Web Services-based Data Integration for Life Science Computations

Sriram Krishnan[1,2], Brent Stearn[1], Karan Bhatia[1], Jerry P. Greenberg[1,2], Wilfred W. Li[1,2], Peter W. Arzberger[2], and Kim K. Baldridge[1,2,3]

[1] San Diego Supercomputer Center
[2] National Biomedical Computation Resource
UC San Diego MC 0505, 9500 Gilman Dr, La Jolla, CA 92093
[3] Institute of Organic Chemistry, University of Zurich
Winterthurerstrasse 190, CH-8057 Zurich, Switzerland
kimb@oci.unizh.ch {sriram,flujul,karan,jpg}@sdsc.edu
{wilfred,parzberg}@ucsd.edu

Abstract. The life sciences have entered a new era defined by large multi-scale efforts conducted by interdisciplinary teams, with fast evolving technologies. Ramifications of these changes include accessibility of diverse applications as well as vast amounts of data, which need to be processed and turned into information and new knowledge. Accessibility via a multitude of clients, and ability to enable composition of data and applications in novel ways for facilitating innovation across disciplines is most desirable. 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. Standardization of data types used by the Web service wrappers enables the use of commodity workflows tools for service composition and data mediation. In this paper, we present our initial efforts in coupling three popular life science applications with the help of Web service wrappers, and discuss their data interoperability and integration issues.

1 Scientific Motivation

The fundamental goal in molecular modeling research is to understand how the interplay of structural, chemical and electrical signals gives rise to functions in biological systems. Experimental advances over the past decades have provided means for obtaining vast amounts of data, from the level of molecules to more complex macromolecular structure and beyond. However, study of molecular and structural specialization and variation across molecular scales is still challenging for a variety of reasons, examples of which include the complexity of the computation involved, and the difficulty in establishing accurate mappings between different computed data and/or computed data with experimental phenomenon.

As a necessary complement to existing experimental approaches, computational scientists use several simulation techniques to explore the functional

consequences of variation in structural features, e.g., shape, size and molecular constituents. These models integrate information derived from multiple disciplines and across scale. Innovative pipelining of a wide range of different but related applications, including quantum mechanics (e.g. GAMESS), continuum and classical mechanics (e.g. APBS and AMBER), docking, and hybrid modeling, is becoming an important aspect of computational modeling.

For example, in molecular *docking*, an important component of drug design, the goal is to estimate optimal three-dimensional configurations of complexes between proteins and ligands. A major component of docking is to determine an energy cost function to select the best possible fit for protein-ligand structures. The cost function usually consists of an electrostatic energy part, a non-electrostatic energy part, and an entropy part. The electrostatic binding energy between a protein and a ligand can be relatively accurately computed by dielectric continuum solvation methods using the Adaptive Poisson-Boltzmann Solver (APBS) application. The charge distribution on the ligand is then determined using quantum chemical calculations provided by GAMESS, as it is small enough to obtain very accurately. The other two parts of the protein-ligand binding energy are difficult to determine accurately by theoretical computations. Thus empirical equations are often validated to real experimental results based on test sets where the correct protein-ligand structures are well known, e.g., from the Protein Data Bank (PDB). In this process, a user has to access multiple applications and databases for performing the docking calculations. Each application has a complex set of parameters that are varied (parameter sweep) until the needed values are determined. Finally, visualization of protein-ligand interactions can be achieved using applications such as QMView

Gaining chemical and biological knowledge by using computational tools for automated data acquisition, management, integration, and mining in a seamless, transparent manner has the potential to revolutionize research in the life sciences. The synergistic development of these methods serves as the motivation of the present work.

2 Data Representation, Web Services and Semantic Integration

Traditionally, scientific applications deal with inputs and outputs (I/O) mostly via custom flat file formats. Furthermore, different communities often use proprietary formats to describe similar data, e.g. the molecule data format used by GAMESS is different from that used by APBS. Application-specific scripts are needed to perform conversions between data formats in order to couple such applications together. Such scripts are error-prone, hard to maintain and not easily reuse-able.

The Web services approach is very promising for such application composition due to the use of standard protocols and data formats. Web services are networked endpoints that exchange data in XML format, and are accessed via standard Internet protocols (HTTP, SMTP). In the past few years, XML has

become a *de facto* format for representing data in the business and Grid communities. XML Schemas are typically used to define the structure of an XML document. There are several robust freely available software tools that can parse and validate XML documents in almost every language in use today.

Web services provide a clean separation of an interface from the implementation. Applications can define their publicly accessible interfaces by using the Web Services Definition Language (WSDL). WSDL operations handle strongly typed I/O parameters that are also defined using XML Schemas. Therefore, Web services are suitable for loosely coupled workflow composition, provided the XML schemas for I/O parameters are properly understood by the applications involved.

The data types involved in the I/O parameters for different applications may be similar *structurally* or *semantically*. Structural equivalence implies the use of the exact same data types, while semantic equivalence implies the use of disparate data types that follow similar *ontologies*. Ontology is the science of describing the kinds of entities in the world and how they are related, using, e.g., the Web Ontology Language (OWL) [11]. Data integration between applications is easier to perform if the data types are structurally similar, i.e. if they use a standard XML schema that is commonly agreed upon within a particular discipline or community. Integration between data that is only semantically similar, but structurally different, is more difficult to accomplish. Ontologies are required when different standards have been developed, and firmly rooted, in related communities. Automatic transformations from one structural data format to another using the mappings for the different structural formats, based on ontological equivalence or reasoning, is a topic of recent research [8].

Data represented as XML can be easily deposited in XML databases, and there exist utilities to perform conversions for mapping XML data into relational databases as well. This makes querying XML data reasonably straightforward. On the other hand, querying traditional flat files for information is extremely difficult because of the lack of strong data-typing and structure. One of our important goals is to enable running multiple instances of our workflows, and archival their input and output data. This data must be easily queried to extract useful information at a later time. The use of strongly typed XML data helps us accomplish that goal. Some scientific applications such as GAMESS have developed an XML format to describe its I/O. Most, however, remain XML illiterate, and are referred to as legacy applications.

3 Extending Existing Data Standards

Competing open or proprietary standards are common place today, due to lack of communication, agreement, or difficulties in creating a comprehensive ontology for a particular field of study.

In the chemical sciences, several efforts have been undertaken to standardize data types used. For example, the Chemical Markup Language (CML) [10] community has been working since 1995 to define a standard XML-based format for

chemical data that is applicable to all scientific applications. CML defines a set of types to represent standard chemical concepts using XML schemas. Some of the important types defined are atoms, atom arrays, bonds, bond arrays, molecules, etc. These types contain a set of standard fields that try to encompass all the possible attributes that may be of use within the scientific community. Furthermore, CML also defines types such as floats, float arrays, strings, string arrays, etc. that may be used to define attributes that have not be already defined by CML. Typically. structural validation of CML documents is done using XML Schemas, while semantic validation is performed using XSLT.

However, there were several hindrances in our adoption of CML. First, CML only defines the types that are commonly used in the community. There are several application-specific types that CML does not define, and rightly so. However, even for the types that are commonly used, CML does not define all the attributes that are needed by the applications in our workflow. For instance, an atom type defined inside a PQR file format (used by APBS) requires a *fieldName* that could be either `ATOM` or `HETATM`. The GAMESS atom type requires a field, *symmetryUnique*, that is either `true` or `false`. In theory, all the additional fields can be defined using the additional types defined by CML such as string, and integer. However, since these fields are optional in CML, a molecule instance that does not contain either of the above fields is still structurally valid. Other tools such as XSLT may be needed to validate such an instance, which is an additional step. Furthermore, since not all standard types and their fields defined in CML are required for the series of applications in our workflow, the exclusive use of CML is not efficient.

Furthermore, CML defines a mechanism to encode arrays as a string containing a concatenation of values delimited by white spaces. This is done to reduce the size of the XML document by eliminating redundant start and end tags in XML. However, this is not consistent with the way arrays are encoded via WSDL and XML schemas. Commodity toolkits available in several languages are oblivious to such encodings. One of our key requirements is that the services have to be accessible from a variety of languages and platforms.

Even though CML has these limitations, defining a separate schema only delays or even worsens potential data integration issues in the future. Consequently, our hybrid approach is to define a relatively simple XML schema for our data types, and provide XSLT-based transformations to and from CML in order to communicate with other applications that exclusively exchange CML-based data.

4 Wrapping Legacy Applications

Scientific applications are often considered legacy applications, because they don't keep up with the fast evolving standards in grid computing or Web services. Rewriting these applications are prohibitively expensive and wrapping them as Web services resolves many problems with interoperability and data integration.

External clients interact with the Web service wrappers, rather than the actual scientific codes.

To define the data types for each application, the I/O file format is analyzed and converted into a WSDL description for the Web services. For our prototype, this was performed manually. A far more elegant solution is to be able to provide automatic conversations between the XML messages received by the Web services into the required file formats, and vice versa. It is reasonably intuitive to translate an XML document into a flat file using technologies such as XSLT; however, automatic conversion from the output files into XML is far more difficult to accomplish automatically. We are investigating specifications such as the Data Format Description Language (DFDL) [6] that provide an XML-like view of data, which is physically stored in any arbitrary format. DFDL parsers can, in theory, convert the file-based outputs into XML messages that can be returned to the clients by the Web services.

APBS	***GAMESS***	***QMView***
calculateBindingEnergy	calculateChargesOfStructure	runQmviewPic
calculateSolvationEnergy	calculateEnergyOfStructure	runQmviewLig
calculateElectrostaticPotential	calculateHessian	runQmviewPQR
...	...	...

Fig. 1. Operations defined for the Web services

The first step in wrapping the applications is to identify the functionalities that they provide. Alternatively, the services could have a single operation *run* which receives the XML-ized version of the input file format of the application. This is probably sufficient if the only goal is to query archived input and output data for information. However, as mentioned earlier, our goal is also to be able to couple various scientific codes together. From a software engineering perspective, breaking the traditional monolithic scientific applications into more finely grained operations is very conducive to the creation of complex workflows.

Figure 1 shows some of the operations that are defined by the Web services in our workflow. These operations can be invoked either synchronously or asynchronously. The former requires the invoker to wait until the execution to complete, while the latter returns immediately with a job ID that can be used later to query status. Asynchronous operations are useful for jobs that are expected to run for long periods of time, as is the case for GAMESS and APBS.

The next step is the definition of data types used by these operations. Figure 2 describes the input data types defined for the APBS Web service. The molecule type is a key data type that is shared across the three applications. For simplicity, we have initially defined the *molecule* type to be a union of all the fields contained by the PQR and GAMESS molecule format. The Web service simply ignores

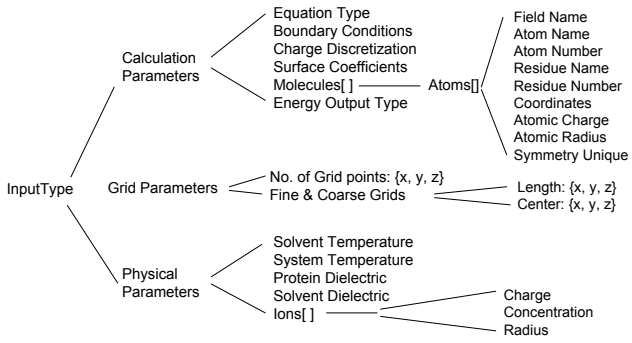

Fig. 2. The data types defined for the APBS service. All the leaf are primitive types.

fields not used by the applications during the conversion to and from the I/O files.

One known problem with XML and Web services is that it is not conducive to large amounts of data that are generated by these applications for large macromolecular complexes. Again, we are investigating DFDL-based techniques for representing large data files, while still being able to access them via an XML-like view. Furthermore, we are researching alternate transport protocols to exchange of large data sets, in lieu of the widely used HTTP.

5 Web Services based Infrastructure for Workflows

The end-to-end architecture of the Web services based infrastructure under development is briefly described and summarized in Figure 3. As shown, the scientific applications are wrapped as Web services, which are accessible from a variety of clients. The services are written in Java, using the Apache Axis Toolkit, and deployed within the Jakarta Tomcat container. Clients that wish to run any of the simulations remotely invoke the Web services. The applications are executed on available cluster and grid resources.

The services are responsible for submitting the jobs to appropriate schedulers. Different clusters and resource pools may run their own specific schedulers, e.g. Condor [7], SGE [4], etc. The Globus Toolkit [9] provides a scheduler-independent mechanism to create description of jobs using the Resource Specification Language (RSL). Each resource runs a scheduler-specific Globus *jobmanager* that is responsible for intercepting the job RSL and converting it to the

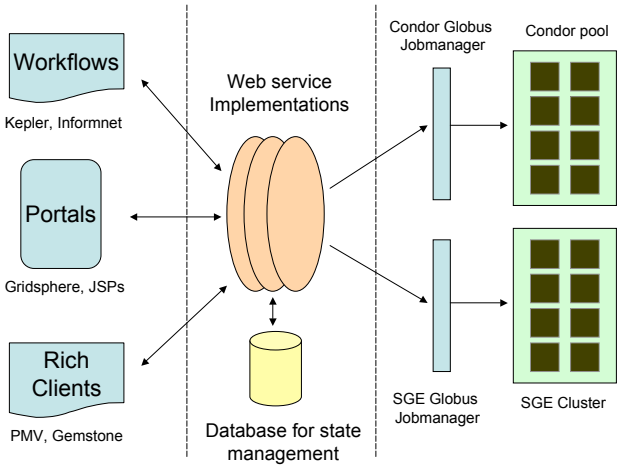

Fig. 3. The end-to-end Web services architecture

format and protocol desired by the respective schedulers. We use the Java CoG kit [12] to interact with the various Globus gatekeepers. Detailed description of the WS based grid computation and security infrastructure is beyond the scope of the paper, and will be published elsewhere.

The Web services can be accessed by a multitude of clients, including a Gridsphere-based web portal, as well as rich clients, such as the Mozilla-based interface called Gemstone [2], or the Python Molecular Viewer (PMV) [3]. In addition, for workflow management, toolkits such as Informnet [1] and Vision [5] may be utilized.

Figure 4 illustrates the data flow between the quantum and classical software, as well as the visualization step, as described in Section 1. Complicated and error-prone application-specific scripts and/or tedious hand processing involved in such a process is replaced by workflow tools that do the composition in a cleaner, more generic fashion. Not shown is the publishing of inputs and outputs into databases, to be available for querying at a later time.

6 Conclusions

This paper describes a Web-services approach for the data integration and composition of multi-scale life science computations. We have adhered to existing standards, and extended them when necessary. We have also developed a scalable infrastructure architecture that enables seamless access to Grid resources.

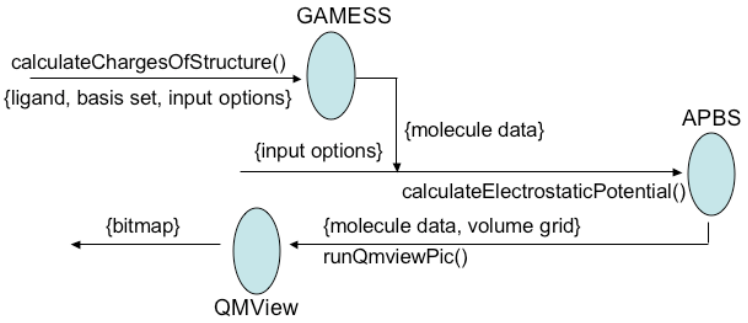

Fig. 4. Sample workflow involving the APBS, GAMESS, and QMView services

The applications are accessed via well-defined WSDL-based APIs by a multitude of clients.

Future developments include the optimization of this infrastructure for large data handling, iteration and refinement of data types, and expansion to a broader community base.

References

1. Informnet: Information Flow and Operation Resource Management on the Net, 2004. http://grid-devel.sdsc.edu/informnet.
2. The Gemstone Project, 2004. http://grid-devel.sdsc.edu/gemstone/.
3. The Python Molecular Viewer (PMV), 2004. http://www.scripps.edu/ sanner/python/pmv/.
4. The Sun Grid Engine, 2004. http://www.sun.com/software/gridware/5.3/index.xml.
5. The Vision programming environment, 2004. http://www.scripps.edu/ sanner/python/vision.
6. Data Format Description Language (DFDL), 2005. http://forge.gridforum.org/projects/dfdl-wg/.
7. J. Basney, M. Livny, and T. Tannenbaum. High Throughput Computing with Condor. In *HPCU news, Volume 1(2)*, June 1997.
8. S. Bowers and B. Ludaescher. An Ontology-Driven Framework for Data Transformation in Scientific Workflows. In *Intl. Workshop on Data Integration in the Life Sciences (DILS04)*, volume 2994. LNCS, 2004.
9. Argonne National Lab. The Globus Toolkit, 2004. http://www.globus.org.
10. 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, pages 757–772, 2003.
11. M. K. Smith, C. Weltey, and D. MacGuinness. OWL Web Ontology Language Guide, 2004. http://www.w3.org/TR/owl-guide/.
12. 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.