Component leasing on the World Wide Web

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

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

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.

Keywords: service outsourcing, software renting, application hosting, MetaHTML, XML, CORBA

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., multistage computations by distributed processing units), multiple intertransactional 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.

* Also with Pole Universitaire Leonard de Vinci, F-92916 Paris La Defense Cedex, France.

© Baltzer Science Publishers BV
**Data provider.** Data providers publish data. These may be individuals who submit files or providers of online databases that deliver data automatically (push) or on demand (pull). Examples include online stock quote services, geographic information systems, and consumer pricing data services.

**Application service provider.** Application service providers (a.k.a. method providers) own the application (method service) being offered online. 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 and provide network access to it. ASPs and application service providers will often coincide.

**Infrastructure provider.** The infrastructure provider offers a framework that grants authorized users secure access to application server sites. 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 *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 service that matches their budget. As the budget increases, a new service can 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 or provider 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 providers to publish 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, as well as the 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. [14]. Initial design, implementation, and deployment lessons gained from this prototype are summarized in Jacobsen et al. [18] and have led to a complete re-design and re-implementation. A brief overview of the revised MMM architecture is presented in Riessen et al. [19] and Jacobsen and Günther [17].

This paper concentrates on a technical discussion of the MMM infrastructure, presenting the MMM object model and architecture. It highlights selected design issues, such as the integration of distributed object and Web technology, and the MMM server design (section 2). The keys to the success of the proposed leasing model are security considerations, which are summarized in section 3. User interface issues are discussed in section 4. Section 5 gives a detailed account of related work, presents a summary of the status of the implementation, and discusses a suitable business model. Concluding remarks and open research questions are discussed in section 6.

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 architectural design, discuss major system components, and communication protocols developed for the specific application context. We then discuss several 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 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 through 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 (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).

**Method Service Objects (MSOs).** MSOs represent abstractions for computational services accessible through the MMM infrastructure. Such services are denoted as methods in MMM parlance. MSOs consist of metadata on the encapsulated method, such as method provider, author, signature (i.e., input and output types), 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).
This object model is implemented through a combination of complementing Web technologies, namely, XML [6] and MetaHTML [10]. The Extensible Markup Language (XML) defined by the WWW Consortium (W3C) [6] is a data format for structured document interchange on the Web. In contrast to HTML, it allows the user...
Figure 2. XML DTD for a data set object. Some parts intentionally left out.

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. Its features include maintenance of user session state information, seamless access to server side applications, and fast ODBC database access.

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. Figures 1 and 2 depict DTDS for MSOs and DSOs, respectively. Figures 3 and 4 show method service object and method plan object instances, respectively.
Figure 4. XML DTD for a method plan object.
2.2. The MMM architecture

The overall architecture of the MMM infrastructure is depicted in figure 5. 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 intercomponent 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 techniques to minimizing data movement among computing resources and to maximize 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, and possible error conditions. 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 un-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 ex-

![Figure 5. MMM infrastructure architecture.](image-url)
tensibility 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: intercomponent communication, demand driven data access (a.k.a. mmmtp = MMM transport protocol), and database access protocols (cf. figures 5 and 8).

2.3.1. The intercomponent 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 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 model supported by the middleware. Table 1 summarizes the main

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGIN</td>
<td>Login with username and password.</td>
</tr>
<tr>
<td>EVALUATE-METHOD</td>
<td>Informs service engine that next call to the DATA operation will be to send a complete method (MPO) for evaluation.</td>
</tr>
<tr>
<td>DATA</td>
<td>Prepare to receive data (specifies type).</td>
</tr>
<tr>
<td>RESULTS-OF-METHOD</td>
<td>Prepare for sending execution results.</td>
</tr>
<tr>
<td>STATUS</td>
<td>Check execution status of service engines.</td>
</tr>
<tr>
<td>ADMIN</td>
<td>To administrate remote service engines.</td>
</tr>
</tbody>
</table>
commands of the iccp-protocol. iccp is implemented on top of http. Each iccp-enabled component supports a MetaHTML iccp-protocol engine.

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 fooobar from the MMM repository:

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

The 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).

The 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, but this goes beyond the definition of ftp. The mmmtp, therefore, extends ftp by removing the requirement that a file is the entity to be accessed.

The 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, send 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 DBMS 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, and what execution permissions need to be set to access its services. Figure 6 depicts the DTD of the service engine interface. Figure 7 shows a sample interface instantiation for the XploRe service engine. XploRe is a statistical computing environment developed at Humboldt University.¹

¹[www.xplore-stat.de](http://www.xplore-stat.de)
2.5. The MMM server

The MMM server manages all infrastructure operations. It performs the 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 8.

The server consists of the following subcomponents: The MetaHTML Web server, the database management system, the MMM infrastructure libraries, the CORBA gateway, and, in the future, the payment 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 5).

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\(^2\) 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.

\(^2\)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.
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, concerns the way in which they are deployed. Packages are script-driven

```
<include header.mh.html>

::: 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.mh.html>
    <elsewhen>
      <redirect payment_error.mh.html>
    </when>
  </when>
</when>
</include>

::: Displayed HTML form
<show-message>
<layout::page><p text> [...]
<form>
[...]
Amount due: <input type=text name=PricePayed size=5 max-length=5>
[...]
<input type=submit name=create value="Submit Payment">
</form>
</p text></layout::page>
<include footer.mh.html>
```

Figure 9. 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.
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 9 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 submission, the MetaHTML document is evaluated at the server site with form input data set in the package “posted” (cf. figure 9). The first part of the document checks whether the user input is correct and consistent. Note, that the initial rendering (i.e., first page access) of this document has variables “posted::PricePaid” not initialized. Its test “<when <get-var posted:: PricePaid>>” evaluates to false and the corresponding code in the when-clause 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 9) 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 for software leasing

3.1. Overview of security issues

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 during 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., noncorruption of data transferred. This prevents both malicious and
   false operation.

4. **Confidentiality**, i.e., data items transferred are encrypted. This prevents both ma-
   licious 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 compromised
   at the site of computation. This may require the additional use of encryption and
   transformation techniques, as well as adequate organizational means.

### Table 2
Overview of threats, middleware security functionality, and technological solution (PKI -
Public Key Infrastructure; SSL - Secure Sockets Layer).

<table>
<thead>
<tr>
<th>Threat</th>
<th>Required functionality</th>
<th>Technological solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>impersonation</td>
<td>server authentication</td>
<td>SSL + Services of PKI</td>
</tr>
<tr>
<td>impersonation, unauthorized use</td>
<td>client authentication</td>
<td>SSL</td>
</tr>
<tr>
<td>falsification, corruption</td>
<td>data integrity</td>
<td>SSL</td>
</tr>
<tr>
<td>eavesdropping, privacy</td>
<td>confidentiality</td>
<td>SSL</td>
</tr>
<tr>
<td>malicious use</td>
<td>secure invocation</td>
<td>logging of all user-system interactions, access control</td>
</tr>
<tr>
<td>non-accountability</td>
<td>non-repudiation</td>
<td>logging of all user-system interactions + PKI</td>
</tr>
<tr>
<td>compromise data</td>
<td>data security</td>
<td>fine grained access control + deferred data retrieval</td>
</tr>
</tbody>
</table>
A detailed discussion of all mechanisms implementing these features would go far beyond the scope of this paper. Table 2 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 IOP, 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 address these risks, techniques such as code signing [11] and code obfuscation [16] 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 application server 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 the risk of data being captured over the communication link. These problems may be fully solved through cryptographic protocols that are commonplace 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 does not support 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 is initialized upon data and method registry. It defines access and computational restrictions on data sets, while it is optional to declare such restrictions for a given data set, they are 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 – compromization at the application server site – 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 [1,2] and has become known 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 as 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.

A possible solution of this problem that would guarantee data security (confidentiality) for the input and output data transmitted involves admissible transformations of the input and result data (i.e., admissible in terms of the computations performed by the ASP). In many cases, 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 $U_T$ (not necessarily $T^{-1}$) to generate $S(P)$ from $S(T(P))$.

Independently of this open research problem, it is often possible for a *particular method* to specify corresponding transformations $T$ and $U_T$, such that $S(P) = U_T(S(T(P)))$. While the specification of these transformations is sometimes obvious, it often requires a deep knowledge of the method in question. We, therefore, believe that the application service provider is the most likely source to provide this information. We are currently working on an extension of the MMM XML metadata format to allow service providers the specification of $T$ and $U_T$ for any given method.

4. **MMM user interface**

As already indicated, a crucial aspect concerning the infrastructure concerns its usability. In principle, any Internet user can join MMM as a provider. In order to create
strong network externalities, however, it is important to quickly attract a large number of service providers. Thus, it has to be easy to register a given method. In most cases few service providers would be willing to spend more than a few hours on registration.

The MMM infrastructure supports service providers by providing an interface that is based on standard technology (Web browser, HTML, XML, MetaHTML) and that is relatively easy to use. Simple methods can be registered in order for potential users to find, evaluate, and run the service. Depending on the nature and complexity of the service, entering this metadata may require up to several days. More work is needed to facilitate this registration process even further (e.g., by providing templates).

Figure 10. The MMM virtual file space.
Users of services are supported by a variety of other tools. On the one hand, MMM offers a specialized search engine that supports users in finding the right method for a given problem. On the other hand, applying a method to a given data set typically takes no more than a few mouse clicks. Essentially, it only takes entering the input data (directly or via a URL) and specifying the mode of payment (typically, credit card or some micropayment system).

For the MMM administrator, we provide administration tools for managing and monitoring user actions (e.g., monitor active user sessions).

Figure 11. XML editor entry panels, to be completed by the user when checking in a method.
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 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.

Figure 10 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 allow 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.

We conclude with a sample user-system interaction scenarios of defining and completing an MSO. Figure 11 shows the XML editor entry panel for an MSO. The page shows the attributes and elements to 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 figure 12.

5. Applications 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) [15]. This application was described in detail in [14]. Second, we used MMM to facilitate the exchange of experimental software within a distributed cooperation of optimization researchers [13].

To verify the new architecture described in this paper, 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 still 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 3 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.

More generally speaking, 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 and time series forecasting. Decision support systems usually encapsulate sophisticated operations research algorithms to solve large optimization problems.
Table 3
Currently supported application services through the MMM infrastructure. *Italicized services* are integrated at present.

<table>
<thead>
<tr>
<th>Provider</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mathematica</td>
<td>All purpose math package</td>
</tr>
<tr>
<td>Maple</td>
<td>Algebra package</td>
</tr>
<tr>
<td>Matlab</td>
<td>All purpose math package</td>
</tr>
<tr>
<td>Gauss</td>
<td>All purpose math package</td>
</tr>
<tr>
<td>Xplore</td>
<td>HU-Berlin statistical computing package</td>
</tr>
<tr>
<td>Scilab</td>
<td>INRIA all purpose math package</td>
</tr>
<tr>
<td>Octave</td>
<td>GNU high-level language for numerical computations</td>
</tr>
<tr>
<td>PSPP</td>
<td>GNU tool to analyze sample data</td>
</tr>
<tr>
<td>Progress</td>
<td>Mathematical tool University of Tübingen</td>
</tr>
<tr>
<td>LAPack</td>
<td>Library of numerical software packages for linear algebra</td>
</tr>
<tr>
<td>BLAS</td>
<td>BLAS (Basic Linear Algebra Subprograms (vector and matrix operations))</td>
</tr>
<tr>
<td>ScalAPack</td>
<td>A Scalable Linear Algebra Library (special hardware required)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data source driver(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC</td>
</tr>
<tr>
<td>Based on Java 1.1.2, serves to integrate data bases</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Visualization tool(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gnuplot</td>
</tr>
<tr>
<td>Package to produce graphical output</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General purpose method service(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gawk</td>
</tr>
<tr>
<td>Serves as general data format transformation tool</td>
</tr>
</tbody>
</table>

Several vendors have recognized this need and offer method bases for particular applications. Olsen and Associates, 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 indicators and to provide trading support for applications in investment and risk management.

Commercial online application service providers are now going to market. Startup 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 [8] provides access to optimization software for use by researchers worldwide. 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 spe-

3 http://www.olsen.ch.
cific 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 [4,5]. DecisionNet4 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 be 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”) [7].

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 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 conver-

4 http://dnet.sm.nps.navy.mil.
sion, map production, map overlay, spatial access methods, statistical analysis (e.g., multivariate data analysis), and visualization tasks.

The Australian SMART project [3] explores the idea of an Internet marketplace for GIS data and related algorithms in greater detail. SMART was developed 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, nor any plans of making it available as a commercial Internet service. MMM, in contrast, is aiming to be a generic middleware that supports a broad range of applications.

The ultimate goal of the project is the implementation and maintenance of an electronic market for application services. Eventually, posting and using such services should be no more difficult than posting or reading a Web page. If this were true, software and service providers could replace or complement the classical business model of software licensing by a “leasing” model: customers do not obtain a license anymore, neither do they have to install the software on their local machine. They rather send their data to the provider's server, which performs the desired computations and sends the results back to the customer. The customer then pays for the service. Payments are shared between three parties:

1. The (MMM) infrastructure provider (transaction fee).
2. The application service provider (software usage fee).
3. The application server provider (system usage fee).

Let us consider a simple example. John, a Ph.D. student, has developed a complex mathematical method for the analysis and forecasting of stock quotes. The method has been implemented in Matlab. First experiments yield encouraging results. John may decide to make his method publicly available, i.e., to become a service provider. He may use MMM to register his method; as a result it becomes part of the MMM marketplace and it will be indexed by a variety of search engines. On the other hand, there is Susan, a medical doctor, who is always interested in new techniques to optimize her private portfolio strategy. Browsing or searching the Web (e.g., by using a mainstream search engine), Susan one day comes across a link to the MMM marketplace, in particular its large selection of financial analysis and forecasting services. Using MMM's special method search engine, Susan finds John's method. With a few mouse clicks, Susan can test this method on some of her own portfolio data. Because Susan uses a standard format to manage her portfolio (e.g., Excel or Yahoo Finance), required data conversions are performed automatically by the system. The portfolio data is sent to the server running John’s method. After the computation has finished – which may take seconds, minutes, or even days – the results are sent back to Susan. Susan pays per credit card (or, in the future, using special micro-payment techniques). The payment is shared between John and the MMM infrastructure providers. John may have to forward parts of his share to Matlab because the transaction involved a somewhat unusual use of his Matlab license (the so-called license multiplexing).

The objective of the MMM infrastructure is to implement a model where software may be deployed as a rentable good. Consumers use the rented application, as they
would use a properly owned system, but do not engage in programming activities to put together a new application. This is in contrast to infrastructures such as Legion [12] and Globus [9] which aim at providing a virtual computer to its users. The virtual computer incorporates heterogeneous computing resources worldwide. However, the user deploys this virtual computer much like a desktop computer in writing and running applications. Legion [12] aims at providing an infrastructure for a single, worldwide, virtual computer, incorporating gigabit networks and connecting high-performance machines and workstations for solving complex problems. Globus [9] constitutes a set of software components designed to support the development of applications for high-performance distributed computing environments. While it certainly would be possible to implement the described software deployment model with these infrastructures, their primary focus is on providing a virtual computer with transparent resource management, and not on renting applications via the World Wide Web.

6. 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 metadata. The MMM platform is fully implemented and accessible on our WWW site: http://meta-mmm.wiwi.hu-berlin.de.

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

First, the development of appropriate licensing and (micro-)payment schemes is still 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, 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.

Third, 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.

Acknowledgements

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 for his substantial 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
