PROJECT DESCRIPTION

1 Introduction

In the last five years we have seen a tremendous increase in network-based computing on the Internet. The popularity of the Internet as a source of dynamic information accounts for most of the increase in search-based systems using text-based information retrieval techniques for matching and indexing information sources. As the need grows for more on-line services for scientific computing on the Internet, the use of a persistent data representation for scientific applications on the Web becomes vital.

To enable the exchange of data between disparate systems on the Internet, a universal data representation is necessary. This has been recognized as one of the key problems by many groups working on protocols and data formats (see e.g. [12, 2, 1]). The choice will ultimately affect the overall development of scientific applications, because the implementation of scientific applications is very much constrained to the type of data structures employed.

The Internet is also used as a network in small-scale distributed computing experiments. Ambitious plans are materializing making global distributed computing on the Internet feasible (Globus [9] and the Grid [10]). These component-based systems provide infrastructures for building distributed scientific applications from separate components.

Despite the efforts, there are still two major hurdles to be overcome in application development for (distributed) scientific computing such as large-scale numerical applications: uniform model development (algorithmic functionality) and data interoperability (persistent data representations).

Large-scale numerical applications are often build from separate modules. Since these modules are integrated, their degree of sophistication has to be high in terms of data structure compatibility and algorithmic selection. Therefore, uniform model development is a primary concern of numerical application integration in general. A recent article in Science titled "Research Council Says US Climate Models Can't Keep Up" [3] addresses this problem. US climate models have problems with regard to both hardware and software: there is a lack of supercomputing power and the absence of a coordinated strategy for building models. This results in a delay of integrating model components such as atmosphere, oceans, and chemistry, because of model incompatibility.

Even in the presence of a paradigm for uniform model development, data representation is important because it is a complementary problem. A persistent data representation would allow integrated computing environments to be extensible and compatible. Scientific data comes in many different forms, and numerical data formats and data structures are often application dependent. Also the units of numerical computation often differs between applications on spatial and temporal scales. For example, in climate models the ocean and atmosphere models differ in discretization, spatial scales, and time evolution rates. This severely hampers to interchangeability of data between these disparate systems.

Component-based systems are powerful tools for solving scientific problems using distributed computing, but the paradigm lacks the most important element the Internet offers: resource discovery and dynamic data.

The potential of data and resource discovery on the Web for scientific computing is huge. Consider the possibilities one could exploit in creating simulations based on dynamic data available from Web services. Current simulations often operate on static datasets and data sources. Many simulations would produce more accurate results if they could access dynamically changing data from other sources. The discovery of new sources of information and the use of dynamic data on the Web is not exploited in scientific computing.

The proposed paradigm for distributed scientific computation defines a set of tools (problem-solving environments, compilers) that relieve application developers from data structure design issues and setting up a distributed system. More specifically, the proposed research addresses the following issues:

- The problems associated with representing scientific data in XML (extensible markup language). As an object representation, XML is a promising new avenue toward effective data interchange. This requires the use of DTDs (document type definitions) to be used as data structure specifications in the context of scientific applications;

- The design of a problem-solving environment for the automatic synthesis of XML application programming interfaces (APIs) and DTDs for scientific applications;

- The adaptation of D'Agents [22, 8] to itinerative agents that can be programmed to move to resources identified by a "functional signature" thereby acting as XML-based object request brokers between scientific applications;

- The use of a lambda-calculus like language for defining the semantics of services as "functional signatures";

- The symbolic processing necessary for a problem-solving environment to generate APIs and DTDs for XML data interchange, and the symbolic processing of lambda-calculus like expressions required in searching, indexing, and matching services identified by functional signatures,

- The development of data and functional proxy servers. Data proxy servers maintain connections established by mobile agents between applications and optimize the network traffic. Functional proxy servers forward agents to the sources described by functional signatures.

- Security problems associated with distributed computing on the Internet in general.

The technical details of each of these issues will be described in the following sections.

1.1 Related Work

Component-based systems are powerful tools, but the current approaches to creating these type of distributed computing environments require a central form of control. Although not always viewed that way, the central control is necessary to

- enable applications in distributed systems to be continuously aware of (or connected to) the other applications in the environment, for example through table lookup in a centralized registry;
- manage and broadcast the data structure definitions to the applications involved in exchanging information;
- employ a form of transaction control to roll back the state of a distributed application when links and/or machines go down.

For example, the component-based distributed computing environments [19] consist of collections of tools that use the Internet to exchange information. These systems are extensible, but only by registering new services explicitly. These systems are comparable to distributed computing environments, except that they use the Internet instead of proprietary networks. In effect, they provide a cheap solution to network-based computing. Network traffic in the environment is restricted to access the registered applications only. Unrelated network activity can impact the performance of the distributed system.

Once these component-based systems are operational, it is not expected that data formats change over time. The problem is that distributed systems build from applications written in different programming languages use object request brokers or Java wrappers for data interoperability. The data structures are basically hardcoded in the applications.

Well known examples of component-based projects in the context of scientific computing are the component project [11], Globus [9], the Grid [10], Information Power Grid (IPG at Nasa) [18]. Systems that offer mobile agent systems for distributed computing are IKinetics [14], Java Voyager [16], Java Enterprise beans, IBM Aglets [7]. The component-based systems and agent systems adopt a centralized approach.

Java-based agent systems typically adopt Java RMI. Transaction control is an important issue when using Java-RMI based agents. When network connections go down, the system needs to be able to roll back. Java Voyager is exceptional as it enables other means of communication (e.g. CORBA). Java Voyager also uses proxies to forward messages from agents to other mobile agents that moved. The forwarding of messages allows agents to move without interrupting inter-agent messages.

Mobile agent systems that do not rely on RPC of Java RMI are for example D'Agents: secure mobile agents [8] and Java-to-go: mobile agents and agent-based applications for itinerative computing [17].

Another approach is used in Mstreams [4]. In this system agents are attached to an Mstream (Mobile Stream) that acts like a channel or bus to exchange messages between the agents. Agents are event handlers. An Mstream can move, and agents can detach and attach themselves to Mstreams dynamically. Mstreams supports the "mobile-server" model of mobility in contrast to the "mobile thread" model where migration can happen anywhere. However, the placement of an Mstream is performed from a central point of control.

2 Proposed Research

A centralized approach is not flexible in that it cannot easily cope with change of services provided on the network. New network services may become available that offer data using different representations. The proposed decentralized mobile agent system cannot rely on RPC methods or the representation of data by Java objects. Data exchange using Java objects assumes a centralized approach, because the corresponding Java classes have to be distributed among the collaborating applications. With mobile agents for itinerative computing, connections may close anytime or the network bandwidth may dramatically change over time. Itinerative mobile agents are less vulnerable to network hazards and do not necessarily need transaction control.

Also the element of resource discovery has to be supported in a flexible environment for distributed scientific computing. This requires the matching and indexing of semantic information represented by "functional signatures" describing the behavior of the application. These signatures can be searched for and matched by itinerative agents looking for specific services to solve a problem.

2.1 Resource Discovery and Data Interoperability

Search engines are the means of choice for Internet users to find new information by discovering and navigating new sites. Web crawlers and mobile agents can navigate the Internet to collect, combine, index, and discover new sources of information. As such, Web-based "computing" differs from distributed computing in that the environment allows for the discovery of new sources. This important feature of the Web distinguishes it from ordinary local network-based computing in which applications and data are managed from a centralized approach.

To enable resource discovery in scientific computing, a uniform persistent data representation and a formal specification of the provided services are required. Symbolic processing techniques are required for matching and indexing services. A text-based search and indexing mechanism based on ontologies and keywords is insufficient for matching searches for functions and algorithms.

In contrast to most of the searches done on the Web by non-scientific applications, a search for a scientific computing service is very much determined by its exact matching of data structures and algorithmic functionality provided by the service. Web technology for locating, indexing, and retrieving static text-based documents is well developed, although certainly not ideal. Object request brokers, as defined by CORBA for example, can in principle broker sessions between distributed components but the matching mechanisms are based on ontologies and keyword constructs that appear to be fragile when scaled to large heterogeneous networks [22]. Discovery can only be made possible when services have a specific and detailed functional description that can be searched for and matched. A functional description consists of a collection of Abstract Data Type specifications (ADTs) that encapsulate data types and algorithmic functionality, much like class specifications in object-oriented systems.

The handling of the same data by different systems requires the definition of persistent representations for that data, and corresponding interfaces to access and use that data. A representation of persistent data must inevitably inter-operate with humans or mechanisms that use other persistent representations [14].

- *Syntactic Data Interoperability:* Implemented by a translator. Data is simply transformed from one persistent representation to another, typically in discrete (or batch) operations.

- *Semantic Data Interoperability:* This transforms data from one persistent representation into an implementation of an interface. This requires both transformation of the data into a new representation and the representation's interface implements the semantic behavior corresponding to the new persistent form of the data.

Achieving semantic data interoperability is often described as "legacy integration", or "legacy wrapping" and is a universal recurring problem which up to now has typically been solved by ad-hoc means. In true semantic data interoperability, an object's representation is translated into another representation depending on the intended use of the object. For example, a satellite image's representation is translated into a set of parameters describing the percentages of land use reflecting agricultural development. Clearly, the data is the same (although information may be lost in the translation process) but its intended use differs.

The interface of the data representation in the form of an application programming interface (API) determines the semantics of the communicated objects for semantic data interoperability. Current component-based systems mainly use Java RMI to communicate serialized objects remotely between applications on the Internet. The communicated objects have a Java interface that defines the semantic data interoperability. To move data from a non-Java application to Java (as in syntactic data interoperability), Java wrappers have to be written for each application that translate data into Java objects.

Java class definitions (or Java interfaces) of an application have to be published and made available to all the other applications that need to communicate. The class definitions are necessary to write the Java wrappers that handle the objects. In this respect, component-based systems lacks extensibility as the classes of the communicated objects have to be specified in each wrapper in advance.

2.2 XML

Recent Web developments have brought an important innovation in data representation for semantic data interoperability: the XML (extensible markup language) standard. As an object representation, XML is a promising universally understood representation. While HTML (hypertext markup language) is primarily intended to be a generic markup language for Web page layout, in contrast XML is content-based. XML has an important benefit over HTML: since it is content-based it allows inter-application data exchange while the layout of XML by Web browsers can be done automatically through XSL. XSL is a specification of a translation of an XML object into HTML. When browsers can access the XSL, it enables them to depict the XML object. Furthermore, XML can contain meta-data, information about the XML object itself.

The use of HTML and XML is not necessarily restricted to Web pages. XML documents can be representations of objects communicated between applications. XML separates data from layout in a way that hides both the data source and the formatting from one another. With XML, an application can locally consume, create, or modify data from a logical data structure that is independent of all back-end implementation. Multiple data sources may feed data into a single type of XML structure, allowing seamless integration of disparate systems.

A Document Type Definition (DTD) is a standardization of the XML object format and it enforces rules on tag naming, attributes, and overall hierarchy of the XML object representation, Using DTDs, applications are able to exchange objects in XML without having the risk of failing to recognize objects. It allows to create data structures that can be shared between and among disparate and otherwise incompatible systems. DTDs are so powerful because DTD enable a translation of one XML object into another. This means that objects with different XML representations are interchangeable through their DTDs.

The Document Object Model (DOM) provides an Application Programming Interface (API) for HTML and XML documents. DOMs are hierarchical representations of XML. An instance of XML is purely textual and lends itself to be portable between applications, while the corresponding DOM of an XML instance is a tree data structure that is used internally by an application (eg. browser). With the DOM, programmers can build documents, navigate their structure, and add, modify, or delete elements and content.

The DOM interfaces are an abstraction in that they are a means of specifying a way to access and manipulate an application's internal representation of a document DOM is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. A "binding" of the DOM to a particular programming language provides a concrete API. Currently, several libraries that handle XML DOMs are being developed for a number of programming languages. This would enable applications to manipulate XML as data structures. However, in designing an API for XML and HTML documents using the DOM, programmers still need to implement the concrete methods to access and manipulate document contents.

2.3 Scientific Computing and XML

Scientific computing has not yet benefited much from availability of Internet services and the existence of HTML and XML. However, there is a huge potential of XML as a persistent data representation to create software environments in which disparate scientific systems running on different machines are able to share many different types of data.

Existing scientific applications are developed in non-Java programming languages, because of efficiency considerations. For scientific applications written in different programming languages, a portable data representation between these applications is essential to create a component-based system. Although XML is a promising new data representation, a number of significant problems have to be solved for a practical use of XML in symbolic and numerical applications.

First, the practical use of XML for data exchange in programs is alleviated by using the DOM. However, application developers still need to implement APIs to access the data from

DOMs. Tool support (compilers, generators, problem-solving environments, and libraries) relieve programmers from the burden of API implementation.

Second, the standardization of XML tags for mathematical constructs, symbolic expressions, and numerical data is necessary. In part, this needs the development of the appropriate DTDs. From experiences in programming language design, an orthogonal set of primitives as defined by DTDs and a simple set of composition XML tags should allow the construction of arbitrarily complex data structures.

Third, it is widely known that directed acyclic graph (DAG) representations of symbolic expressions are crucial for symbolic computing environments. Trees alone require too much storage space and time to be manipulated. The problem is that a naive use of XML using the DOM for symbolic expressions leads to trees instead of DAGs. Unfortunately, proposed standards such as MathML do not take this important issue into account.

Fourth, numerical data exchange in XML requires a textual representation of numeric values. Meta data on the precision and format should be included in XML. For example, the Multi-Protocol (MP) library [12] for symbolic and numerical data exchange can be used as a protocol.

Fifth, in a distributed computing environment security threats have to be taken seriously into account. Bogus services may be set up to intercept data and unreliable data can be created that is tampered with by intruders.

2.4 Representing Persistent Data in XML

Scientific data may consist of symbolic expressions and data structures or numerical data. Care has to be taken for representing graph-like data structures in XML. For example, computer algebra systems (CAS) rely on directed acyclic graph (DAG) representations of symbolic expressions. For the exchange of symbolic data between scientific applications it is crucial that objects have to be translated to XML in DAG form. A naive use of XML and the DOM leads to trees, because of the hierarchical layout of XML in the DOM. Fortunately, XML is flexible enough to device a data structure that can refer to itself by using an indexing scheme where the indices refer to the objects represented in XML.

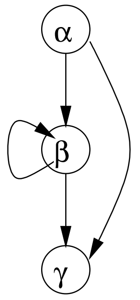
However, it is a well-known problem that the inspection of data structure declarations alone cannot reveal the exact usage of the data structure in question. Consider for example the tree data structure defined in C:

```
struct Node { Value val; struct Node *left; struct Node *right; }
```

The data type declaration above is no exception to this rule. Although the declaration suggest that it is a tree (because of the use of `left` and `right` field names), there are no limitations for using this data structure as a graph in which a node is referred to by more than one `left` or `right` pointer from another node.

Consider for example the following a graph data structure declaration depicted in Figure 1. A node has a value and a list of outgoing edges to nodes. An instance of this data structure can be converted to XML by using indexing as is shown on the right. Note that there is almost a one-to-one correspondence between the type declarations in C and the XML tag naming conventions.

```
struct Node {
  Value val;
  struct List *list;
};
struct List {
  struct Node *head;
  struct List *tail;
};
```



```
<node idx=1><value>α</value><listptr>1</listptr></node>
<node idx=2><value>β</value><listptr>2</listptr></node>
<node idx=3><value>γ</value><listptr>0</listptr></node>
<list idx=1><nodeptr>2</nodeptr><listptr>3</listptr></list>
<list idx=2><nodeptr>3</nodeptr><listptr>4</listptr></list>
<list idx=3><nodeptr>3</nodeptr><listptr>0</listptr></list>
<list idx=4><nodeptr>2</nodeptr><listptr>0</listptr></list>
```

Figure 1: Graph Type Declaration, Example Graph Instance, and XML Representation

In our approach we use the lowest-level data structure type definitions to construct an XML representation of objects when the data structure in question is application-specific. It does not matter for XML whether the data structure is a graph or a tree. But the applications handling the data may depend on this information. In such cases, the XML has to contain meta-data to describe properties of the data.

The intrinsic feature of XML to be able to translate one XML object into another using their DTDs is very powerful. Even when applications use different type declarations for the same object, the objects are interchangeable through their DTDs when the overall structure is similar. In this sense, XML truly supports semantic data interoperability.

Numerical data representation in XML can be implemented similarly to the Multi-Protocol [12] implementation for data interchange. The numerical precision, dimensions/units, and data structures such as the standard sparse matrix representations is included as meta-data in XML.

The mapping of application-specific data structures to XML and vice verse is comparable to object serialization in Java. Except that object serialization in Java allows for the export of methods besides object data, which is not directly supported by XML.

2.5 Problem-Solving Support for API Development

A problem-solving environment (PSE) with a library of commonly used data types in scientific applications is a powerful tool to support the development of an API using the DOM. A programmer has to go through the tedious task of implementation an API for the XML data representations discussed in the previous section. In addition, from data type declarations alone the properties of the data structures cannot always be inferred.

The proposed PSE for API development in scientific computing offers the following features:

- Symbolic processing of functional signatures by a built-in computer algebra system for matching and indexing services.
- A library of APIs for common data structures used in scientific applications. For example, data structures for a wide variety of sparse matrix representations are made available in a database. The PSE offers the choice of data structures to be included in newly developed applications. The DOM as an API together with the appropriate data

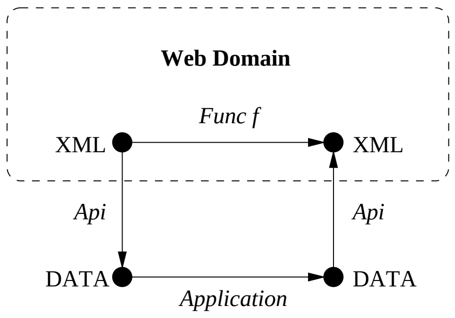


Figure 2: Abstract Functional Signature f and Concrete Application Implementation

structure declaration in a given programming language are automatically generated by the PSE.

- A parser/compiler for Fortran, C, C++, Java data structure and class declarations in source codes. The data structure and class declarations of an existing application are parsed by the PSE. An internal type structure is build from the declarations and then mapped to an XML representation. This is possible by using the DOM (implemented in the specific language) and additional software synthesis necessary to implement the access to the XML document that represents the objects. Only the input/output data structure declarations of an application have to be compiled into APIs, because the application data interchange will take place with these data structures.

- A generator of APIs for application XML input/output using the DOM. The generator creates code that implements the access methods to extract the application-specific data structures from XML. Also, XML output code is generated. The generated API codes can be readily hooked into the application. The mapping of internal data structures to XML was discussed in 2.4.

All of these PSE functions can be readily implemented in our problem-solving and computer algebra system CTADEL [21]. CTADEL has strong capabilities for symbolic processing by pattern matching and software generation as has been demonstrated in generating efficient parallel code for a weather forecast system [20]. Furthermore, the system includes HTML/XML generators and parsers based on the PiLLoW library [6] enabling the manipulation of HTML/XML as symbolic expressions. The system currently has a Fortran code generator. The adaptation to C and Java code generation for application programming interfaces will not pose major problems.

The main objective of the PSE is to relieve the programmer writing APIs for handling XML. A user of the PSE can hook a generated API in the I/O part of an application. Through this interface, the application becomes a server. Application data "lives" in the Web domain, as is illustrated in Figure 2. The API converts XML input to the native data structures of the application and application output data is converted back to XML.

The PSE technology in this project differs from PDE-based PSE research efforts e.g. PSEware [13], Ellpack [15], and SCINAPSE [5] in that the PSE tasks are specifically targeted towards data integration.

To identify the service provided by the application, a functional signature *Func f* specifies the functionality of the application. The diagram in Figure 2 depicts the isomorphism relation resulting from a virtual operation on XML but actually implemented by the application and its API.

2.6 Mobile Agents for Itinerative Computing

Agents are software systems that can act autonomously and at high "semantic levels". These high semantic levels suggest human-like functionality in information processing tasks. Mobile agent systems such as D'Agents [22, 8] and Java-to-go [17] are agent-based applications for itinerative computing. Mobile agent technology permits dynamic, decentralized creation and execution of modules that can execute on remote machines and spawn clones for parallel computation.

In an itinerative application, a series of hosts are traversed in sequence by a single mobile agent or in parallel by a group of agents attempting to "solve" a problem by processing information at or supplying information to the local agent-handling entity [17]. Itinerant mobile agents are persistent in space and time. They can migrate within a network, docking on remove nodes should network conditions be unstable. This functionality is most valuable in volatile environments. An itinerant agent is given the freedom to transport and perform a variety of active computations at one or more remote agent servers, where an agent server is a networked process that provides the resources a agent needs to complete its goal. This is in contrast of a message-based transaction, where a message is limited to data or a directive (in the styles of RPC and Java RMI). The Java RMI toolkit adopts stub-based interface generation, similar to that of Remote Procedural Call (RPC) programming, to provide the appearance of objects being executed remotely. However, it suffers from client-server problems (for example, the client interface needs to be continuously connected to the server object).

In itinerative computing a series of hosts is traversed in sequence by a single mobile agent or in parallel by a group of agents attempting to solve a problem, applications should not only create data structures (objects) but also a set of commands to operate on the object. These commands define a scenario for solving a problem.

2.7 Object Request Brokerage With Mobile Agents and Proxy Servers

In a distributed computing environment object request brokers (as defined by CORBA for example) provide syntactic data interoperability. In contrast, the use of mobile agents as object request brokers exploits three important advantages compared to existing object request broker systems:

- Mobile agents can move to a large data resource as an alternative to moving the large data sets to a client [22]. The mobile agent performs a remote computation on the data and sends back only the relevant data products.

- Mobile agents are dynamic and can make decisions based on the properties of the current environment. For example, an agent can decide to establish connections to servers that have lower process loads among the servers that offer the same applications. This enhances scalability.
- Mobile agents can carry objects along for itinerative computing. Connections can be disconnected after agents have moved.

Mobile agents must be small-sized programs in order to be efficient. Since mobile agents for object request brokerage have very specific tasks to perform, their code size is limited. They do not have to carry global network and server information. This type of information is maintained by proxy servers that are assumed to reside on each machine an agent moves to. Data proxy servers have updated network information and maintain connections between applications and agents. Together, mobile agents and proxy servers act as object request brokers.

2.8 Functional Validation

Mobile agents for itinerative computing solve problems by traversing a sequence of hosts that offer specific services. The mobile agents establish the connections and the data proxy servers maintain the connections. To find a specific service, an agent must search for the functional behavior of the service. Functional proxy servers keep information related to the available services in tables using "functional signatures". Functional signatures provide the means of matching mechanism necessary to execute commands that represent services.

Current text-based search and indexing mechanisms are based on ontologies and keywords. This is insufficient for matching searches for functions and algorithms. A functional specification of the semantics of an algorithm using predefined constructs provides a solution to this problem. For symbolic and numerical applications, standard mathematical constructs and a functional notation borrowed from a strongly typed functional language is sufficient to define functional signatures.

Lambda calculus normal forms guarantee uniqueness of representation. A computer algebra system for symbolic computations (part of the PSE for designing APIs) can translate a lambda expression into a normal form. For example, suppose $solve(M, x) = y$ implements a solver $x = My$. An anonymous functional signature can be written in lambda calculus and may look something like $\lambda(M, x).(M^{-1}x)$. Since M and x are formal arguments of the lambda abstraction, their names are irrelevant in matching a search for a solver (eg. $\lambda(A, b).(A^{-1}b)$).

Besides lambda applications, operator properties have to be exploited to guarantee uniqueness of representation. For example, the functional signature $\lambda(A, b, x).(Ab+x)$ should be normalized be reordering the operands of the addition $Ab + x$ if necessary, since addition is commutative in this case. Normalization of functional signatures using associativity, commutativity, and the distributive rule is legal, although the application of these rules in codes actually changes the output. Functional signatures specify the semantics, not the implementation.

The OpenMath [2] standard can be used instead of lambda calculus to describe functional signatures. Recently, the "binding" concept was added to OpenMath. A binding in OpenMath is similar to a lambda abstraction in which the name of the formal argument used is immaterial to the meaning of the construct. This property of lambda abstractions and OpenMath constructs will prove to be very powerful in matching functions and algorithms by abstracting away naming details.

Data type information associated with formal arguments of a function and result object of the function provides an abstract interface. Suppose a solver only solves with sparse tri-diagonal real matrixes and real-typed vectors. The specification of the functional signature in lambda calculus is insufficient, as this function only operates on a subset of possible data types. This type information is provided by the DTD of tri-diagonal real matrices that are represented in XML. The DTD defines the valid XML representations of the data structures. A complete functional signature uses a lambda-calculus like notation with additional type information given by DTDs. DTDs are associated with the formal arguments and result of the operation. In this way, a service can be matched and indexed by matching lambda-calculus normal forms and checking for DTD compatibility.

Using lambda-calculus, a numerical computation service can be described in terms of its mathematical semantics as expressed in a lambda calculus like language (or OpenMath). However, some type of services cannot easily be described by mathematical constructs alone. Some types of services can only be distinguished by using some universally understood reference (eg. a satellite image provider). This requires a name space with global names identifying the services. For example, the satellite image provider can be referred to simply by *satellite*. In this respect, the function name is used instead of an anonymous lambda abstraction.

Functional signatures do not have to be written by users. Instead, a problem-solving environment can translate a user-defined scenario into a set of commands, each command being a functional signature of a service. The problem-solving environment adopts a library that is shared between multiple platforms so additions and to the library are immediately available.

A sequence of functional signatures can describe a process in which objects are transformed. As such, this sequence when given to a mobile agent leads to purely itinerative computing using mobile agents where agents move to resources (described by the functional signatures) to solve a given problem.

The lambda-calculus notation for functional signatures also supports higher-order programming, such as mapping and filtering of data by functions. For example, a matrix object may contain a command to apply a function on the matrix elements. Some higher-order functions can be predefined by agents, so agents are able to map functions, filter, and apply selections on data structures.

2.9 Security

Two security problems have to be addressed: bogus services and data manipulation by intruders. A digital signature can be embedded within the XML representation of an object to authenticate it source. The digital signature tells the source of the information and allows the verification of the reliability of the data in terms of filtering out bogus services.

The API can optionally encode the data using a public-private key system (eg. RSA). The digital signature can be used both to encode the data en verify its source.

3 Example Application

In this section we present an example application of the proposed services for distributed scientific computing. We envision the following scenario in which multiple applications written in different programming languages share data on the Web. The objective of this example scenario is to predict the impact of pollution of a river on the environment. A simulation is performed by using accurate dynamic data generated by different applications on the Web.

To solve this problem, the scenario is as follows: first the geometric data of the river location has to be obtained from a geographical information system (GIS). Then a simulator with the geometric information and the sources of the pollution creates simulation results. Finally, the simulation results have to be visualized and results are updated when new information about the pollution is available.

In this example, the sequence of commands described by the scenario is translated into a functional signature. An itinerative mobile agent takes the functional signature and communicates with a functional proxy server on the machine on which the agent currently resides. The functional proxy servers match the specifications of the commands to servers on the network. With this information, the agent can move to the appropriate service resources to execute the command.

The movements of the mobile agent and data exchanged by the applications involved in the example is shown in Figure 3. The component-based PSE on machine **A** helped a user to define a scenario to solve the pollution modeling problem. The scenario is translated by the PSE on the machine into a functional signature. The PSE translates this scenario into the following functional signature:

$$\mathit{visualize}(\mathtt{A}, \mathit{simulate}(\mathtt{transport}, \mathit{geometry}(\mathtt{river}, \mathit{area})))$$

where *area* describes the affected river location represented in XML.

From machine **A** the PSE sends out an agent with the functional scenario. No XML objects are communicated at this point. The agent's responsibility is to seek out the services and establish the connections between the services.

The agent engages the functional proxy server on machine **A** to find a server that can supply the geometry for the specified area adapted to river streams. The functional proxy forwards the agent to machine **B**. The agents docks with the XML API of the GIS application and retrieves the river geometry and current pollution levels. The next function is the simulation by a transport model for the pollution. The functional proxy server of machine **B** forwards the agent to machine **C**. Meanwhile, the data proxy servers maintain the communication channel with the GIS, that transports XML (and possibly functional signatures). The APIs translate XML into internal data structures. In this way, updated data from the GIS can be retrieved by the simulator. The data proxy servers take care of network problems such as nodes that go down and possible network congestion by keeping current tables with

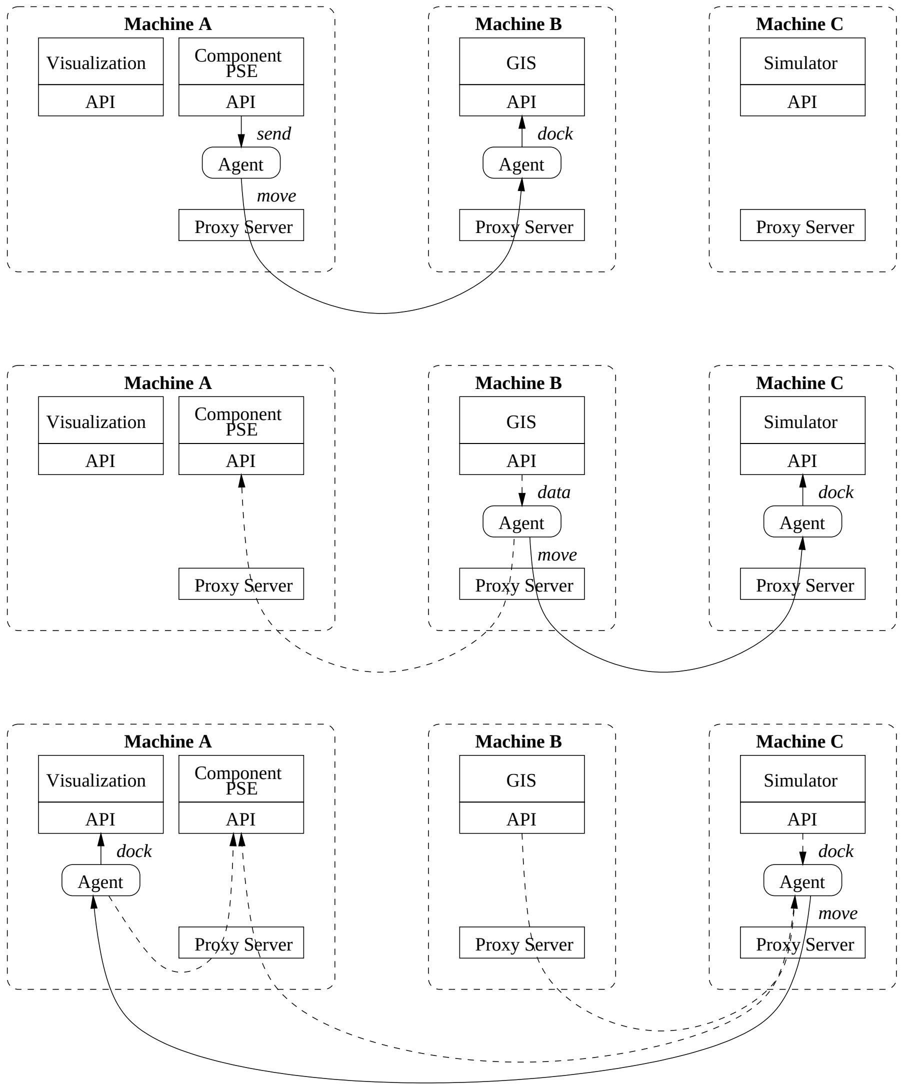


Figure 3: An Itinerant Mobile Agent Solving a Pollution Modeling Problem

network profiles. Proxy servers define a virtual network by recording machines that provide well-defined services. The actual topology of the network is unimportant to the distributed application and mobile agents do not have to "know" how to locate a service and move. Proxy servers create a virtual topology and maintain the data streams between applications and agents.

After docking to the simulator on machine **C**, the agent looks for a visualization tool on machine **A** (as specified in the functional signature). The reason for using machine **A** is that it is more efficient to visualize data using the same machine on which the software (and hardware) runs. Again, the functional proxy servers forward the agent to machine **A** and maintain the information streams between machines **A**, **B**, and **C**.

4 Summary

In this proposal we described a new paradigm for the development of distributed scientific applications by enabling resource discovery and dynamic data to be exploited in scientific computing. By exploiting dynamic data, simulations can produce more accurate results. To enable resource discovery, systems cannot be composed of fixed collections of components. Instead, dynamic configurations of components for scientific computing can be formed in which new services can be added and removed without severely affecting the environment. Existing component-based systems have a central point of control. This central control makes distributed system development easier, but denies a dynamic environment capable of changing data representations, services, and resource discovery. The paradigm combines the use of XML as a persistent data representation, functional signatures to define the semantics if applications, mobile agents and proxy servers as object request brokers, and itinerant mobile agents that solve problems by moving between host to establish connections.

5 Results From Prior Support

Robert van Engelen is a Co-PI on the NSF CISE CCR grant *"Automatic Validation of Code Improving Transformations and Related Applications"* awarded Summer 1999. Since this is a recent award, no results have been reported yet.

References

[1] MathML. the World-Wide Web Consortium. Available from `http://www.w3.org/Math`.

[2] OpenMath. the OpenMath Consortium. Available from `http://www.openmath.org`.

[3] Research council says US climate models can't keep up. *Science*, 283(5403):766, 1999.

[4] A. Acharya. Mobile streams. In *Sixth Annual Tcl/Tk Workshop*, San Diego, CA, 1998.

[5] R. Akers, E. Kant, C. Randall, S. Steinberg, and R. Young. SciNapse: A problem-solving environment for partial differential equations. *IEEE Computational Science & Engineering*, 4(3):32–42, July/September 1997.

[6] D. Cabaza, M. Hermenegildo, and S. Varma. The pillow/CIAO library for internet/WWW programming using computational logic systems. In 1st *Workshop on Logic Programming Tools for Internet Applications, JICSLP'96*, Bonn, Sept. 1996. Available from `http://www.clip.dia.fi.upm.es/miscdocs/pillow/pillow.html`.

[7] P. Clements, T. Papaioannou, and J. Edwards. Aglets: Enabling the virtual enterprise. In *International conference on Managing Enterprises - Stakeholders, Engineering, Logistics and Achievement*, 1997.

[8] G. Cybenko, R. Gray, D. Kotz, and D. Rus. D'agents: Security in a multiple-language, mobile-agent system. *Mobile Agent Security, Lecture Notes in Computer Science*, 1998.

[9] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Jnl. of Supercomputer Applications*, 11(2):115–128, 1997.

[10] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, San Francisco, 1998.

[11] D. Gannon, R. Bramley, T. Stuckey, J. Villacis, J. Balasubramanian, E. Akman, F. Breg, S. Diwan, and M. Govindaraju. Developing component architectures for distributed scientific problem solving. *IEEE Computational Science & Engineering*, 1998.

[12] S. Gray, N. Kajler, and P. S. Wang. Design and Implementation of MP, a Protocol for Efficient Exchange of Mathematical Expressions. *Jnl. of Symbolic Computing*, 1997.

[13] P. group. The PSEware project: A toolkit for building problem-solving environments, 1996. Available from `http://www.extreme.indiana.edu/pseware`.

[14] M. Higgs and B. Cottman. Solving the data interoperability problem using a universal data access broker. I-Kinetics, Inc., http://www.componentware.com.

[15] E. Houstis, J. Rice, N. Chrisochoides, H. Karathanasis, P. Papachiou, M. Samartzis, E. Vavalis, K.-Y. Wang, and S. Weerawarana. //ELLPACK: A numerical simulation programming environment for parallel MIMD machines. In 4th *ACM International Conference on Supercomputing*, pages 96–107, New York, 1990. ACM Press.

[16] O. Inc. Java voyager, 1998. Available from `http://www.objectspace.com/products/vgrOverview.htm`.

[17] W. Li and D. G. Messerschmitt. Java-to-go: Itinerative computing using java. Department of Electrical Engineering and Computer Sciences University of California at Berkeley http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go/.

[18] N. Project. Information power grid (ipg), 1998. Available from `http://www.nas.nasa.gov/IPG/`.

[19] C. Szyperski. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley, 1998.

[20] R. van Engelen, L. Wolters, and G. Cats. Ctadel: A generator of multi-platform high performance codes for pde-based scientific applications. In 10th *ACM International Conference on Supercomputing*, pages 86–93, New York, 1996. ACM Press.

[21] R. van Engelen, L. Wolters, and G. Cats. Tomorrow's weather forecast: Automatic code generation for atmospheric modeling. *IEEE Computational Science & Engineering*, 4(3):22–31, July/September 1997.

[22] L. F. Wilson, G. Cybenko, and D. Burroughs. Mobile agents for distributed simulation. Technical report, Thayer School of Engineering, Datmouth College, 1998.