complex process that requires a certain familiarity with the system.

Interoperability: This requirement is met by the data translation services and by the fact that we require each module to have a functional interface.

State maintenance: The database and method base services are stateful services that meet the essential requirements on state maintenance. The connection to the execution engines in the execution services is currently stateless - at least, state is not transparent to users and other services.

Scripting: This is not yet covered by our system design. However, as we will see in Section 4, service calls can be integrated into C++-callable libraries, which can be invoked in turn from a Mathematica or Matlab module. Hence, at least some of the packages and their execution services may be used to provide scripting.

Internet/WWW: This last requirement is covered by the WWW-based user interface.

4 The Implementation of MMM

This section describes MMM, a method management system that implements the design presented above 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 object technology is used to realize agents that provide services for authoring and executing statistical methods, and for the maintenance and format conversion of data sets.

For the implementation of MMM we used Ypsilon, a C++ environment for model-based software development. After giving a short overview of Ypsilon, we illustrate how we used this tool 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. For a more detailed description of Ypsilon we refer the reader to [MM94].

4.1 The Software Development Environment: Ypsilon

Ypsilon was developed as a toolbox for model-based software development and rapid prototyping. It provides an infrastructure to encapsulate services into objects implemented in C++. The original focus of Ypsilon was the implementation of algorithm libraries for project scheduling

[MS95]. Its design was influenced by the ASCEND modeling language [Pie89, KMP93] and by the adaptive method base shell AMBAS [HS92]. Ypsilon consists of

1. a generator that produces C++ class implementations from models (vice versa, each Ypsilon class corresponds to a model);
2. compiled C++ and X11/Motif libraries that provide Ypsilon objects with a rich set of functionalities, including graphical user interfaces, ASCII string representations, and memory handling.
3. a generic runtime environment to instantiate and process Ypsilon objects.

The following sections highlight several features of Ypsilon that are of particular relevance for the MMM implementation.

4.1.1 Type Information at Runtime

Using member functions, Ypsilon objects provide runtime access to their own class information. These member functions report, for example, whether a class implements a record or a table pattern and, in the latter case, the index type of the table. MMM uses this kind of meta-information access to implement *generic services*. Generic services are best explained using an example: a (generic) user agent that generates HTML form representations of Ypsilon objects. The meta-information about the class declaration is used to format the HTML form, and current object values are used to set default variables. When a user submits the completed form via a Web browser, the user agent receives the content of the form variables via a CGI application. Based on the form content it then updates the object content. Fig. 5 and Fig. 6 show examples of such forms generated by the user agent.

Why is type information at runtime a helpful feature? We believe that this approach can decentralize generally applicable services. It permits a *slim* implementation of Ypsilon core libraries, while new functionalities, which would usually be implemented as additional member functions of base classes, can be realized as remote services. For example, one could independently develop a new user agent that generates Java applets instead of HTML forms.

4.1.2 Serialization of Ypsilon Instances as Character Strings

Each Ypsilon instance has a representation as a formatted ASCII character string that contains the complete type information. Together with the instance value, this information can be used to define the protocols 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.

4.1.3 Standardized Encapsulation of Services

A *function model* is a special kind of Ypsilon model that encapsulates an (external) function as a C++ class. Each function model has fields `input` and `output`. 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 to the external function, but also format conversions from and to the external function's data structures. Moreover, function models implement a standardized interface for function calls that can be used in a simple manner by other applications.

Function models can also encapsulate stateful services. As such, they are modeled with additonal data fields maintaining the state of the service. The content of `input` is then interpreted as a request to the service, `evaluate` processes this request, and `output` contains the results of evaluation. We will see examples for this usage of function models when we describe MMM services in Section 4.3.

4.1.4 Wrapper Classes for Data and Functions

`GCData` is a wrapper class whose objects can be initialized dynamically with any other Ypsilon model. In combination with the typed ASCII representation, this allows a generic handling of Ypsilon model instances. As an example, suppose that a file `somefile` contains an instance of the class `SomeTable`. Then the lines

```
GCData d;
d.read("somefile");
```

change the virtual type of `d` to `SomeTable` and initialize it with the instance from the file. Although we cannot use any member functions of the class `SomeTable`, we may assign `d` to the `input()` field of a function model and then evaluate this function:

```
SomeFunction f;
f.input() <<= d;
f.evaluate();
```

The class `GCFunction` implements a similar wrapper for Ypsilon function models. We could thus replace the previous three lines of code by

```
GCFunction g;
g.initialize("SomeFunction");
g.input() <<= d;
g.evaluate();
```

The second line initializes the generic function object `g` with the class `SomeFunction`.

4.2 The Communication: MMM Client and Server Models

While Ypsilon function models generally 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], which contains C++ wrapper classes for interprocess communication. The ACE library 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 of type `GCData`. Additionally it has a field `server`, a record that is 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 `server` field, takes the `GCData` object from the `input` field, and sends it to this address. The reply received is assigned to the field `output`, followed by closing the connection to the remote address. Since `input` is of type `GCData`, objects of type `MmmFClient` are usable to send any Ypsilon object via the Internet: one just has to assign the object to `input` and call `evaluate`.

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. Therefore the field `service` is of type `GCFunction`: It can be initialized by any Ypsilon function model (see Section 4.1.4). The member function `evaluate` of `MmmFServer` listens to the port specified by the field `port`. When it receives an Ypsilon model instance from a client, it assigns this instance to the `input` field of `service`. Then the member function `evaluate` of `service` is started and the result, contained in the `output` field of `service`, is sent back to the client.

To summarize, 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 from a C++ program requires initializing an `MmmFClient` object, initializing its `input` with the request, and retrieving 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, which defines in turn which messages are acceptable next [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, such as a request for meta-information, or an execution request to the method base.

MMM employs the concept of acceptable messages at two different levels. First, it is used at the level of general operations that a service can perform. General operations are those which are *not* specific to a software module provided by an author. For those operations, the function that relates state to acceptable messages is hard-coded in the implementation of the service. Second, there are rules for acceptable messages that are specific for software modules authored by system users. For example, it is specific to each Matlab script which data sets have to be initialized before the script can be executed. The function that relates state to messages that start the execution of a module is therefore not hard-coded but parameterized by the signature of the function. This is a simple example for parameterized transaction models for agents in electronic markets [BKM95b].

In summary, our services implement the concept of acceptable messages in the sense that a service has explicit knowledge about which messages it may accept next. Services thus have a local, dynamic repository of their interface. In the case of services for executing software modules, the rules for updating this repository are mostly provided by the software authors.

The MMM broker can be considered as a middleware between user agent and services [Ber96]. Like an object request broker (ORB), it provides transparent access to a distributed system of services. It maintains a dynamic collection of interfaces to these services, consisting of the set of acceptable messages for each service. The connection between Web browser and broker is established by the user agent as described in Section 4.1.1. Whenver the user submits a request, the broker consults one or several services by sending some (acceptable) message to each service concerned. The service's reply message is sent back to the user agent, together with an update of the set of acceptable messages for that service. Based on that new set of messages, the broker generates an HTML form for the user to choose the next operation (Fig. 7). Note that the broker is a stateful service itself; its state is essentially defined by the set of acceptable messages for each connected service.

5 Discussion

In this paper we described the design and implementation of MMM, a method management system to support collaborative statistical computing. 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 with their interface definition.

These features of MMM are not only relevant for collaborative statistical computing. They concern the question of distributed management of software modules in general. A brief comparison with related approaches confirms that MMM offers some functionalities that are not available elsewhere.

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 represents 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 [SUN95] 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 are 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 an option for our typical user, who makes considerable use of the expressiveness and richness of mathematical libraries in scripting languages like Mathematica or Matlab.

As discussed above, there is considerable overlap between the MMM concepts and middleware architectures, in particular object request brokers [Ber96]. 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.

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 scientific software, as they are currently investigated for decision support technologies [BKM95a, BKC+96]. Rather than buying expensive licenses for

comprehensive software environments, consumers will be able to use software modules installed on some other computer in the Internet, paying just a relatively small usage fee.

We also believe that technologies like MMM will have a major impact on the verification and benchmarking of software, both in practice and in research. At this point, only a small percentage of all experiments published in the computer science literature are ever verified [TLPH95]. Once every published paper includes a URL (Uniform Resource Locator), i.e., an Internet address that points to an implementation, other people will be able to test the experimental results with their own data via the Web. This would substantially increase transparency and credibility of our discipline as a whole.

Acknowledgments

Thanks to Peter Becker (University of Tübingen), Wolfgang Härdle, Christian Haffner, René Hoppe, Sigbert Klinke, Ralf Koerstein, Thomas Kötter, Helmut Lütkepohl, Richard Stehle, Rolf Tschernig (all of Humboldt University, Berlin), and Andreas Weigend (New York University) for many interesting discussions and for their contributions to the design and implementation of MMM.

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 T. 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 S. 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.

[Bol86] T. P. Bollerslev. Generalized autoregressive conditional heteroscedasticity. *Journal of Econometrics*, 31:307–327, 1986.

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

[HS92] M. Holocher and D. Solte. AMBAS – an adaptive method base shell. In J. C. Petrie Jr., editor, *Enterprise Integration Modeling, Proc.* MIT Press, 1992.

[JB95] M. Jeusfeld and T. X. Bui. Interoperable decision support system components on the Internet. In S. 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.

[KMP93] R. Krishnan, R. Müller, and P. Piela. Modeling project scheduling models in ASCEND. Working paper, Carnegie Mellon University, 1993.

[LT96] H. Lütkepohl and R. Tschernig. Nichtparametrische Verfahren zur Analyse und Prognose von Finanzmarktdaten. In G. Bol, G. Nakhaeizadeh, and 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.

[Pie89] P. Piela. *ASCEND, An Object Oriented Computer Environment for Modeling and Analysis*. PhD thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, 1989.

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

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

[SUN95] SUN Microsystems Inc. *JAVA Home Page*. http://java.sun.com, 1995.

[TLPH95] W. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heinz. Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software*, 28(1):9–18, 1995.