An End-to-end Web Services-based Infrastructure for Biomedical Applications

Sriram Krishnan*†, Kim K. Baldridge*†‡, Jerry P. Greenberg*†, Brent Stearn† and Karan Bhatia†
*National Biomedical Computation Resource
†San Diego Supercomputer Center
UC San Diego MC 0505, 9500 Gilman Dr, La Jolla, CA 92093
‡Institute of Organic Chemistry, University of Zurich
Winterthurerstrasse 190, CH-8057 Zurich, Switzerland
sriram@sdsc.edu, kimb@oci.unizh.ch, {jpg, flujul, karan}@sdsc.edu

***Abstract*— Services-oriented architectures hold a lot of promise for grid-enabling scientific applications. In recent times, Web services have gained wide-spread acceptance in the Grid community as the standard way of exposing application functionality to end-users. Web services-based architectures provide accessibility via a multitude of clients, and the ability to enable composition of data and applications in novel ways for facilitating innovation across scientific disciplines. However, issues of diverse data formats and styles which hinder interoperability and integration must be addressed. Providing Web service wrappers for legacy applications alleviates many problems because of the exchange of strongly typed data, defined and validated using XML schemas, that can be used by workflow tools for application integration. In this paper, we describe the end-to-end architecture of such a system for biomedical applications that are part of the National Biomedical Computation Resource (NBCR). We present the technical challenges in setting up such an infrastructure, and discuss in detail the back-end resource management, application services, user-interfaces, and the security infrastructure for the same. We also evaluate our prototype infrastructure, discuss some of its shortcomings, and the future work that may be required to address them.**

I. Introduction

For a number of years, there has been a shift in grid computing towards a "services-oriented" architecture leveraging Web services technologies and methodologies. With the recent Web Services Resource Framework (WSRF) [19] specification and initial releases of a reference platform, that shift continues. Despite much enthusiasm for this shift, there are remarkably few articles or papers that show how to build a production-level services-oriented infrastructure with the technologies that are available today. Such an illustration would serve to highlight those technologies and methodologies that are in a mature state, those that are not yet production-ready, and those that are entirely missing and needed.

In this paper, we aim to present such an end-to-end architecture for a production infrastructure currently being built as part of the National Biomedical Computing Resources (NBCR). This infrastructure is scheduled to go into production use by the end of this calendar year and is already available for alpha testers. The infrastructure is focused on providing access to a variety of biomedical applications at different molecular scales. However, the initial version is limited to chemical and molecular analysis applications.

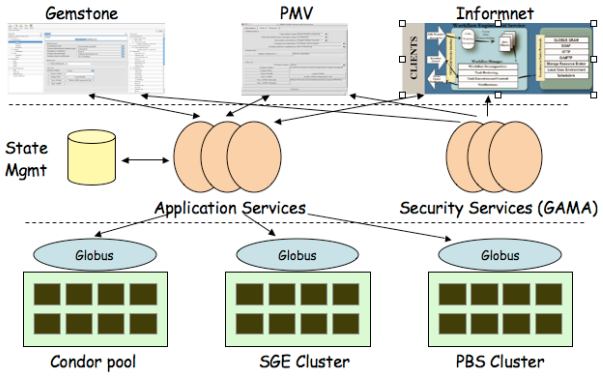

Fig. 1. The end-to-end Web services architecture

As a production infrastructure, considerable effort has been made to leverage mature software with known reliability and performance characteristics. Security is a major concern at all layers in the infrastructure as is ease-of-use for the end user – we strive to conform to the known best practices for Grid security but wrap these capabilities in an easy-to-use and easy-to-manage way. Performance of a single application is not a driving priority, but rather stability of the system and scalability in terms of concurrent users, compute resources, and number of applications have taken precedence. Integration and interoperability between the different applications is of particular interest; therefore the infrastructure should support aggregation and composition using higher-level workflow tools and semantic mediators. Finally, although we are working to provide a complete end-to-end environment for the end-user, we have strived to build an open system with published interfaces at all layers where different tools can be substituted as appropriate.

Figure 1 shows the overall multi-tiered architecture of our system, with the compute resources at the bottom tier, the application services in the middle tier, and the end-user environment in the top tier. The bottom tier is composed of multiple clusters that are located at different sites. NBCR itself has several clusters, including a 103 node cluster of Dell

PowerEdges, and a 16 node cluster of Intel Xeons. Additional clusters may be available if they are not used by other projects, including GEON's [1] 12 node HP cluster, and the UCSD Department of Bioengineering's 210 node Dell PowerEdge cluster. Section III discusses resource management on these clusters in greater detail. The Application services form the middle tier and provide domain-appropriate Web service interfaces for any client application. Section IV describes the implementation of these services. Finally, the top tier is the environment in which the end-users interact with the services. This includes common Web-based portals, workflow tools like PMV/Vision [4] and Informnet [2], and a custom environment being built called Gemstone [3] that is described in Section V. Section VI describes the security infrastructure that is used throughout our system. Section VII presents an evaluation of the performance of the system and a discussion regarding the state of the tools for building Web services. Section VIII presents our conclusions and future work.

II. Scientific Background

The fundamental goal in molecular modeling research is to understand how the interplay of structural, chemical and electrical signals manifest themselves into important biological functions. Computational experiments can often fill in important gaps or provide verification for the experimental studies. However, as molecular structure increases in size and complexity, addressing the consequences of molecular and structural specialization and variation becomes more and more difficult for either experimental difficulties and/or computational challenges. Computational scientists have at their disposal several powerful simulation techniques which provide the means to explore the functional consequences of variation in structural features. Computational modeling can be viewed in a hierarchical fashion, ranging in accuracy as well as complexity of computation.

Of the hierarchy of models available for molecular computations, first principles, (ab initio) computational methods offer the most accurate approximations. These models solve the Schrodinger Equation and molecular Hamiltonians, from which molecular structure, mechanisms, properties, and dynamics, are determined. Such enormously powerful techniques have, to date, had little direct impact on the prediction of macroscopic properties of biological materials due to the associated computational cost. Detailed quantum mechanical treatments of molecular and electronic structure, however, can predict molecular geometry, follow the reaction paths of chemical transformations, predict electrostatic effects in a variety of environments, estimate pKa shifts, and provide interpretations of spectroscopic probes of molecular environments, all with considerable accuracy.

As the molecular system size grows, the use of quantum chemical methodologies is typically prohibitively expensive. Instead, less accurate, more affordable, methodologies, such as empirical force fields or electrostatic methodologies are often employed. There are many different applications of this type and while related, each represent a different algorithmic methodology for understanding molecule structure and properties, and as such provide different levels of accuracy. In many areas of research, such applications may be accessed by scientists in one of many combinations. More recently, with advancements in computational technology, innovative couplings between a wide range of different but related applications, hybrid modeling, is becoming an important aspect of computational modeling.

In the present work, it is our goal to investigate hybrid methodologies particularly focused for study of protein-ligand interactions. We employ a technique using the highly accurate quantum chemical model for the small ligand, employing the computational chemistry package GAMESS [29], with the less accurate electrostatics model for the ligand-protein complex, using the APBS [26] computational package. The associated tool, QMView [12], is then employed in the pipeline for visualization and analysis. Estimating correct three-dimensional atomic structures of complexes between proteins and ligands is an important component of the drug-design process in the pharmaceutical industry. The computational method used for this procedure is called "docking". This process usually involves extracting separate geometries for the protein and the ligand of interest from structural databases, often in a high-throughput manner by scanning many proteins or ligands against each other. Then the relative orientation between the protein and the ligand is varied, which corresponds to the scanning of three translational and three rotational degrees of freedom between both molecules, until their optimal orientation is found. In some cases parts of the protein or the ligand are also allowed to be flexible, that is, certain bond torsions are permitted, however, this is beyond the scope of our project at the moment.

III. Cluster Back-end

Our goal is to leverage the set of computational resources available not only across the UCSD campus, but also across other partner sites. However, clusters at different sites run schedulers of their choice, e.g. some of the schedulers used in practice include Condor [13] and Sun Grid Engine (SGE). Hence, it is mandatory that these schedulers be accessible in a generic way, so that the clients are able to communicate with them in a uniform non-scheduler-specific fashion. Furthermore, users typically interact with schedulers such as Condor and SGE via command line interfaces. However, in order to expose the applications as services, we need to access these schedulers programmatically.

The Globus Toolkit [27] is an open source software toolkit used for building Grid systems and applications. It provides a "bag of services" that are useful for interacting with Grid resources in a secure way. One of the services that it provides is the Globus Resource Allocation Manager (GRAM) [18], which processes the requests for resources for remote application execution, allocates the required resources, and manages the active jobs. Furthermore, GRAM provides an API for submitting and canceling a job request, as well as for checking the status of a submitted job. The specifications are written by

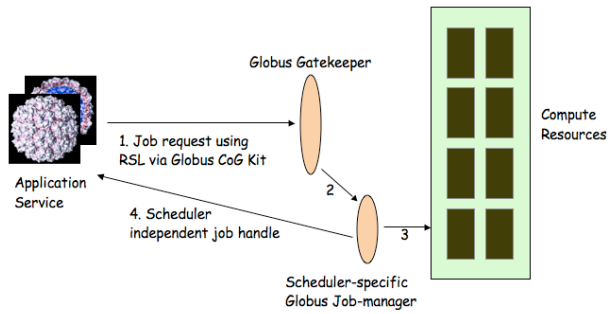

Fig. 2. Access to Schedulers using Globus GRAM

the user in the Resource Specification Language (RSL), and are processed by GRAM as part of the job request. The RSL is scheduler-agnostic; hence, the clients are oblivious to the actual schedulers being used at the back-end. This enables the use of disparate schedulers without any modification to the client-side code.

Figure 2 shows the flow of control when a job is launched using GRAM. A client submits a job request using the RSL. This RSL is received by a Globus *Gate-keeper*, which then spawns off a scheduler-specific *Job-manager* for every job. The job-manager is responsible for interpreting the RSL, converting it to the appropriate scheduler-specific parameters, and launching the job using the scheduler. Once the job is launched, a handle to the job-manager is returned to the client. Clients can use this handle to contact the job-manager and query it for status of the executing job. Note that the Globus Gate-keeper authenticates and authorizes a user before launching the Job-manager, which is not shown in the diagram.

With the advent of OGSA [23] and WSRF, GRAM is being implemented using Web service technologies. However, we still use the more stable pre-Web service versions of GRAM on the computational resources. On the client side, we use the Java CoG Kit [32] to communicate with the remote Globus services. We plan on migrating to the newer Web service versions once they have become production quality. However, we don't anticipate any major road-blocks in doing so because the Java CoG Kit would still be responsible for providing the same APIs to the new Globus services that are Web service-based.

Our current implementation can only support one type of scheduler at any point in time. *Meta-scheduling*, which provides an ability to leverage multiple schedulers at the same time, is not implemented. However, our architecture is highly conducive for providing the same, and that is indeed our eventual goal. One approach for providing meta-scheduling is with the help of the Grid Information Service (GIS), which is one of the other services that the Globus Toolkit provides. GIS provides information about the state of the Grid infrastructure. Information from the various resources (running their individual schedulers) may be published into the GIS. The meta-scheduler can then select the most lightly loaded resource from the GIS and choose to schedule the job on that resource via the GRAM API, irrespective of the scheduler being used by that particular resource.

Another approach to meta-scheduling is with the help of a Condor *glide-in*, which is a tool for dynamically adding Globus resources onto a Condor pool. Using this approach, Condor daemons are spawned on a resource using GRAM, and these resources become part of a Condor pool even if they are running their own schedulers. This approach is also realizable within our architecture.

IV. Application Services

To make the biomedical applications that are part of NBCR *grid-aware*, the following requirements have to be fulfilled - remote execution and access to Grid resources, support for multiple concurrent users, access via a set of disparate clients, and the use of standards-based security mechanisms. Furthermore, as discussed in Section II, one of our primary goals is to couple together applications across different molecular scales.

A services-oriented architecture lends itself very well to satisfying these requirements. Applications can be *wrapped* as services that provide transparent access to Grid resources to the clients. Since the services are exposed via well-defined published APIs, different user-interfaces can be built for the same set of services, as appropriate for the end-user.

Web service technologies are used as the *de facto* standard for applications on the Grid, and we use the same to build our services-oriented architecture. Standard Grid security mechanisms are used for authentication and authorization of the users, as discussed in Section VI. Furthermore, Web services-based *workflow* tools can be used for complex interactions between the individual biomedical applications, in order to enable novel scientific research.

A. Strong Data Typing and Workflows

Traditionally, scientific applications deal with inputs and outputs that are based on flat files. Typical inputs consist of a set of control options and a series of data. However, these files are not strongly typed, and can only be parsed by application-specific tools. This serves as a serious impediment if applications written by different user communities (such as ours) need to be coupled together. This is especially difficult if third-party workflow tools are to be used for application integration, since they do not understand the application inputs and outputs due the lack of strong data typing and standardized formats.

Since the advent of the Web service technologies into the Grid world, several projects have attempted to expose their applications as Web services. However, it is very often the case that the inputs and outputs of these Web services are simple strings that represent the same data as the regular input and output files of the applications. Although this may provide remote execution and access to Grid resources via a Web service interface, this is not very useful for workflow composition. This is because the string-based inputs are not strongly typed. The Taverna [22] workflow engine, that is being developed by

the MyGrid project, uses the concept of *shims* to get around this problem. Shims are application-specific translators that convert data from one format to another. Shims can be used to convert outputs of one service into inputs of another, thus enabling composition between different applications. However, this is not a generic approach because shims would need to be written between every pair of applications that need to be coupled together.

A more generic approach is to define strong data types for representing the inputs and outputs of the applications themselves. This will enable the use of third-party workflow tools for application integration. Since our architecture is Web services-based, we can use XML Schemas for defining the inputs and outputs. Standard Web service workflow specifications that mediate between services with the help of strongly typed data can then be used to enable complex interactions between these applications.

B. Applications as Web services

For our first prototype, we are providing the functionality of the APBS, GAMESS, and QMView applications as Web services. These are implemented in Java using the Apache Axis SOAP Toolkit, and are hosted inside a Jakarta Tomcat container. As an example, we will describe the implementation of the APBS Web service.

First, we abstract out the core functions that are provided by APBS. An analysis of the application shows that the three key functions provided by APBS are the calculations of solvation energy, binding energy, and electrostatic potential. Each of these is represented as an operation in the WSDL description of the service. Additionally, we analyze the inputs and outputs to APBS, resulting in the definition of strong input and output types with the help of XML Schemas. Figure 3 shows the data structure representing the input type for the calculation of electrostatic potential. It can be observed that the input contains a set of parameters that can be filled in by a user via a user-interface or retrieved programmatically by a Web service client from another service or from a repository of input data. A key part of the input data is the data type representing a *Molecule*. Several representations of a molecule exist in the life sciences, viz. the PQR file format used by APBS, the Chemical Markup Language (CML) [28], etc. We have attempted to create a common Molecule type that can be used by our applications. However, the use of strong data typing ensures that the conversions to and from disparate data formats used by other applications are relatively straightforward, e.g. XSLT [8] transformations can be used between CML and our Molecule format. Finally, the service itself is implemented to receive the strongly typed inputs, convert it into the input formats used by the APBS executable, and use the back-end resources described in Section III to execute the same.

We use the `WSDL2Java` tool provided by the Apache Axis Toolkit to generate stub and skeleton code from the WSDL service description. Hence, we don't have to deal with parsing and (de-)serializing the XML-based input and output data. The stubs and skeletons ensure that both the server and client programs only operate on Java data structures that are generated from the WSDL description of the services. Similar tools exist in other languages such as Python and C/C++.

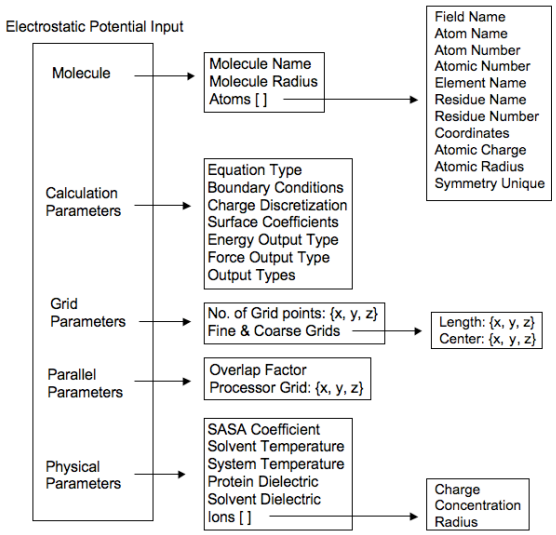

Fig. 3. Input type for calculation of electrostatic potential

C. Remote Invocations and State Management

Jobs can be executed synchronously or asynchronously. If they are executed synchronously, the client blocks until the remote execution is complete. The job outputs are returned as a response to the original request. However, this style of invocation is not always appropriate. Jobs may possibly spend a lot of time being queued if the resources happen to be heavily loaded. Furthermore, if the jobs are long running, the client will stay blocked until it finishes executing, or possibly time out.

To overcome this shortcoming, jobs can be launched asynchronously. A response is immediately sent back to the client with a `jobID` for the job being executed. The clients can use this `jobID` to query the service for job status and output metadata at a later time. However, this makes the service *stateful*. Apart from the job status and metadata about job inputs and outputs, the service state also includes user information and job history. A PostgreSQL database is used to persist this state, and is accessed via JDBC.

The Grid community has been deliberating for a few years to arrive at a consensus for representing stateful Web services.

The Open Grid Services Infrastructure (OGSI) was the first attempt by the Grid community to define a standard representation for transient stateful Web services (called Grid services). This recently evolved into the Web Services Resource Framework (WSRF) specification that better aligns the functionality of OGSI with current and emerging Web service standards. WSRF defines standard conventions for Web services for managing state so that applications discover, inspect, and interact with stateful resources in standard and interoperable ways. In the future, when WSRF implementations are more

mature and stable, we plan to use WSRF-based mechanisms for management of our service state. One possible approach is to create a new WS-Resource for every asynchronous execution, instead of the current `jobID`. Implementations of one-way messaging techniques such as WS-Notification [9] could then be used to notify clients of changes to job state.

V. Rich User Environment

The application services described in the previous section are accessible through programmatic APIs and are not meant for end-users to interact with directly. Instead the end-user will use a variety of tools and applications that in turn communicate with the application services. We don't believe that there is any one tool that will be sufficient for all the end-user needs. Some tools such as Web browsers provide ubiquitous access to the services (any machine with a Web browser and IP connectivity), but suffer from inflexibility. Other tools, such as Kepler [11], Taverna and PMV/Vision, provide a desktop environment for building domain-specific workflows. Each of these end-user environments can easily work with the application services using SOAP libraries for the various platforms and languages.

A. Gemstone Framework

In this section, we describe a new end-user environment that is currently being developed specifically for the computational chemistry community. Gemstone (Grid Enabled Molecular Science Through Online Networked Environments) provides end-users with a convenient user interface for interacting with back-end applications similar in nature to Web-based portals. Gemstone, however, is not a Web-portal and is not limited to HTML in its interaction. It is a desktop application that is fully integrated into the end-users desktop environment, for example supporting drag and drop across files and data. Gemstone is more dynamic than Web-portals as well: Gemstone accesses remote *registries* and lists the application services that are available at that time. As more services are added or services removed, this list is updated. A listed service is a application Web service such as the GAMESS and APBS services. When a user selects a particular service, a *service panel*, that provides its user interface, is loaded dynamically from the remote service. The Gemstone interface also provides a local file system browser for dragging and dropping files to and from the service panels, and GSI integration using GAMA (see the Section VI on security). Additionally, a light-weight workflow composition tool called Informnet will be integrated inside Gemstone so that the above services can be coupled together.

Figure 4 shows a screen-shot of the Gemstone application. The registry listing is shown on the left, the local file browser is on the right and the various service panels are in the middle and represented as a set of tabs. The service panel provides the user with a rich user interface for interacting with the remote service, launching jobs and monitoring running jobs. The framework provides APIs for the service panels to communicate with other service panels, to transfer files, and to access the security certificates.

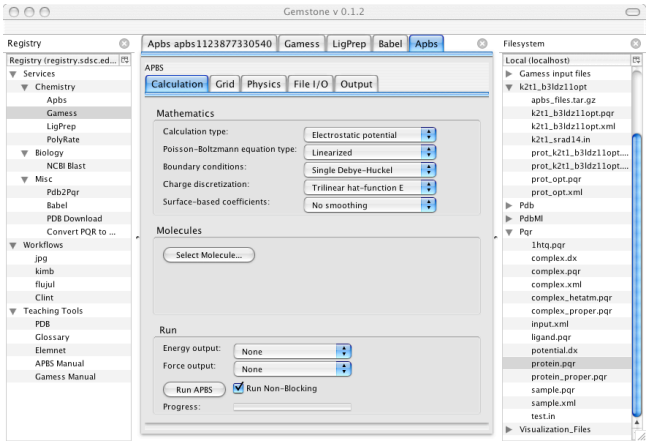

Fig. 4. The Gemstone User Interface

The Gemstone framework is built on the Gecko rendering engine developed by the Mozilla Corporation. Gecko is the same core that is used in the Firefox browser, and Thunderbird mail client. The Gecko engine provides layout and rendering of user interface elements. The user interface elements are described using an XML syntax called the XML User-interface Language (XUL) [17]. It defines all the GUI elements such as buttons, text fields, and menu items. The GUI elements are tied together using the JavaScript which is natively interpreted by the Mozilla SpiderMonkey JavaScript interpreter. A large library of components is accessible to the user in JavaScript through the XPCOM (Cross Platform Component Object Model) layer written in C. Both the JavaScript libraries and the XPCOM components are easily extensible.

Gemstone is built as an open framework that provides base capabilities such as registry lookup, user login and credential management, and local and remote file system access. This framework provides a *shell* in which different service panels (e.g. application user interfaces) are run. The service panels can interact arbitrarily with the user but is constrained in its access to local resources for security reasons.

While each application service panel operates more or less independently from the other service panels, there may be need for different service panels to communicate with other service panels and for service panels to communicate with the core Gemstone framework. This communication infrastructure is currently being developed and is using a publish-subscribe event model: service panels can register interest in certain types of events and broadcast event notifications as appropriate. When broadcast, the Gemstone framework will trigger event handlers for all service panels that have registered an interest for such events.

VI. Security

A reliable yet easy to use security infrastructure is pivotal if application scientists are to adopt a Web services approach. One of the contributions of the Globus Toolkit is the Grid Security Infrastructure (GSI) [24]. GSI is a public-key-based

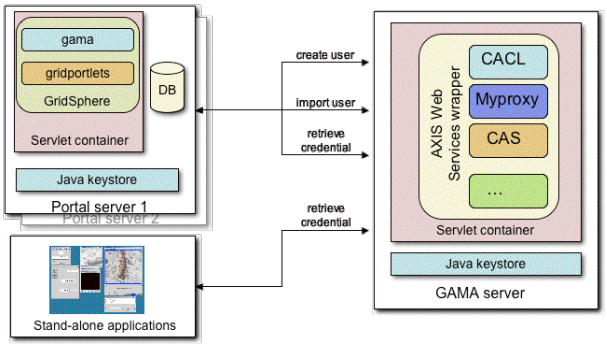

Fig. 5. Grid Account Management Architecture

system that uses X.509-based [21] user and host certificates signed by trusted third parties called Certificate Authorities (CAs). Typical usage models require that each user is assigned a user credential consisting of a public and private key. Users generate ”delegated proxy” certificates with short life spans that get passed from one component to another and form the basis of authentication, access control and logging.

However, there are several technical hurdles in using GSI-based security within a Web services infrastructure. We discuss these issues in detail in the following subsections.

A. Credential Management

Although GSI-based systems have been adopted universally in Grid systems, these are known to be notoriously difficult for administrators to deploy, and for users to use. From the perspective of an end-user, several portal-based tools are now available for simplifying certificate management. Although such tools are promising, we wish to access our services not just via Web portals, but also through other rich interfaces. Hence, simple portal-based interfaces do not suffice.

From the perspective of a security architect, there are several software packages for building CAs. It is desirable that these are abstracted out into a single set of services for easy deployment. We use the Grid Account Management Architecture (GAMA) [14] for accomplishing these goals. Although a detailed discussion of GAMA is beyond the scope of this paper, we discuss its architecture briefly.

The two-tier architecture of GAMA is described in Figure 5. The back-end (shown on the right) is a set of Web services that wraps the functionality of the various CA packages. These services provide operations to create users on Grid resources, and retrieve credentials for users at a later time. End-users interact with these services via a set of Web service clients. Initially, users use an intuitive Web portal to create accounts easily on the Grid resources. After account creation, users can retrieve their Grid credentials from the back-end services using any Web service clients with the help of a user-name and password that they chose during account creation. The use of GAMA greatly reduces not only the learning curve for the end-user, but also the set-up time for the security architect.

B. Authentication

In the context of Web service invocations, *authentication* is the process by which the two parties, the client and the service, attempt to confirm the identities of the other prior to any message exchange. Currently, there are two ways to do authentication using X.509-based certificates:

- **Transport-level:** This method relies on the creation of a secure point-to-point connection between the client and the server, using a Secure Sockets Layer (SSL) implementation. Once a connection has been created and identities of the client and service have been confirmed, the channel can then be encrypted, if need be.
- **Message-level:** This method relies on signing and/or encrypting SOAP messages between the client and server. No secure connection is required between the client and the service. Most of the Web services security specifications operate at the message level.

Since transport-level security relies on a point-to-point connection between the client and the service, it is not easy for it to work for a connection that includes multiple hops, e.g. in the presence of intermediaries. Furthermore, it doesn't provide an ability to sign or encrypt specific portions of messages. Message-level security, on the other hand, solves both of these problems. However, it suffers from severe performance problems. This is because signing and encrypting SOAP messages involves manipulating XML. In particular, XML canonicalization, which is an essential step before signing or encrypting, has been shown as a bottleneck by Shirasuna et al [30]. They recommend that message-level security should only be used only if there is a need to sign or encrypt portions of SOAP messages, or in the presence of intermediaries. Neither of the above are critical requirements for our applications. Hence, we use transport-level security for our authentication purposes.

To enable transport-level security between the Web services and their clients, we needed to modify a few of the default settings of Tomcat and Axis. The default SSL implementation, using Sun's Java Secure Sockets Extension (JSSE), does not work with GSI-based proxy certificates. Hence, we use a custom implementation of SSL provided by the Java CoG Kit on both the client and server sides. On the server side, we configure the `server.xml` of Tomcat to use CoG's implementation of the `HTTPSConnector`, which uses a GSI-based server socket implementation. Furthermore, we configure the Axis client to use CoG's implementation of a GSI-based `HTTPSSender`. Additionally, the `GSIConstants.GSI_CREDENTIALS` property is set to the user's proxy on the Axis stub for the remote service.

C. Authorization

In the context of Web services, *authorization* is the process where a decision is made whether a particular user has the permissions required to perform a particular operation. Authorization can be performed in a variety of ways - from the traditional GSI-based grid-map files that map user certificates

to local users, to the use of the Virtual Organization Membership Service (VOMS) [10], or the Community Authorization Service (CAS) [20]. At present, we use a grid-map based authorization technique. We are investigating other tools, and may incorporate it in our system in the future.

The grid-map authorization for the application services is implemented as an Axis Handler. Apache Axis uses a *Handler Chain* model where every message passes through a sequence of Handlers before the Web service operation is invoked. Developers are free to add their own handlers in order to manage their services. We added an Grid-Map Authorization Handler that retrieves the client's identity from the message context, and looks up a grid-map file to verify if a user is allowed to make a remote invocation. If there is an entry for that particular user in the grid-map file, then the invocation proceeds as usual. If not, an exception is returned to the client with a message that (s)he is not authorized to make that particular invocation. Other authorization techniques can be added in a similar fashion.

VII. Performance Evaluation

It is well-known that Web services and SOAP add a lot of overhead because of the verbose nature of the XML format. However, as we have discussed earlier, strong data typing with the help of XML Schemas is extremely beneficial for complex coupling of services with the help of workflow tools. Lack of strong data typing would make the application integration non-generic, and error-prone. The verbose nature of XML is a small price to pay for the other benefits that it provides. However, having said that, it is still desirable that the Web service invocations be as fast as they possible can, and that emerging faster alternatives be investigated. Here, we present a preliminary analysis of the overhead caused by Apache Axis, and discuss a few alternatives for the future.

As an example, we investigate the performance of the APBS Web service. APBS traditionally uses the PQR file format to represent a molecule. An interesting molecule typically contains a few thousand atoms. Generally, we do not encounter many molecules with more than ten thousand or so atoms. The PQR file representing a molecule with ten thousand atoms is about 800KB. The corresponding size of the XML representation that is generated by Axis is about 8MB. We have generally noticed that the XML representation of a molecule is about an order of magnitude larger than the corresponding representation in PQR format. This increases the latency between the client and the Web service due to the time needed to transmit the molecule, especially when the client is not connected by a high speed Internet connection.

We measured the performance of a non-blocking remote invocation on the APBS Web service over a Gigabit Ethernet connection, as well as over a slower cable modem. This is a fair indicator of the Web service overhead because this operation involves transfer of XML data, marshaling and unmarshaling of parameters, conversions to the application-specific input types, and job submission. We do not measure the performance of the actual APBS execution because this stays the same irrespective of whether it is executed via a Web service wrapper, or via the command-line.

In both cases, the Web services were running on a Intel Xeon cluster with four compute nodes and one head node, each having a 4-processor 3GHz CPU and 2GB memory, and running Java version 1.4.2_04. To test the performance over Gigabit Ethernet, the client ran on the head node of the same cluster. For the slower network connection, the client ran on a Macintosh Powerbook G4 with a 1.5GHz processor and 1GB memory, and running Java version 1.4.2_05. The available network bandwidth for uploads was approximately 60KB/s. It was observed that a non-blocking invocation from the Gigabit connection for the said molecule size took about 10s. However, the invocation from the slower cable modem connection took about 110s. The additional time required for the slower cable modem connection can be attributed to the time required to transfer the 8MB molecule represented in XML over the 60KB/s network. Both of these numbers were higher than we anticipated. However, the APBS execution lasts for tens of minutes or more, which is a couple of orders of magnitude greater than the Web service overhead on the Gigabit Ethernet connection. Hence, the performance is quite acceptable.

However, we made two observations that were worrying. First, the memory footprint of the Web server running Apache Axis was inordinately high. In fact, the server would run out of memory on using the default heap size. Increasing the heap size to 256MB circumvented this problem. However, this is not a very scalable solution as the memory required is proportional to the number of clients being served at any point in time. Furthermore, we noticed that the bottleneck, for both memory and time, was the de-serialization of the XML message into the molecule data type. Around 5s was spent in the de-serialization, which is about half the time taken by the Gigabit client.

Despite these poor performance numbers, it would be naive to dismiss XML and XML Schemas for describing data structures such as ours. Govindaraju et al [25] discuss the performance of various SOAP toolkits, and their observations are very interesting. They have shown that despite the fact that Axis is one of the most well-accepted SOAP toolkits, its performance is in fact one of the worst among the ones tested, especially for large data sizes. Other toolkits tested were XSOAP4/XSUL [5] which is written in Java, and gSOAP [31] which is C/C++. XSOAP4 uses techniques such as parsing of streaming data, as well as pull-parsing [7] to improve the performance to about an order of magnitude better than Axis, and to almost approach the performance of gSOAP, despite the fact that gSOAP is written in a compiled language (C/C++). This proves that fast processing of XML is possible if clever parsing techniques are used. In fact, the new generation of Axis being developed (called Axis2) is centered on a new representation for SOAP messages called AXIOM (AXIs Object Model). AXIOM is built on a StAX-based (Streaming API for XML) pull parsing API, that allows one to stop building the XML tree and just access the pull stream directly; thus enabling both

maximum flexibility and maximum performance.

The validation of XML instances against a schema is usually performed separately from the parsing of the more basic syntactic aspects of XML. However, schema information can be used during parsing to improve performance, in what is called schema-specific parsing [16]. Binary representation of XML data has also been proven to be useful for improving performance. Several groups are investigating the use of binary XML for scientific applications ([15], [6]).

VIII. Conclusions

In this paper, we presented an end-to-end Web services-based infrastructure for biomedical applications. This infrastructure enables remote execution of biomedical applications on Grid resources, with the help of appropriate schedulers. It allows multiple users to concurrently access these applications, via a multitude of user-interfaces. Furthermore, it provides GSI-based authentication and authorization mechanisms for secure access to the application services. Workflow tools are leveraged for coupling different application services in order to enable integration of multi-scale biomedical applications.

However, several technical challenges still remain to be addressed. We are investigating meta-scheduling techniques to schedule jobs across multiple clusters. We plan on leveraging WSRF toolkits for implementation of stateful Web services so that they can be discovered and used in an interoperable manner. We are researching alternate representations and parsing techniques for XML data in order to improve Web service performance. Furthermore, we are looking at alternate authorization techniques for providing a more role-based access control to the biomedical applications.

For more information about our work, including software releases, readers are strongly encouraged to visit our Web-site: http://nbcr.net/services.

IX. Acknowledgments

We thank the NIH for supporting NBCR through the National Center for Research Resources program grant P41RR08605, and the NSF for supporting Gemstone through the Middleware Grant SCI-0438430. We also thank Kurt Mueller and Sandeep Chandra for their work on GAMA, Steve Mock for his work on Informnet, Nadya Williams and Chris Misleh for Globus, Condor, SGE, and other infrastructure support, Robert Konecny for his inputs on APBS, Michel Sanner for his work on PMV and Vision, Wibke Sudholt, Celine Amoreira, Anne Bowen, and Yohann Potier for discussions on Gemstone and application integration, and Wilfred Li and Peter Arzberger for their leadership and vision for NBCR.

References

[1] GEON: The Geosciences Network. http://www.geongrid.org/.
[2] INFORMNET: Information Flow and Operation Resource Management on the Net. http://grid-devel.sdsc.edu/informnet/.
[3] The Gemstone Project. http://grid-devel.sdsc.edu/gemstone/.
[4] The Vision programming environment. http://www.scripps.edu/ sanner/python/vision/.
[5] WS/XSUL2: Web and XML Services Utility Library (Version 2). http://www.extreme.indiana.edu/xgws/xsul/index.html.
[6] XML Binary Characterization Group. http://www.w3.org/XML/Binary/.
[7] XML Pull Parsing. http://www.xmlpull.org/.
[8] XSL transformations. http://www.w3.org/TR/xslt/.
[9] Akamai, C.A.I., Fujitsu, Globus, HP, IBM, SAP AG, So nic, and TIBCO. Web Services Notification, June 2004. http://www-106.ibm.com/developerworks/library/specification/ws-notification/.
[10] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro. VOMS: An Authorization System for Virtual Organizations. In *1st European Across Grids Conference, Santiago de Compostela*, 2003.
[11] I. Altintas, C. Berkley, E. Jaeger, M. Jones, N. Ludaescher, and S. Mock. Kepler: An Extensible System for Design and Execution of Scientific Workflows. In *16th International Conference on Scientific and Statistical Database Management (SSDBM'04)*, 2004.
[12] K.K. Baldridge and J.P. Greenberg. QMView: A Computational 3D Visualization Tool at the Interface Between Molecules and Man. In *J. Mol. Graphics, vol. 13*, 1995.
[13] J. Basney, M. Livny, and T. Tannenbaum. High Throughput Computing with Condor. In *HPCU news, Volume 1(2)*, June 1997.
[14] K. Bhatia, S. Chandra, and K. Mueller. GAMA: Grid Account Management Architecture. Technical report, SDSC, UCSD, 2005. SDSC TR-2005-3.
[15] K. Chiu. XBS: A Streaming Binary Serializer for High Performance Computing. In *High Performance Computing Symposium 2004. Society for Computer Simulation International*, 2004.
[16] K. Chiu and W. Lu. A Compiler-Based Approach to Schema-Specific XML Parsing. In *First International Worksop on High Performance XML Processing(Satellite of WWW2004*, 2004.
[17] The Mozilla Corporation. XML User Interface Language (XUL). http://www.mozilla.org/projects/xul/.
[18] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *IPPS/SPDP 98, Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.
[19] K. Czajkowski et al. WS-Resource Framework, May 2004. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf.
[20] L. Pearlman et al. A Community Authorization Service for Group Collaboration. In *IEEE 3rd International Workshop on Policies for Distributed Systems and Network*, 2002.
[21] S. Tuecke et al. Internet X.509 Public Key Infrastructure Proxy Certificate Profile, 2003. IETF.
[22] T. Oinn et al. Taverna: A tool for the composition and enactment of bioinformatics workflows. In *Bioinformatics Journal*, volume 20(17), 2004.
[23] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *Computer 35(6)*, 2002.
[24] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *ACM Conference on Computers and Security*, 1998.
[25] M. Govindaraju, A. Slominski, K. Chiu, P. Liu, R. van Engelen, and M.J. Lewis. Toward Characterizing the Performance of SOAP Toolkits. In *5th IEEE/ACM International Workshop on Grid Computing*, 2004.
[26] M. Holst and F. Saied. Numerical solution of the nonlinear poisson-boltzmann equation: Developing more robust and efficient methods. In *J. Comput. Chem., 16*, 1995.
[27] Argonne National Lab. The Globus Toolkit. http://www.globus.org/.
[28] P. Murray-Rust and H. S. Rzepa. Chemical Markup, XML, and the World Wide Web. 4. CML Schema. In *J. Chem. Inf. Comput. Sci.*, volume 43, 2003.
[29] M.W. Schmidt, K.K. Baldridge, J.A. Boatz, S.T. Elbert, M.S. Gordon, J.J. Jensen, S. Koseki, N. Matsunaga, K.A. Nguyen, S. Su, T.L. Windus, M. Dupuis, and J.A. Montgomery. GAMESS. In *J. Comput. Chem., 14*, 1993.
[30] S. Shirasuna, A. Slominski, L. Fang, and D. Gannon. Performance Comparison of Security Mechanisms for Grid Services. In *5th IEEE/ACM International Workshop on Grid Computing*, 2004.
[31] R. van Engelen and K. Gallivan. The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks. In *2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002)*, 2002.
[32] G. von Laszewski, J. Gawor, S. Krishnan, and K. Jackson. *Grid Computing: Making the Global Infrastructure a Reality*, chapter 25, Commodity Grid Kits - Middleware for Building Grid Computing Environments. Wiley, 2003.