Component leasing on the World Wide Web

H.-A. Jacobsen, G. Riessen, O. Günther

Institute of Information Systems

Humboldt-Universität zu Berlin

Spandauer Str. 1

D–10178 Berlin, Germany

{jacobsen, riessen, guenther}@wiwi.hu-berlin.de

Abstract

MMM is an infrastructure for managing the deployment and use of distributed application services on the WWW. MMM propagates a paradigm that enables the leasing of software components, as opposed to the classical software licensing model. Applications reside and execute on the software provider's platforms, but are managed through the MMM infrastructure. Users interact with the application services through a standard Internet browser, not requiring any additional software. The MMM user interface offers users a virtual file space, application service composition functions, execution support, and visualization features. 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 data input 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 ships or pushes data on demand. Examples include online stock quote services, geographic information systems, and consumer pricing data services.

Computational Service Provider: Computational service providers (a.k.a. application 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 computational services (application) 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.

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 point–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 development is put into an overall infrastructure providing basic services and facilities.

Reusable components would be important for a broader success for two main reasons. First, there are many issues of data security, accounting, and transaction management that components could solve once across the various applications. 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 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 a revenue can be derived, they will move on to a new service or provider that allows to get an appropriate cost/revenue–relationship.

As our own response to these concerns, we have designed and implemented MMM (Middleware for Method Management), a computational infrastructure for administering application services ("methods") from heterogeneous application server provider sites. MMM is a middleware that allows application providers to publish computational services, data providers to publish data sets, and consumers to apply methods to data. The middleware aims at making the location of data and methods, and platform for

executing the methods transparent to the interacting user.

The first MMM prototype, focusing entirely on sharing statistical software, was discussed by Günther *et al.* [12]. Initial design, implementation, and deployment lessons gained from this prototype are summarized in Jacobsen *et al.* [15] and have led to a re–design and re–implementation. A brief overview of the revised MMM architecture is presented in Riessen *et al.* [16].

This paper concentrates on a technical discussion of the MMM infrastructure, presenting the MMM object model, the MMM architecture, system internal protocols, and highlights several selected design issues, such as the integration of distributed object and Web technology, and MMM server design (Section 2). Key to the success of the proposed leasing model are security considerations, which are summarized in Section 3. User interface issues and a status summary of our prototype implementation are given in Sections 4 and 5, respectively. We defer a discussion of related work to Section 6 to be able to better place our work in the context of related efforts. Open research questions for further exploration are summarized in Section 7. Concluding remarks are given in Section 8.

2 MMM middleware 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 MMM architecture design, discuss major system components, and communication protocols developed for the specific application context. We then discuss several innovative issues original to the MMM infrastructure, such as a fine–grained access control scheme to protect valuable data from being compromised and support for the integration of distributed object technology.

2.1 The MMM object model

MMM defines an object model for the management of methods 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 population, 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*

```
<!ELEMENT mso (general, source, library*, interface, domain*)>
<!ATTLIST mso name CDATA "">

<!ELEMENT general (person+, creation, version, rights)>
<!ELEMENT person (affiliation?, address?, url?, email)>
<!-- [...] some parts left out -->
<!ELEMENT rights >
<!-- [...] some parts left out -->
<!ELEMENT refer EMPTY>
<!ATTLIST refer URL CDATA ""  encoding (ASCII|Binary) "">
<!ELEMENT code (#PCDATA)>
<!ELEMENT source (refer|code)>
<!ELEMENT library (refer|code)>

<!-- interface definition language ------------------------------>
<!ELEMENT interface (language, input, output, precond?, postcond?)>
<!ELEMENT language (other?)>
<!ATTLIST language
      choose  ( LANGUAGE | Matlab | Gauss | Progress | Maple |
                XploRe | Mathematica | JDBC | Other ) "LANGUAGE" >
<!ELEMENT other EMPTY>
<!ATTLIST other language CDATA "Name of Language">
<!ELEMENT input      (parameter*)>
<!ELEMENT output     (parameter*)>
<!ELEMENT parameter  (type)>
<!ATTLIST parameter   name CDATA "Method Name of Parameter"
                      description CDATA "Description of Parameter">
<!ELEMENT basetype EMPTY>
<!ATTLIST basetype
          type  (NAME|integer|double|string) "NAME"
          encoding (BITS|8|16|32|64|128|UTF-8|US-ASCII) "BITS">
<!ELEMENT matrix (basetype)>
<!ATTLIST matrix
           rows CDATA "Rows of the matrix, -1 if unknown"
           cols CDATA "Columns of the matrix, -1 if unknown">
<!ELEMENT type          (matrix|basetype)>
<!ELEMENT precond        (#PCDATA)>
<!ELEMENT postcond       (#PCDATA)>

<!-- domain-specifics ------------------------------------------->
<!ELEMENT domain       (description, keyword*, abstract?, paper?)>
<!ELEMENT keyword      (#PCDATA)>
<!ELEMENT description  (#PCDATA)>
<!ELEMENT abstract     (#PCDATA)>
<!ATTLIST abstract     href CDATA "">
<!ELEMENT paper        (citation?)>
<!ATTLIST paper        href CDATA "">
<!ELEMENT citation     (#PCDATA)>
```

Figure 1: XML DTD for a *Method Session Object*. Some parts of "`person`" intentionally left out.

service objects (MSO), and *method plan objects* (MPO). All objects instantiated through these classes are first class objects and 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 input drawn from bulk data providers. DSOs contain meta-data on the data used in computations, such as data format, type, location, size, source, and provider. *Access restrictions* (a notion similar to UNIX file permissions) that designate *who may access the data by which methods* are also encoded in the DSO (cf. Section 3.2). 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). DSOs designate input to computational methods, i.e., they are primarily used to represent computational method input and result data.

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

  MSOs represent abstractions for computational units accessible through the MMM infrastructure. Such units are denoted as *methods* in MMM parlance. MSOs contain meta-data on the encapsulated method, such as method 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 executions. MPOs tie MSOs together with DSOs for input and DSOs for output, i.e., they represents 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 2: XML DTD for a *Data Set Object*. Some parts intentionally left out.

The object model is implemented through a combination of complementing Web technologies, namely XML [7] and MetaHTML [10]. The *Extensible Markup Language* (XML) defined by the WWW Consortium (W3C) [7] is a data format for structured document interchange on the Web. In contrast to 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 [10] is a server side interpreted language for the generation of dynamic HTML pages. Amongst its features is the maintaining of user session state information, seamless access to server side applications, and fast ODBC database accesses.

For the MMM object model we use XML DTDs (Data Type Definitions) 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 objects (MSO, DSO, and MPO) is defined through XML DTDs. This permits the generation of all kind 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 plan is fully defined through an XML DTD. A particular plan instantiation is an XML document. Figures 1 and Figure 2 depict DTDs for MSOs and DSOs respectively.

2.2 The MMM architecture

The overall architecture of the MMM infrastructure is depicted in Figure 3. 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 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

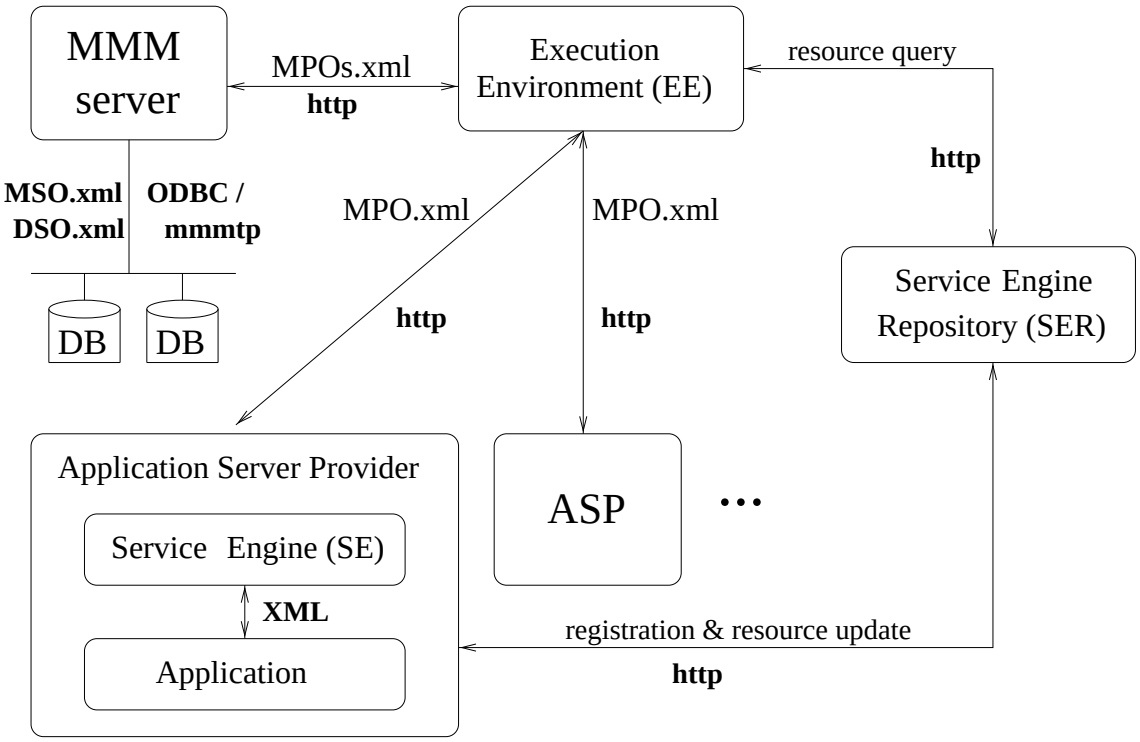

Figure 3: MMM infrastructure architecture.

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 over the available services, ensuring that no one service engine is overburdened.

The *service engine* (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 3 and Figure 6).

2.3.1 The inter–component communication 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 later modus of operation is applied if the requested computation takes more than a specific amount of time to complete.

2.3.2 Demand driven data access protocol — mmmtp

The mmmtp protocol permits service engines to retrieve data from the system database through our MetaHTML enabled web server. The protocol has been developed as an extension of http for several reasons:

1. Fast, direct, and highly optimized database access.
2. Demand driven data retrieval, i.e., data is only actually retrieved where and when it is needed for computations. For all other cases referential information and meta–data is manipulated (i.e., MSOs, DSOs).
3. Need for fine-grained access control mechanisms beyond what is currently available for http.

For mmmtp we define a new type of URL: `mmmtp` which allows retrieval of data and objects from the MMM database. 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 Database:

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

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

```
<?xml encoding="US-ASCII" ?>
<!ELEMENT user EMPTY>
<!ATTLIST user name CDATA "" id   CDATA "" password CDATA "">

<!ELEMENT gparas (appdesc, wrpdesc)>

<!ELEMENT appdesc (type, cline?, initial?, desc?, post?)>
<!ATTLIST appdesc name CDATA ""  path CDATA "" direct_access (yes|no) "">
<!ELEMENT desc (#PCDATA)>
<!ELEMENT post (#PCDATA)>
<!ELEMENT cline (#PCDATA)>
<!ELEMENT initial (#PCDATA)>
<!ELEMENT type (batch|inter)>
<!ELEMENT batch EMPTY>
<!ATTLIST batch success CDATA "Success Value" outfile CDATA "" infile CDATA "">
<!ELEMENT inter EMPTY>
<!ATTLIST inter prompt CDATA "">
<!ELEMENT wrpdesc (desc?)>
<!ATTLIST wrpdesc name CDATA "" sdesc CDATA "" procnt CDATA "0">
<!ELEMENT datadesc (inputs, outputs, source, libraries)>
<!ELEMENT pdesc (mapping)>
<!ATTLIST pdesc type CDATA ""  encoding CDATA "">
<!ELEMENT mapping (#PCDATA)>
<!ELEMENT inputs (pdesc*)>
<!ELEMENT outputs (pdesc*)>
<!ELEMENT source (pdesc*)>
<!ELEMENT libraries (pdesc*)>
```

Figure 4: XML DTD for a *Service Engine*. Some parts intentionally left out.

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. Figure 4 depicts the DTD of the service engine interface. Figure 5 shows a sample interface instantiation for the XploRe service engine.

```
<!-- xplore.xml Configuration file for the XploRer Application.
     Strings are assumed to represent XploRe lists. -->
<options>
  <user name = "Gerrit Riessen" id = "*" password = "*"></user>
  <gparas>
    <appdesc name = "xpl4b.exe" path = "/www/xplorer" direct_access = "yes">
    <initial>setenv( "xpl4lib", "/www/xplorer/lib" )\n</initial>
    <type>
      <batch success="0" infile="*standard-input*"outfile="*standard-output*"></batch>
    </type>
    <desc>XploRe Engine</desc>
    </appdesc>
    <wrpdesc name = "XploRe"
             sdesc = "XploRe Version 4 Batch Engine"
             procnt = "10">
    </wrpdesc>
  </gparas>

  <datadesc>
    <inputs>
      <pdesc type = "file" encoding = "*">   [...]  </pdesc>
      <pdesc type = "matrix" encoding = "*"> [...]  </pdesc>
      <pdesc type = "integer" encoding = "*"> ...
      <pdesc type = "double" encoding = "*">  ...
      <pdesc type = "string" encoding = "*">  ...
    </inputs>

    <outputs>
       <pdesc type = "file" encoding = "*">   [...]   </pdesc>
       <pdesc type = "matrix" encoding = "*"> [...]   </pdesc>
       <pdesc type = "integer" encoding = "*"> ...
       <pdesc type = "double" encoding = "*">  ...
       <pdesc type = "string" encoding = "*">  ...
    </outputs>

    <source>
      <pdesc type = "string" encoding = "*"> [...]    </pdesc>
    </source>

    <libraries>
      <pdesc type = "string" encoding = "*"> [...]    </pdesc>
    </libraries>
  </datadesc>
</options>
```

Figure 5: XML interface definition for the XploRe*Service Engine*.

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 6.

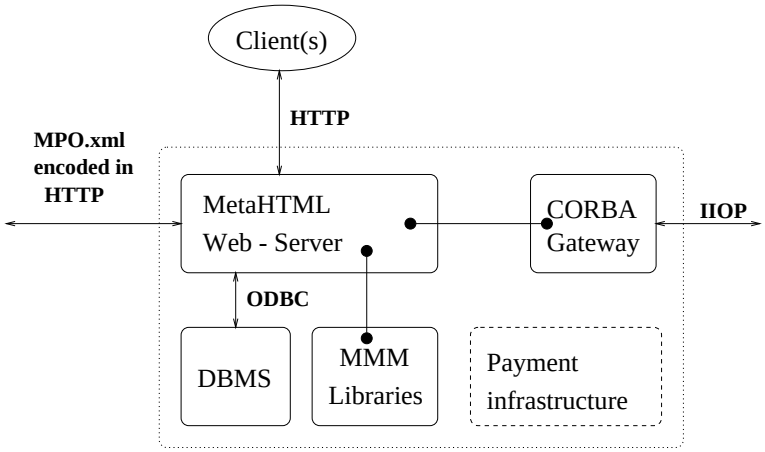

Figure 6: MMM server architecture.

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 costumers 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 3).

2.6 Gateway for server–side 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[1] integration of specialized libraries, such as LaPACK, BLAS, or other special pur-

[1] 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.

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

Figure 7 demonstrates the integration of a third party CORBA compliant payment system into an HTML form. 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.

Upon user submit the MetaHTML document is evaluated at the server site with form input data set in the package "`posted`" (cf. Figure 7). The first part of the document checks whether the user in-

```
<include header.mhtml>
;;; check whether page may be evaluated in this state and disable browser
;;; fwd./back buttons, re-load button actions
[...]
<set-var title="SET BUYER Payment Entry Form">
;;; import variable packages for this page
<session-import buyer buyer> [...]

;;; input value integrity, consistency, and error check
<when <get-var posted::PricePayed>>
    <set-var Error="<validPayment <get-var posted::PricePayed>>">
    <when <get-var Error>>
        <message The entered amount: ‘<get-var posted::PricePayed>’
                 is not a valid amount. <br>>
    <elsewhen>
        ;;; correct payment record entered: initiate payment
        <when <string-eq <SET_pay posted> true>>
            <redirect payment_success.mhtml>
        <elsewhen>
            <redirect payment_error.mhtml>
        </when>
    </when>
</when>

;;; Displayed HTML form
<show-message>
<layout::page><ptext>     [...]
<form>                    [...]
Amount due: <input type=text name=PricePayed size=5 maxlength=5>
[...]
<input type=submit name=create value="Submit Payment">
</form>
</ptext></layout::page>
<include footer.mhtml>
```

Figure 7: Example of an HTML document for a *buyer* entering payment information. The actual payment system integration is done within the implementation of the server–side included function `<SET_pay ...>` through CORBA DII invocations to the SET (Secure Electronic Transaction) package.

put is correct and consistent. Note, that the initial rendering (i.e,, first page access) of this document has variables "`posted::PricePayed`" not initialized. Its test "`<when <get-var posted::PricePayed>>`" evaluates to false and the corresponding code in the when–cluase is not evaluated (cf. MetaHTML programmers guide [10]).

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. For instance, the call to "`<SET_pay posted>`" (cf. Figure 7) makes use of this interface to access the payment system.

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 Security considerations

3.1 Overview of security issues addressed in MMM

Security has been one of the primary concerns in the design and development of the MMM infrastructure. As a platform for remote service integration and large scale electronic commerce deployment MMM requires strict security measures for several reasons:

- to protect sensitive data from being stolen, corrupted, and intentionally falsified in the transmission and at the remote side;
- to protect the cooperating systems from malicious use (abuse) by impersonators;
- to protect the cooperating systems from unauthorized use;
- commercial or national security concerns may require additional steps to preserve the privacy of the data transmitted (or of the encryption technology used).

To enforce these security requirements the following middleware functionality is needed:

1. *Server authentication*, i.e., remote site/server authentication, ensures the client application that it is truly operating on the intended site.
2. *Client authentication*, i.e., user authentication ensures remote site/server that an authorized client is interacting.

3. *Integrity*, i.e., non–corruption of data transferred. This prevents both malicious and false operation.

4. *Confidentiality*, i.e., data items transferred are encrypted. This prevents both malicious and false operation, as well as eavesdropping.

5. *Secure invocation* of methods from client application to remote services, routed (i.e., delegated) through a logging facility to gather "evidence" of "who" initiated an invocation "when".

6. *Non-repudiation* of invoked methods to ensure liability.

7. *Data security* to prevent sensitive or "expensive" data from being compomised at the site of computation.

Threat	**Required functionality**	**Technological solution**
impersonation	server authentication	SSL + Services of PKI
impersonation, unathorized use	client authentication	SSL
falsification, corruption	data integrity	SSL
eavesdropping, privacy	confidentiality	SSL
malicious use	secure invokation	logging of all user–system interactions, access control
non–accountability	non–repudiation	logging of all user–system interactions + PKI
compromize data	data security	fine grained access control + deferred data retrieval

Table 1: Overview of threats, middleware security functionality, and technological solution (PKI – Public Key Infrastructure; SSL – Secure Sockets Layer).

A detailed discussion of all mechanisms implementing these features would go far beyond the scope of this paper. Table 3.1 gives an overview that relates the above threats, required middleware security functionality, and technological solution available. A more detailed discussion of how we aim at guaranteeing "data security" is presented in the next section since it is fundamental to the model of operation propagated by the MMM infrastructure. Most other security issues raised are well understood and supported in part by most web and middleware platforms (e.g., MetaHTML, CORBA Security Service, Secure iiop, Secure Socket Layer).

3.2 A fine-grained access control scheme

The MMM infrastructure manages data and methods from widely distributed sources and providers. As methods are executed on the provider's platform, neither the platform is at risk by executing "unknown" code, nor the method code (i.e., algorithm) is at risk of being exposed, stolen, or compromised.

Note, that for byte–code approaches, such as the Java VM model, both risks apply, since byte–code is shipped to remote execution sites. To leverage these risks, techniques such as code signing [11].and code obfuscation [14] have been developed. Within MMM, however, we assume that the method code is executed on the provider's platform, such that these risks do not arise.

However, for data this is a different issue. Method input data has to be shipped to the computational site (i.e., to the method provider). It is therefore subject to exposure, at least, at the provider's platform, where the method is executed on the data. Note, that we are not referring to any risk of the data being captured over the communication link. These problems may be fully solved through cryptographic protocols that are common place to date. However, two threats remain. Firstly, data may be compromised by an infrastructure customer who seeks to obtain the data for purposes other than the "leased computations", and secondly, data may be compromised at the application server site where it is processed.

For data providers, whose only asset is the data itself, the exposure of the data to potentially untrusted third parties poses a major threat and severely limits the distribution of the data. Premium stock data, for instance, that has been collected, maintained, and updated over years is very hard to come by. For the model propagated by MMM this poses a severe limitation. Similar problems arise for sensitive, or simply personal, user and corporate input data to a computational service, offered through MMM (e.g., taxation, personal asset management, business logistics, financial risk and portfolio management, revenue and earnings balancing.)

An obvious solution would be to ship the method to the data provider. This, of course, would just reverse the problem and raise questions as we have discussed above and impose additional implications for the cases where method shipment is not possible. The MMM paradigm strictly disallows this mode of operation, while propagating the software leasing model discussed throughout this paper.

To address the first threat we have designed a fine–grained access control scheme that defines a list of

non–admissible operations for a DSO. This list has to be initialized upon data and method registry. It defines access and computational restrictions on data sets. It may optionally be provided for sensitive data sets, but it is strictly enforced upon presence. This access control scheme together with the demand driven data access design of MMM, i.e., data is uploaded as late as possible and to the computational site only (cf. Section 2.3.2) leverages the first data security threat.

The second threat, however, is much more difficult to solve. A theoretic solution, with proven guarantees of the data remaining unknown to the application server provider, is provided by Abadi and Feigenbaum [2, 1] and has become know as *secure circuit evaluation*. However, their algorithm is impractical for our scenario since it requires a huge amount of interaction between client and server. This problem can therefore only be addressed by a pragmatic solution. In the MMM infrastructure model only *trusted application servers* are admitted in the collective of service providers. A provider's computational site must fulfill a list of stringent security requirements to qualify as trusted. Other solutions also including *untrusted* application servers pose open research problems. We outline a possible approach based on transforming the input data in Section 7.

4 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 and data objects. This includes composition of method plans, selection of input parameters, execution support, and result notification functions. Moreover, for method providers the MMM UI provides bulk method entry functions that allow easy and quick registration of computational methods. 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 method space is supported by providing each user with a *Virtual File Space* (VFS), a notion similar to the Windows file manger. Technically, this view maps a flat RDBMS table space into a file and directory hierarchy combined with file permissions on each file

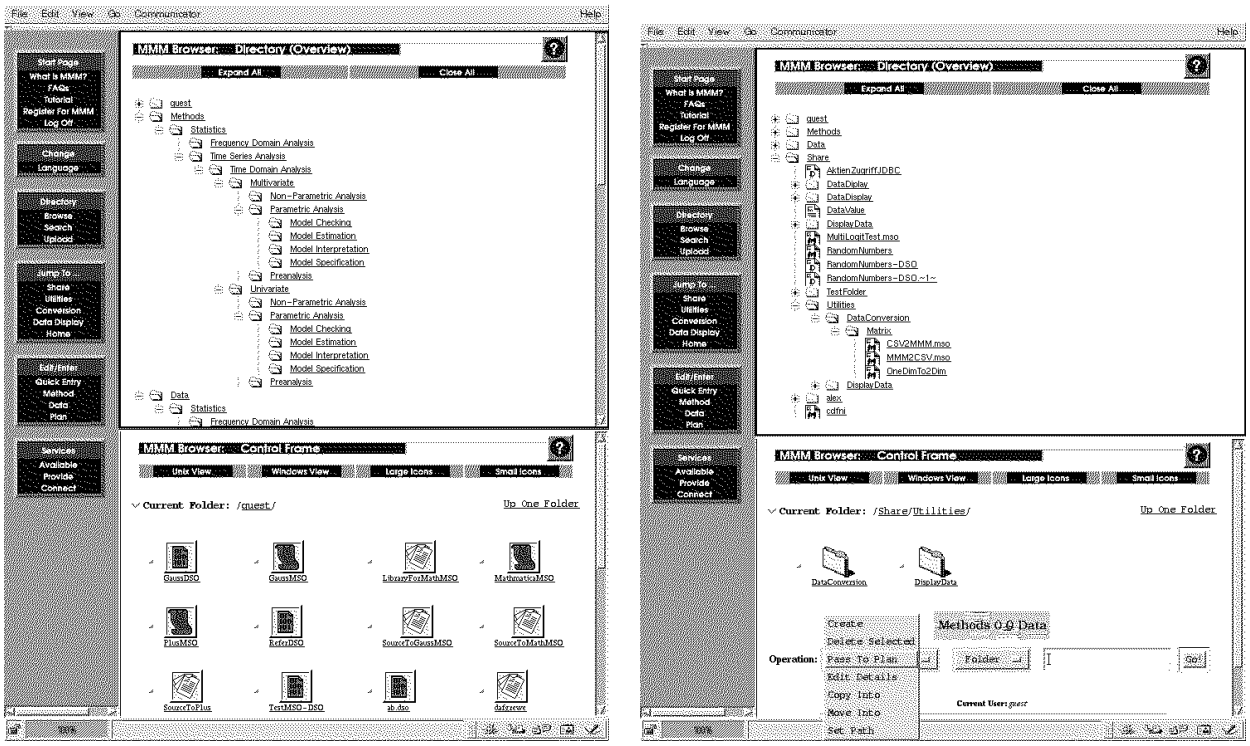

Figure 8: The MMM Virtual File Space.

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.

Figure 8 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.

5 User experience and implementation status

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) [13]. This application was described in detail in [12]. Second, we used MMM to facilitate the exchange of experimental software within

a distributed cooperation of optimization researchers [17].

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. Table 5 summarizes the application services that are currently managed and applicable for registered users through the MMM middleware. The system is available online at

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

and accessible via a guest account.

Provider	**Description**
Method services	
Mathematica	All purpose math package
Maple	Algebra package
Matlab	All purpose math package
Gauss	All purpose math package
Xplore	HU–Berlin statistical computing package
Scilab	INRIA all purpose math package
Octave	GNU high-level language for numerical computations
PSPP	GNU tool to analyze sample data
Progress	Mathematical tool University of Tübingen
LApack	Library of numerical software packages for linear algebra
BLAS	BLAS (Basic Linear Algebra Subprograms (vector and matrix operations)
ScaLApack	A Scalable Linear Algebra Library (special hardware reuqired)
Data source driver(s)	
JDBC	Based on Java 1.1.2, serves to integrate data bases
Visualization tool(s)	
Gnuplot	Package to produce graphical output
General purpose method service(s)	
gawk	Serves as general data format transformation tool

Table 2: Currently supported application services through the MMM infrastructure. *Italized services* are integrated at present.

The realization of a payment scheme based on available Internet payment technology is work in progress. We are also currently prototyping the security features outlined in Section 3.

We continue with a sample user–system interaction scenarios of defining and completing an MSO. Figure 9 shows the XML editor entry panel for an MSO. The page shows the attributes and elements to

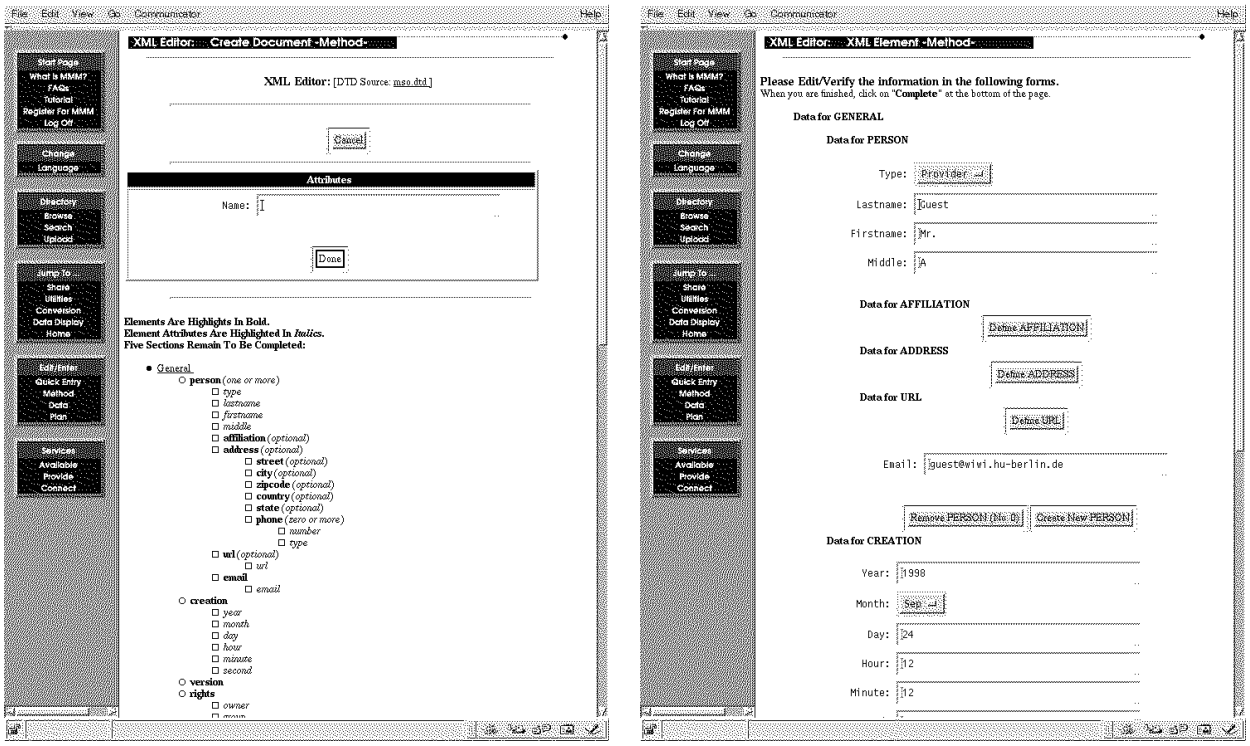

Figure 9: MSO entry panels, to be completed by the user when checking in a method.

be completed by a user who registers a method. The panel links to an HTML form to enter the specific details, this is shown in the second figure.

6 Business cases and related approaches

There are two driving forces for this kind of technology: (i) the algorithmic complexity required in certain application domains; and (ii) the size and price of today's software packages. We discuss these two aspects in turn.

Algorithmic complexity is a typical characteristic of many applications in finance and decision support. Finance applications typically involve complex statistical and econometric computations for data analysis, time series forecasting, etc. Decision support systems usually encapsulate sophisticated operations research algorithms to solve large optimization problems.

Several vendors have recognized this need and offer method bases for particular applications. Olsen and Associates[2], for example, offers a comprehensive set of software modules for financial market traders. The underlying technology consists of mathematical models to compute directional forecasts and timing

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

indicators and to provide trading support for applications in investment and risk management. The Argonne National Laboratories' NEOS service, [9] provides access to optimization software, for use by researchers world wide. In these two cases, 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 [5, 6]. DecisionNet[3] 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 *size and price factor* concerns application domains that are characterized by large, complex software packages that are expensive to license and operate. In those contexts one often observes that, once installed, the software is underutilized in the sense that customers use only a small fraction of the functionalities available. Enterprise resource planning (ERP) is a typical application domain with these characteristics. ERP software packages, such as SAP R/3, are notoriously expensive and complex. Their installation requires major investments and the establishment of special task forces. Moreover, they often necessitate changes in the corporate structure itself, i.e., the customer enterprise has to reorganized to become compatible with the assumptions made in the ERP software. Many potential customers are reluctant to commit themselves to such large investments and restructuring operations, especially if they are only interested in a subset of the functionalities offered by the ERP vendor. They would, however, be interested to use those specific functionalities on a *pay–per–use* basis, e.g., by sending their data to some ERP service provider who is contracted to do the required analysis. This business model is increasingly discussed by ERP vendors ("software leasing") [8].

Another application domain with similar characteristics are geographic information systems (GIS). While not quite in the same price range as ERP systems, GIS are still relatively expensive to license and operate: a single license often costs several thousand US$, it requires powerful hardware to run on, and

[3] http://dnet.sm.nps.navy.mil

it takes considerable training on the customer's part to use the software in a productive way. For most commercial GIS, potential customers face an all–or–nothing choice. Either they invest a relatively large amount to get the license, the required hardware, and some training — or they do not, in which case they get nothing. Especially small and medium–size enterprises often have difficulties justifying the expenses associated with running a GIS. Similar to our argument above, we believe that many potential GIS users would become faithful GIS customers if they could do so at a smaller entry cost. Of course, the lower ticket price would not buy them the whole license indefinitely. But rather than putting a time limit on the license, as is typically done, vendors could limit the functionalities they offer. This could mean in particular that the vendor does not sell a classical system license but selected services that perform GIS–typical tasks at the customer's request. Typical services in this context include data retrieval and conversion, map production, map overlay, spatial access methods, statistical analysis (e.g., multivariate data analysis), and visualization tasks.

The Australian *SMART* project [3, 4] explores the idea of an Internet marketplace for GIS data and related algorithms in greater detail. SMART was developed as tool for use by the government to assist in county planning tasks and to simplify related administrative tasks. We are not aware of it being available to the general Internet community for use, or any attempts of making it available as a commercial Internet service. MMM, in contrast, is aiming at a generic middleware that supports a very broad range of applications.

7 Research questions and future work

Before these kind of software component leasing infrastructures will become commonplace, several critical research questions will still have to be answered.

First, the development of appropriate licensing and (micro–)payment schemes is still work in progress. It is crucial for the success of Internet marketplaces that service providers can be certain to collect fees from all customers that use their services (directly or indirectly). Systems like MMM are independent of any particular payment scheme. They can in principle be combined with any of the major systems. The main problem is economical, not technical: what prices are consumers willing to pay, such that it is still attractive for potential providers to enter the market?

Second, one needs ways to guarantee confidentiality for the input and output data transmitted. If the service in question requires the input data in its original form, only organizational measures are possible to identify trustworthy service providers that maintain confidentiality. In many cases, however, a service can be performed on some transformed version of the data. The traveling salesman problem is a typical example of this case: in order to solve a given problem P, one can first apply a series T of geometric transformations (such as rotations or scalings) to P, then ask the service provider to solve the resulting problem $T(P)$. Once the service provider has found a solution S of $T(P)$, one obtains the solution of the original problem P by applying T^{-1}, the inverse of T, to S. This way the service provider never has access to the input and output data in its original form. It is an open research problem to define problem classes for which this kind of transformation is possible, i.e., where $S(P) = T^{-1}(S(T(P)))$ or, more general, for which there is *some* transformation (not necessarily T^{-1}) to generate $S(P)$ from $S(T(P))$.

Third, there is the related problem of *certification* of services. How can we make sure that a service does what it promises to do, in the desired quality? Some providers may already have acquired a reputation in their respective field. Others, however, may have to submit proofs of their competence and trustworthiness before customers decide to use their services.

Fourth, the usage and configuration of services should not be too complicated. Many software vendors pride themselves on the *turnkey* nature of their systems: setup efforts are minimal, and one can start using the system shortly after purchase. This may not always be the case in a digital marketplace type of situation, where users have to select and combine the services they need before they can use them.

8 Conclusions

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` .

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 Brian Fox who has considerably contributed to the re–implementation of the MMM infrastructure through his MetaHTML Web development language and tool. We would also like to thank the members of the MMM team for their contributions to this paper.

References

[1] M. Abadi and J. Feigenbaum. Secure circuit evaluation. *Journal of Cryptology*, 2:1–12, 1990.

[2] M. Abadi, J. Feigenbaum, and J. Kilian. On hiding infomation from an oracle. *Journal of Computer nd System Sciences*, 39:21–50, 1989.

[3] D. J. Abel. Spatial Internet marketplaces: A grand challenge. In M. Scholl and A. Voisard, editors, *Advances in Spatial Databases*, volume 1262 of *LNCS*, Berlin/Heidelberg/New York, 1997. Springer-Verlag.

[4] D. J. Abel, V. Gaede, K. L. Taylor, and X. Zhou. SMART: Towards spatial Internet marketplaces. Technical Report SIS-1997-01, CSIRO, Canberra, Australia, 1997.

[5] 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.

[6] H. K. Bhargava, R. Krishnan, and D. Kaplan. On customized access to a WWW-based distributed network of decision support services. In T. X. Bui, editor, *Proceedings of the Third International Society for Decision Support Systems Conference, Vol. II*, pages 551–560, Hong Kong, 1995. International Society for DSS.

[7] T. Bray, J. Paoli, and C. M. Sperberg-McQueen (editors). Extensible markup language (xml) 1.0. Technical report, W3C, Feb. 1998.

[8] T. Busse. ERP oursourcing options coming to your neighborhood. *InfoWorld Daily News*, (1), Sept 1998.

[9] 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.

[10] B. J. Fox. *The Meta-HTML Reference Manual.* Universal Access Inc. URL http://www.metahtml.com, 1998.

[11] L. Gong and R. Schmemers. Signing, Sealing, and Guarding Java^{TM} Objects. In G. Vigna, editor, *Mobile Agents and Security.* Springer, 1998.

[12] 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.

[13] W. Härdle and J. Horowitz. Internet based econometric computing. Discussion paper, SFB 373, Humboldt-Universität zu Berlin, 1998.

[14] F. Hohl. Time Limited Blackbox security: Protecting Mobile Agents from Malicious Hosts. In G. Vigna, editor, *Mobile Agents and Security.* Springer, 1998.

[15] H.-A. Jacobsen, G. Riessen, O. Günther, and R. Müller. MMM — towards an infrastructure for emerging electronic commerce application. Submitted to ITEC'99., 1999.

[16] G. Riessen, H.-A. Jacobsen, and O. Günther. MMM — Middleware for Method Management on the WWW. Submitted to Web Engineering Workshop, 1999.

[17] 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.