Middleware for Software Leasing over the Internet*

H.-A. Jacobsen and O. Günther†

Institute of Information Systems
Humboldt–Universität zu Berlin
Spandauer Str. 1
D–10178 Berlin, Germany
{jacobsen, guenther}@wiwi.hu-berlin.de

Abstract

MMM (Middleware for Method Management) is an infrastructure for managing the deployment, integration, distribution, and use of application services via the World Wide Web. Application services may range from a simple database access to a fully fledged application package. MMM propagates a software leasing paradigm, as opposed to the classical software licensing model. Applications reside and execute on the software provider's platforms, managed through the MMM middleware. Users interact with the application services through a standard Internet browser, not requiring any additional software. This relieves user from software installation, maintenance, and upgrading overhead, while always offering the most recent software release. The MMM client component offers users a virtual file space, application service composition functions, execution support, and visualization features. These functions are all available through an Internet browser. The MMM implementation is based on standard Web technologies, such as HTML, XML, and MetaHTML; distributed object computing frameworks, such as CORBA; and database technology, such as ODBC. In this paper we give a technical account of the MMM architecture and discuss its primary features.

1 Introduction

Business–to–consumer electronic commerce applications, such as online ordering of books, online scheduling and booking of flights, online information services of various kinds, and online banking have found widespread deployment and acceptance. Main characteristics of these applications are limited user–interaction, the small amount of input data required by the user, and the straightforward application design.

Electronic commerce applications that are more computationally intricate are only slowly gaining momentum. These kinds of applications require moving the data, the required algorithms, or both among remote processing nodes. Moreover, such applications are characterized by complex user interaction patterns, large volumes of data that need to be made available to the remote sites (e.g., data analysis), workflow–oriented nature of the processing, (e.g., multi–stage computations by distributed processing units), multiple inter–transactional user interactions, and longer processing times of the individual transaction steps. In order to discuss such more complex services, we first introduce some basic concepts:

Data Provider: Data providers publish data. They may simply submit files to an online repository, or provide a full–fledged online accessible data source that delivers data automatically (push) or on demand (pull). Examples include online stock quote services, geographic information systems, and consumer pricing data services.

Computational Service Provider: Computational service providers (a.k.a. application service providers) own the application or service being offered online. Application services may be offered by companies, public institutions, or individuals (e.g., researchers). Examples include complex data analysis tools (e.g., for financial forecasting or geographic information management), or sophisticated business software (e.g., for enterprise resource planning and accounting purposes).

Application Server Provider: Application server providers (ASPs) host the application services and provide network access to it. ASPs and computational service providers will often coincide.

Infrastructure Provider: The infrastructure provider offers a framework that grants secure access to application server sites to authorized users. The infrastructure provides functions to establish secure connections, to interoperate applications, and to manage services remotely. It also offers payment and accounting services to support a pay–per–usage business model.

The interplay between these various agents leads to *computational service infrastructures* that provide a wide range of services on a pay–per–usage basis. This service approach, often also called "software leasing", has to be seen in contrast to today's stand-alone software solutions, such as decision support systems (DSS) and business software (e.g., SAP R/3) that usually require large investments in software for installation, maintenance, upgrading, and an armada of well–trained personnel to deploy the system effectively.

†Also with Pole Universitaire Leonard de Vinci, 92916 Paris La Defense Cedex, France.

We believe that the main problem hindering a more rapid evolution of computational service infrastructures is the lack of a common platform that targets specifically the integration of heterogeneous information services and electronic commerce applications. The problem is amplified by the fact that most current electronic commerce systems are ad hoc solutions lacking interoperability, re–usability, and extensibility. The momentum and popularity gained by current online solutions are driving the development of ever new systems. To achieve short time to market, little effort is put into the construction of an overall infrastructure providing basic services and facilities.

Reusable components would be important for two main reasons. First, there are many issues of data security, accounting, and transaction management that components could solve once and for all, independent of any particular application. Second, transparent interoperation would be a key to add value to individual services. Indeed, interoperation would enable a workflow between services which would be convenient for users and, moreover, create strong network externalities.

Such techniques would also facilitate the move from current all–or–nothing payment schemes (i.e., the classical software license business model) to more flexible ways to pay for selected data sets or for the utilization of specialized software. Such *pay–per–usage* schemes would allow smaller suppliers to enter the market easily (i.e., with little set-up cost). Users will find it easy to get exactly the data they want, or to get the data service that matches their budget. As the budget increases, a new data service will be chosen without need to recode the interface to their application. In the case of stock data, for example, users can decide whether to buy expensive "premium" stock data or stick to freely available sources. Perhaps they will start application development with free data, determining their needs. As development is finished and revenue increases, they may move on to a service that is more sophisticated but also more expensive.

As our own response to these concerns, we have designed and implemented MMM (Middleware for Method Management), a computational infrastructure for administering application services from distributed and heterogeneous application server providers (ASP). MMM is a middleware that allows application service providers to publish services, data providers to publish data sets, and consumers to use the online services. The middleware aims at making the location of data and application services, as well as the platform for executing the applications, transparent to the interacting user.

The first MMM prototype, focusing entirely on sharing statistical software, was discussed by Günther *et al.* [5]. Initial design, implementation, and deployment lessons gained from this prototype are summarized in Jacobsen *et al.* [7] and have led to a complete re–design and re–implementation. Jacobsen *et al.* [6] present a detailed discussion of the new MMM infrastructure focusing on deployment, security, and economic issues.

This paper concentrates on a technical discussion of the MMM infrastructure, presenting the MMM object model and the MMM architecture It highlights selected design issues, such as the integration of distributed object and Web technology, and the MMM server design (Section 2). User interface issues are discussed in Section 3. Section 4 gives a detailed account of related work, presents a summary of the status of the implementation, and motivates a business model for deploying a software renting based scheme. Concluding remarks and open research questions are discussed in Section 5.

2 MMM design and architecture

This section discusses the overall MMM architecture. We begin by describing the MMM object model that forms the foundation for all system operations. We then motivate the architecture and discuss major system components, as well as communication protocols developed for the specific application context. We then discuss support for the integration of Web and distributed object technology in MMM.

2.1 The MMM object model

MMM defines an object model for the management of application services and data from distributed and heterogenous provider sources. These objects define primary abstractions for all operations in the MMM infrastructure and constitute the basis for all user–system interaction, such as method and data entry, database loading, communication among system components, and communication with distributed application services.

All entities managed by MMM belong to one of the following classes: *data set objects* (DSO), *method service objects* (MSO), and *method plan objects* (MPO). All objects instantiated through these classes support an extensible set of operations. All MMM object classes are derived from the abstract *MMM Root Class* which offers a common set of operations and attributes to its child classes. The following gives more detail on these classes:

- **MMM Root Class:**

  This abstract class defines a common set of operations and attributes which are inherited by all child classes. This comprises operations for HTML data entry form generation from object descriptions, database storage and access functions to retrieve and manipulate objects, wire transfer operations to send serialized objects to remote locations, and visualization support to browse though the objects.

- **Data Set Objects (DSOs):**

  DSOs represent abstractions for user input and output, as well as input drawn from bulk data providers. DSOs consist of metadata about the data used in computations, such as data format, type, location, size, source, and provider. *Access restrictions* (a notion similar to UNIX file permissions) designate *who may access the data and by which methods.* All attributes may be set and read by *authorized* clients of this object. DSOs support operations for demand–driven data retrieval, data storage, and for security management (i.e., to enforce the access restrictions).

- **Method Service Objects (MSOs):**

  MSOs represent abstractions for managing application services accessible through the MMM infrastructure. Such services offer computational *methods* in our current application context. MSOs consist of metadata on the encapsulated method, such as service provider, author, signature (i.e., input and output parameters), source code, if available, and pertinent execution environment. All attributes may be set and read by authorized clients of the MSO. MSOs are the primary

building blocks for *method execution plans* that either describe the execution context for one single method, or an entire sequence of methods to be executed together with necessary input parameters.

- **Method Plan Objects (MPOs):** MPOs represent abstractions for method execution. MPOs tie MSOs together with DSOs for input and DSOs for output, i.e., they represent a computational method (or sequence of methods) together with their input and output parameters. MPOs have attributes designating user identification, password, and linked methods and data objects. Operations on MPOs include plan creation, population with MSOs and DSOs, segmentation into computational units amenable to distribution, execution initiation, and intermediate result manipulation. MPOs constitute the unit of communication between user and MMM infrastructure and between execution environment and service engines (see below for a detailed discussion).

```
<!ELEMENT dso (general, set+, domain*)>
<!ATTLIST dso name CDATA "">

<!ELEMENT description (#PCDATA)>
<!ELEMENT general (person+, creation, version, rights)>
<!-- [...] some parts left out -->

<!-- Element set: loc/value and description of the actual data -->
<!ELEMENT set  (refer|value)>
<!ELEMENT refer (file|database)>
<!ATTLIST refer URL CDATA "">
<!ELEMENT file (type)>
<!ATTLIST file encoding (ASCII|Binary) "">

<!-- Element: basetype either Integers, Doubles, or Strings. -->
<!ELEMENT basetype EMPTY>
<!ATTLIST basetype
          type  (NAME|integer|double|string) "NAME"
          encoding (BITS|8|16|32|64|128|UTF-8|US-ASCII) "BITS">

<!ELEMENT type (matrix|basetype)>
<!ELEMENT matrix (basetype)>
<!ATTLIST matrix
          rows CDATA "Number of rows, -1 if unknown"
          cols CDATA "Number of columns, -1 if unknown">

<!-- Host details are given by the refer URL attribute. -->
<!ELEMENT database (type)>
<!ATTLIST database
          user     CDATA "Name of User"
          password CDATA "Password of User"
          query    CDATA "Query Statement"
          driver   CDATA "Driver required for Database">

<!ELEMENT value (type, vdata)>
<!ELEMENT vdata (#PCDATA)>

<!-- [...] some parts left out -->
```

Figure 1: XML DTD for a *Data Set Object*. Some parts intentionally left out.

This object model is implemented through a combination of complementing Web technologies, namely XML and MetaHTML. The *Extensible Markup Language* (XML) defined by the WWW Consortium (W3C) is a data format for structured document interchange on the Web. In contrast to

```
<mso name = "MSO2">
<general>
  <person type = "Provider"  lastname = "Riessen"
          firstname = "Gerrit"  middle = "A">
  <email email = "riessen@wiwi.hu-berlin.de"></email>
  </person>
  <creation year = "1998" month = "Sep" day = "24" [...] >
  </creation>
  <version>demo version</version>
  <rights owner = "All" group = "Read/Execute" world = "Read">
  </rights></general>
<source>
   <refer URL ="mmmtp://meta-mmm/riessen/SourceToPlus/
                riessen:zxeFA72FP1mVY" encoding = "ASCII">
  </refer>
</source>
<interface>
  <language choose = "Other">
  <other language = "expr"></other>
  </language>
  <input>
    <parameter name = "B" description = "First Input">
      <type>
        <basetype type = "integer" encoding = "16"></basetype>
      </type></parameter>
    [...]
  </input>
  <output>
    <parameter name = "E" description = "1st Output Parameter">
      <type>
        <basetype type = "integer" encoding = "16"></basetype>
      </type>
    </parameter></output></interface>
</mso>
```

Figure 2: An XML Method Session Object instantiation. ("[...]" denote minor pars left out.)

HTML it allows the user to *extend* the language features to process many different classes of documents. XML may thus be used to define *customized markup languages* for document processing on the Web. MetaHTML [1] is a server side interpreted language for the generation of dynamic HTML pages. Its features include the maintenance of user session state information, seamless access to server side applications, and fast ODBC database accesses.

In the MMM object model we use XML DTDs (Data Type Definitions) to declare the object classes and XML documents to define the object instances. MetaHTML implements the class behavior manipulating the XML objects. The structure of all MMM object classes (MSO, DSO, and MPO) is defined through XML DTDs. This permits the generation of all kinds of support tools for the manipulation of object instances from one single definition, such as HTML forms for data entry and browsing, database records for storage, and transport records for communication. An XML document instantiates an MMM object (i.e., provides its attributes). Method plan objects describe the relation between multiple methods and the data required to execute them. A particular plan instantiation is an XML document. Figure 1 depicts the DTD for data set objects. Figure 2 shows an instantiation of an MSO and Figure 3 shows an instantiation of an MPO (see also [6] for further details).

[1] `http://www.metahtml.com`

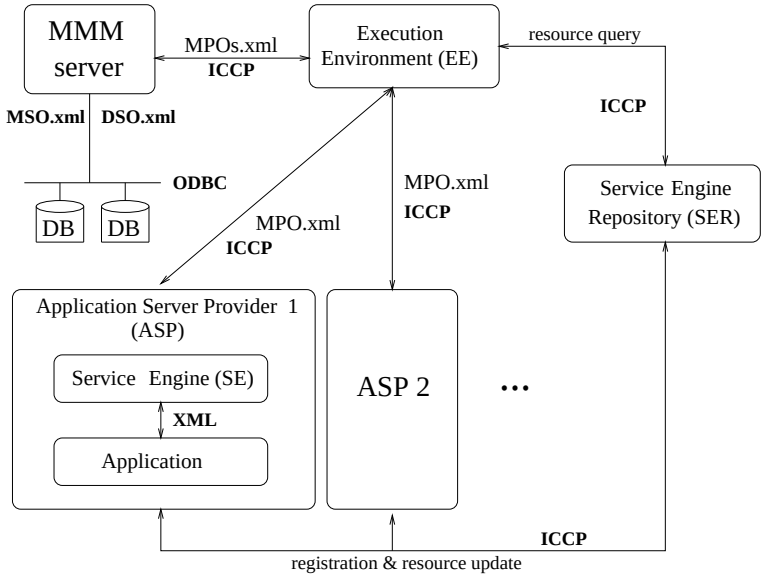

Figure 4: MMM infrastructure architecture.

2.2 The MMM architecture

The overall architecture of the MMM infrastructure is depicted in Figure 4. The MMM middleware consists of the following key components: *execution environment*, *service engine registry*, *service engines*, and *MMM server*. It implements several application–level protocols for inter–component communication.

The *execution environment* (EE) is responsible for the execution of methods. It schedules the method plans trying to utilize the available computing resources in an optimal manner. The EE incorporates several optimizations for minimizing data movement among computing resources and maximizing throughput. It knows about the available resources and performs load balancing. The EE receives a method plan object as input and returns a method plan object as output. The output MPO describes the result of the computation. It specifies location of result data, computation time, possible error conditions, and the like. The execution environment interfaces with an application service through a service engine. It uses the service engine registry to obtain location details of the service engine (SE) driving the application service.

The *service engine registry* (SER) is used to register and de-register application services and to maintain resource and availability information (location, network parameters, machine specifications, etc.). The registry also maintains an up to date mapping of application services to service locations and resource characteristics. It thus provides vital information for the EE, which regularly queries the SER for this information to derive plan scheduling decisions. The SER also serves to decouple the EE from the service engines (i.e., the application services) and allows for dynamic extensibility of the system. Application servers may be added at system run time without loss of operation and become immediately available as services. By frequently updating its resource list, the SER provides simple load balancing functionality that help to distribute the computational load across the available services, ensuring that no one service engine is overburdened.

The *service engines* (SE) encapsulate application services linking them into the MMM middleware. Service engines are responsible for converting data between the internal MMM format and the format required by the underlying application, retrieving data from remote sources, invoking the requested method on the application, and passing result information back to the MMM middleware. Note that "result information" can either be the result of the computation or a reference to the result of the computation, depending on the mode of invocation (cf. discussion on application-level protocols below).

Each component in the MMM architecture is designed to be independent of implementation language and location within the distributed computing environment (i.e., the Web). Components can be configured to dynamically alter their information on other components. Components can be duplicated on the Web to increase availability.

2.3 Application-level protocols

Three kinds of application–level protocols have to be distinguished in the MMM middleware: inter–component communication, demand driven data access (a.k.a. mmmtp = MMM Transport Protocol), and database access protocols (cf. Figure 4 and Figure 5).

2.3.1 The iccp protocol

The protocol linking the middleware components is a three stage sendmail–like protocol: logging in to a component, placing a request with a component, and transmission of the data required for the request. Results are either returned immediately, or the addressed component re-establishes the connection initiating the transmission of the result data. This latter modus of operation is applied if the requested computation takes more than a certain amount of time to complete. The protocol design is in line with the object

```
<plan id = "N2H4YTY9P5">
  <order> <mth id = "PlusMSO" position = "1"></mth> </order>

  <linkedmethod>
    <method id = "PlusMSO" language = "expr">
      <inputs>
          <nvpair name = "parameter1">
            <refer encoding="ASCII" url="mmmtp://[...]">
              <type>
                <basetype type="int" encoding="16"></basetype>
              </type>
            </refer>
          </nvpair>
          <nvpair name = "parameter2">
            <refer encoding="ASCII" url=``http:// [...]''>
              <type>
                <basetype type="int" encoding="32"></basetype>
              </type>
            </refer>
          </nvpair>
      </inputs>

      <outputs>
          <nvpair name = "OutPara1">
            <value>
              <type>
                 <basetype type="int" encoding="32"></basetype>
              </type>
              <data>Outputs Are Undefined</data>
            </value>
          </nvpair>
       </outputs>
       <source>
         <refer encoding="ASCII" url="mmmtp://[...]guest:GUEST">
           <type>
             <basetype type="string" encoding="US-ASCII"></basetype>
           </type>
         </refer>
       </source>
       <libraries></libraries>
    </method>

    <linkmap>
      <link name = "OutPara1">
        <save url="mmmtp:// [...] /guest:cdEi19E3I">
        </save>
      </link>
    </linkmap>

  </linkedmethod>
</plan>
```

Figure 3: XML DTD for a *Method Plan Object*. ("[...]" denote minor pars left out.)

model supported by the middleware. Table 2.3.1 summarizes the main commands of the iccp–protocol. iccp is implemented on top of http. Each iccp enabled component supports a MetaHTML iccp–protocol engine.

Operation	Description
`LOGIN`	Login with `username` and `password`.
`EVAL`	Informs service engine that next call to the `DATA` operation will be to send a complete method (MPO) for evaluation.
`DATA`	Prepare to receive data.
`RESULTS-...`	Prepare to receive results.
`STATUS`	Query status of service engines.
`ADMIN`	To administrate service engine.

Table 1: The MMM inter component communication protocol key operations summary.

2.3.2 The mmmtp protocol

The mmmtp protocol permits service engines to retrieve data directly from the infrastructure repositories via the MetaHTML enabled web server. The protocol has been developed as an extension of ftp for several reasons:

1. Fast, direct, and highly optimized database access.
2. Demand driven data retrieval, i.e., data is only retrieved *when* and *where* it is needed; otherwise,the system manipulates referential information and metadata only (i.e., MSOs, DSOs, MPOs).
3. Need for fine-grained access control mechanisms beyond what is currently available for ftp.

For mmmtp we define a new type of URL: `mmmtp` which allows retrieval of data and objects from the infrastructure repositories. The form of this URL is:

```
mmmtp://<host>:<port>/<path>/
        <file name>/<user name>
```

Notice that the user name must be appended to the URL definition. For example to retrieve the file `/guest/MatrixData` as user `foobar` from the MMM repository:

```
mmmtp://meta-mmm.wiwi.hu-berlin.de:80/
       guest/MatrixData/foobar
```

mmmtp offers an abstraction for the underlying application when retrieving data. The requester of the data is unaware of whether the supplier is an application, a file, an object, or a repository. All that is required is an identification of the source (host, port and path components in the mmmtp resource locator) and a requester identification (username).

mmmtp is similar to ftp, however, an ftp request always represents a file. One could argue that such a file may in fact be a stream or process abstraction. However, this breaks the definition of ftp, hence, mmmtp extends ftp by removing the requirement that a file is the underlying entity being accessed.

mmmtp is also two-way: it can be used to define the target entity waiting to receive data. An example is the request that an application, on completion, sent its output to a specific mmmtp location. In this case the resource locator represents a receiving entity, and the user identification represents the identification information to be used by the sending application to identify itself with the receiving entity.

Common URIs such as `http` and `ftp` are also supported. We are currently prototyping support for the OMG iiop protocol (URI `iiop`) for direct communication with CORBA–compliant applications.

2.3.3 Database access protocol

For database access we rely entirely on the ODBC support provided by our primary development language — MetaHTML. We currently use the mSQL database from Hughes Technology as underlying DBMS. Due to the strict use of the ODBC interface to access the database, the underlying DBMS technology may be substituted at any project stage.

2.4 Service engine interfaces

Interfaces between SE and application are also defined through XML documents to be easily accessible from the Web–protocol based MMM server. A single XML document is required to configure a new application server. The document (application wrapper) defines how parameters are passed to the application, whether or not the application is used in batch or interactive mode, what execution permissions need to be set to access its services, and the like.

2.5 The MMM server

The MMM server manages all infrastructure operations. It performs the actual method execution management, the assembly, storage, and retrieval of MMM objects, the security and access control management, and the communication with foreign object systems. In the future it will also perform the service accounting and payment functions. MMM objects are stored in a relational database accessed through the MetaHTML ODBC interface directly from the server (cf. Section 2.3.3). The MMM server architecture is depicted in Figure 5.

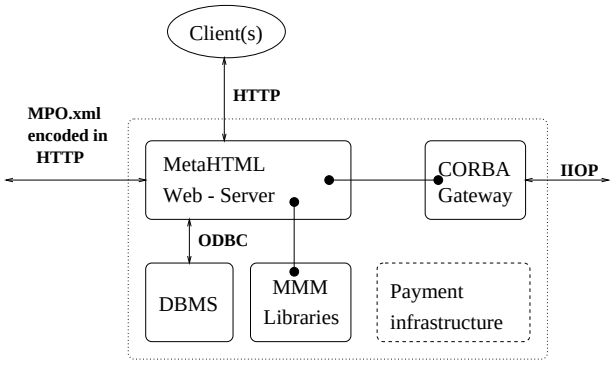

Figure 5: MMM server architecture.

The server consists of the following sub–components: The MetaHTML Web server, the database management system, the MMM infrastructure libraries, the CORBA gateway, and, in the future, the payment sub–system. The MetaHTML Web server supports the MetaHTML Web development language, our primary tool for the infrastructure development.

The MMM server communicates with customers via the standard http protocol including the necessary security precautions. It sends XML–encoded MPOs via http request to the execution environment, which further distributes the load to service engines (cf. Figure 4).

2.6 CORBA object system integration

To this end we have described how the MMM infrastructure integrates stand–alone computational packages. Remote system functions and input data are encoded as MMM platform objects (MSO, DSO, MPO) which are manipulated by the infrastructure components. However, several reasons mandate alternative integrative approaches:

1. application–level[2] integration of specialized libraries, such as LaPACK, BLAS, or other special purpose computing libraries, for the particular application context targeted;
2. application–level support for highly specialized computational hardware platforms;
3. infrastructure–level integration of horizontal domain facilities, such as the OMG payment object framework, the OMG negotiation facility, and object document models, for instance;
4. infrastructure–level integration of distributed computing services (e.g., naming service, trading service, security service, and transaction service).

The former two points address the application context, i.e., the components leased via the MMM infrastructure. The latter two address the infrastructure design. Both categories are equally important and are solved through the same mechanism by integrating complementing CORBA distributed object technology and MetaHTML Web technology.

The fundamental difference between stand–alone computational packages, on the one hand, and specialized libraries and standard distributed system services, on the other hand, is the way they are deployed. Packages are script–driven and completely independent, whereas libraries and services are linked into the application serving as a dependent functional layer.

The integration is based on the CORBA Dynamic Invocation Interface (DII) which allows to assemble method invocations without explicitly generating stubs, as is common for remote method invocation supporting systems. This design allows us to by–pass the stub generation phase and directly access the foreign object at system run–time without the need to halt operation and re–compile the involved MMM infrastructure and foreign object system components. The CORBA DII provides operations to create, populate, and invoke a request object (i.e., an object that incarnates an

[2] With "application–level" we denote the MMM application, as seen by a user; with "infrastructure–level" we denote the MMM infrastructure, as seen by the infrastructure developer.

operation invocation). A request object has attributes for operation name, argument mode, type, and value, as well as result value.

The integration of a third party CORBA compliant system is performed through HTML forms. User input is passed as form data over http to the Web–server (i.e., MMM–server in our case) which further processes the input by calling on the server–side integrated CORBA gateway functions of the payment system. Operation requests are passed through the dynamic invocation capabilities of a CORBA distributed object platform.

The HTML–CORBA gateway is realized as MetaHTML implementation of the CORBA DII interface, i.e., by passing MetaHTML calls to C/C++ CORBA DII calls.

This HTML–CORBA gateway allows us to interface to any CORBA compliant system. Gateways to other distributed object systems may be implemented in a similar fashion.

3 MMM user interface

The MMM user interface (UI) is entirely based on Internet browser technology using a combination of HTML, XML, and MetaHTML. The MMM UI needs to support viewing, browsing, searching, entering, editing, and modifying functions for all objects managed by MMM. Another crucial issue important to the MMM UI design is simplicity, both in terms of navigation and in terms of usability.

The UI provides a desktop environment for managing method service and data set objects. This includes composition of method plans, selection of input parameters, execution support, and result notification functions. Moreover, for application service providers the MMM UI provides bulk data entry functions that allow easy and quick registration of application services. For the MMM infrastructure administrator the UI provides administration tools for managing and monitoring user actions (e.g., monitor active user sessions).

Navigating through the MMM infrastructure and service space is supported by providing each user with a *Virtual File Space* (VFS), a notion similar to the Windows file manager. Technically, this view maps a flat RDBMS table space into a file and directory hierarchy combined with file permissions on each file and directory. The access control is enforced by the Web server and users are able to restrict areas of their file space from being viewed by other users. These features may be tested by accessing our web site (`http://meta-mmm.wiwi.hu-berlin.de`).

Figure 6 shows the split window view of the MMM VFS. The top frame gives an overview of the directory hierarchy showing the user's directory, the MSO and the DSO directories, and the *shared* directories. The latter three are intended as public spaces for users to share methods and data. The user's private spaces allows for experimentation and testing. The bottom view shows a single directory and allows the user to invoke "*file–operations*" on single items in the directory. Operations include `Deletion`, `Move`, `Copy` and `Plan Insertion`. Directory items are selected using check boxes, and file–operations are chosen using a selection menu. Searching the MMM repository for specific functionality is also possible.

4 Status and related work

Since early 1997, MMM has been used as an experimental method management infrastructure in two contexts. First, it was used to exchange statistical software modules within the German National Research Center on the Quantification and Simulation of Economic Processes (SFB 373). This application was described in detail in [5]. Second, we used MMM to facilitate the exchange of experimental software within a distributed cooperation of optimization researchers [8].

To verify the described architecture and to test the concept of component leasing we have implemented the MMM middleware and populated it with several data sources, application services, and methods. Our primary area of application lies in the mathematical computing domain. However, the MMM infrastructure is not limited to this application domain. It is well suited to integrate all kinds of script–driven computing applications, supports the integration of stand–alone libraries, and the access to IIOP supporting distributed object systems. The MMM prototype is fully operational. It allows to register methods, apply methods, form heterogeneous execution plans, and integrate remote data sources. The effort of populating the middleware is still ongoing. The MMM infrastructure is available online at:

`http://meta-mmm.wiwi.hu-berlin.de`

The infrastructure may be tested via a guest account.

Commercial online application service providers are now going to market. Olsen and Associates[3], for example, offers a comprehensive set of software modules for financial market traders. Start–up companies such as GreatPlains Inc., FutureLink Inc., and most big software and hardware vendors are providing or planning to provide online application services. Some companies, such as Thinter Net, go one step further and offer the computational infrastructure (server farm) for others to quickly deploy online applications. This latter undertaking is closest to our objectives. However, we go yet one step further and provide functions to interface heterogeneous application services and data sources in a workflow oriented manner and provide the necessary mediation technology.

The Argonne National Laboratories' NEOS service, [2] provides access to optimization software, for use by researchers world wide. This service is close to the emerging ASP model, where a single institution is responsible to set up all resources, including the server–side infrastructure, the integration of the computational packages at the server, and the client–side user interface. A parameterizable set of algorithms is provided that solves a specific task on a single site. Resource diversity and heterogeneity can thus be largely avoided. These latter points are the primary concerns of the here introduced MMM infrastructure.

A more open approach is explored in the *DecisionNet* project [1]. DecisionNet is an organized electronic market for decision support technologies. The market infrastructure consists of agents that support consumers and providers in transactions. The decision technologies themselves reside on provider machines distributed across the Internet. Anybody with Internet access can participate in DecisionNet as provider or consumer.

The objective of the MMM infrastructure is to implement a model where software may be accessed as a rentable

[3] http://www.olsen.ch

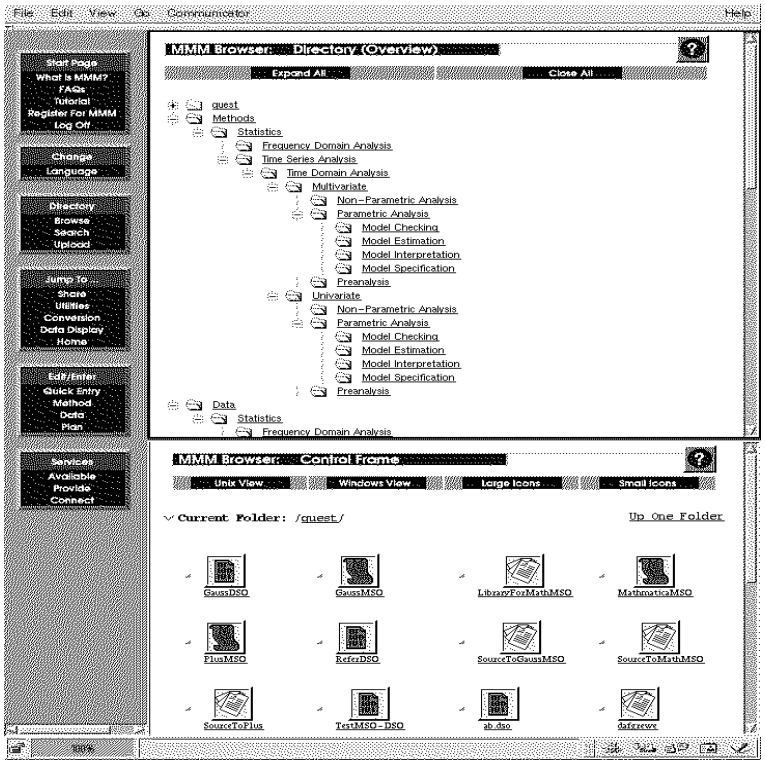

Figure 6: The MMM Virtual File Space.

good. A consumer deploys the rented application, as he or she would deploy a properly owned system, but does not engage in major programming activities to put together a new application. This is in contrast to infrastructures such as Legion [4] and Globus [3] which aim at providing a virtual computer to its users. The virtual computer incorporates heterogeneous computing resources world wide. However, the user uses this virtual computer much like a desktop computer in writing and running applications.

5 Conclusions and future work

Many of the functions performed by complex software packages, such as geographic information systems, data analysis tools, financial packages, accounting solutions, and taxation software, seem to be amenable to a software deployment and business model that is fundamentally different from the one we see today. At present, users typically own the hardware and software they operate on. They pay license, upgrading, and maintenance fees to various vendors and they have to train their staff in using and maintaining the system. The alternative would be a leasing–based approach where users make their input data available to a service that performs the necessary computations remotely and sends the results back to the user. Customers pay only for that particular usage of the technology — without having to own the entire package. Moreover, the underlying infrastructure is open for anybody to participate as consumer or provider of computational services and data sources. We believe that with such an approach the number of consumers of specialized computational services, such as the ones motivated throughout this work, will be much greater than the number of users of stand–alone packages today. We also expect customer satisfaction to increase.

Our MMM system is one example of a middleware that implements such a software infrastructure and offers a component leasing based business model. Its support for consumers includes features for browsing, searching, and querying available methods and data sources. Support for providers concentrates on the registration of new methods and on the related management of meta–data. The MMM platform is fully implemented and accessible on our WWW site: `http://meta-mmm.wiwi.hu-berlin.de`.

Before these kind of software component leasing infrastructures will become commonplace, however, several critical research questions will still have to be answered. This includes issues related to licensing and pricing of software deployed on a leasing basis, certification of services to ensure customers a certain "quality of service", and simplicity of user–system interaction.

Acknowledgments

Support from the German Research Society (DFG grant nos. SFB 373/A3 and GRK 316) is gratefully acknowledged. We would also like to thank Rudolf Müller and Gerrit Riessen for their contribution to the MMM infrastructure; Brian Fox who has considerably contributed to the re–implementation of the MMM infrastructure through his MetaHTML Web development language and tool; and all other members of the MMM team for their contributions to this paper.

References

[1] 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.

[2] J. Czyzyk, M. P. Mesnier, and J. J. Moré. The networked-enabled optimization system (NEOS) server. Preprint MCS-P615-1096, Mathematics and Computer Science Division, Argonne National Laboratory, March 1997.

[3] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Int. Journal of Supercomputing Applications*, 11(2):115–128, February 1997.

[4] A. Grimshaw and W. Wulf. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, January 1997.

[5] O. Günther, R. Müller, P. Schmidt, H. Bhargava, and R. Krishnan. MMM: A WWW-based approach for sharing statistical software modules. *IEEE Internet Computing*, 1(3), 1997.

[6] H.-A. Jacobsen and O. Günther. Component leasing on the World Wide Web. *NETNOMICS*. (accepted for publication).

[7] H.-A. Jacobsen, G. Riessen, O. Günther, and R. Müller. MMM — Towards an infrastructure for emerging electronic commerce applications. In *Information Technology and Electronic Commerce*, DEXA Workshops, Florence, September 1999.

[8] O. Günther und R. Müller. From GISystems to GIServices: Spatial computing in the internet marketplace. In M. Egenhofer und M. Goodchild, editor, *Interoperability in Geographic Information Systems*. Kluwer Academic Publisher, 1999.