SOAs for Scientific Applications: Experiences and Challenges

Sriram Krishnan, Karan Bhatia
San Diego Supercomputer Center, UC San Diego
9500 Gilman Drive, La Jolla, CA 92093-0505
{sriram, karan}@sdsc.edu

Abstract

Over the past several years, with the advent of the Open Grid Services Architecture (OGSA) [11] and the Web Services Resource Framework (WSRF) [15], Service-Oriented Architectures (SOA) and Web service technologies have been embraced in the field of scientific and Grid computing. These new principles promise to help make scientific infrastructures simpler to use, more cost effective to implement, and easier to maintain. However, understanding how to leverage these developments to actually design and build a system remains more of an art than a science. In this paper, we present some positions learned through experience that provide guidance in leveraging SOA technologies to build scientific infrastructures. In addition, we present the technical challenges that need to be addressed in building an SOA, and as a case study, we present the SOA that we have designed for the National Biomedical Computation Resource (NBCR) [5] community. We discuss how we have addressed these technical challenges, and present the overall architecture, the individual software toolkits developed, the client interfaces, and the usage scenarios. We hope that our experiences prove to be useful in building similar infrastructures for other scientific applications.

1. Introduction

Over the last decade, there has been significant advancements made to the core infrastructure used to build large-scale scientific systems, termed *Cyberinfrastructures* [8]. The current focus has shifted towards the definition and implementation of a standard set of services, sometimes called Grid services [11], and towards protocols and interoperability [13]. This focus ensures that the services are not tied to any particular middleware implementation, thereby supporting multi-system, multi-language open architectures.

From a computer science perspective, the architectural concept of *service orientation* provides several interesting features:

1. Services reduce complexity through encapsulation of service implementation. A service publishes its interface and ensures that it provides the appropriate quality of service, presumably agreed to before the client accesses the service. Service implementations can leverage legacy applications, disparate resources, and even other services to implement their functionality; however, the key is that the implementation complexity is hidden from the client applications behind the service interface.

2. Service interfaces enable interoperability across systems and architectures through the use of open standards. The service interface is specified by a Web Service Definition Language (WSDL) document that includes all information needed by client applications to interact with the service. The WSDL specification itself leverages XML standards such as SOAP for message exchange and XML Schemas for complex data structures. The broad adoption of these standards has greatly facilitated interoperability across systems and vendor products unlike in the past with other distributed systems standards. This lends itself very well to the creation of *workflows*, or composition of services for the creation of complex application pipelines, where the services themselves may be hosted at disparate locations.

3. Services support a loosely-coupled model where clients can bind to service endpoints at run-time. Such late-binding provides greater fault tolerance and flexibility. Services can be discovered at run-time using metadata-based discovery mechanisms [22], and Quality-of-Service (QoS) can be negotiated dynamically.

While the benefits of an SOA approach for building Grid middleware are well known [11], it is less clear how to use SOA principles to build a Cyberinfrastructure for a particular scientific community. In particular, how can they be leveraged to help build science infrastructures that are (1) more scalable, powerful, and responsive to the needs of the end-user community, (2) more flexible and adaptable to changing requirements, and (3) more manageable and cost effective from a software development standpoint?

In our experience in building infrastructures for a num-

ber of different science domains, we have found that system requirements are hard to determine at the onset and change frequently as the user community gains experience with the new capabilities. Therefore, it is essential that the infrastructure be flexible and adaptable as the community evolves. From the perspective of system scale, the system may have a small number of initial users (tens to few hundreds). But as the capabilities of the system grow, so will the number of users, and the system architecture must scale to support more physical resources (e.g. clusters and storage) without significant re-engineering. Finally from the perspective of cost, while it is true that the cost of raw hardware (clusters, servers, and disk) is decreasing constantly, the cost of software development on these large-scale systems is increasing due to the complexity of the underlying infrastructure.

Given these constraints, building (architecting and engineering) Cyberinfrastructures is, in many ways, more of an art than a science. And answering the above questions is difficult in absolute terms – as they say in the auto industry "your mileage may vary". Nevertheless, it is an important topic for discussion and in that vein, we present our experiences as a set of positions on how to effectively use SOA principles to engineer a scientific Cyberinfrastructure.

The rest of the paper is organized as follows. Section 2 describes our positions based on our experience in a variety of Cyberinfrastructure projects, and Section 3 discusses the technical challenges. In Section 4, we present our experiences in building an SOA for biomedical applications that are part of the NBCR community. We discuss the NBCR SOA architecture, various software components, service usage, and how we have addressed some of these challenges. Finally, we present our conclusions in Section 5.

2. Positions

1. Not everyone needs an SOA: First and foremost, it is worthwhile to consider whether you really *need* an SOA for the infrastructure you are building. In cases where 1) the requirements are relatively well understood and stable, 2) the set of resources are fixed, and 3) a single interface to the applications is sufficient for the user community, it may well be sufficient to build a traditional Web portal where the application is integrated into the portal.

However, it may be still be worthwhile to use an SOA-based approach under certain circumstances. For example, if you have multiple applications that need to be shared, building a monolithic Web portal to enable access to all of them might be bad from a software engineering standpoint. It may be a better idea to build a lightweight Web portal interface for better scalability, and delegate the application-specific functionality to the back-end Web services.

Next, you have to decide whether the applications you wish to share lend themselves well to be exposed as Web services. There are certain types of applications that may be very difficult to wrap as Web services. For instance, highly interactive applications may not be appropriate as they may incur significant round-trip delays. Furthermore, traditional Web service toolkits may not be very conducive to handle a high degree of interactive traffic.

In addition, tightly coupled applications – that is, applications with multiple components that communicate frequently, as is the case for typical parallel applications running on a cluster – must not be implemented using traditional SOA techniques. SOA implementations typically use XML-based data representations, which are less efficient than binary data structures. Furthermore, SOA tools don't provide the necessary capabilities such as process creation, multicast communication, or synchronization primitives. Other approaches such as MPI are designed specifically for parallel applications, and hence should be used for implementing tightly-coupled applications. Having said that, it might very well be appropriate to wrap *entire* parallel or tightly coupled applications as a single service. In this case, the service would just provide an interface to launch and monitor such an application, but not the mechanisms for inter-component communication.

2. Focus on application-level services: Until recently, SOAs have only been used to build Grid middleware. For instance, the Globus Toolkit [10] has been redesigned to be consistent with latest Web service technologies. This is great for middleware developers who can use the platform-independent WSRF-based APIs to access standard Grid functionality, such as job submission and data transfer. However, developers of scientific tools are not experts in Grid technologies, irrespective of the simpler and more consistent APIs being provided. From their perspective, the services that are most relevant are services that perform a scientific operation and where the semantics of the operations are defined in terms of the domain science. For example, a Blast service can provide biologists with the capability to compare multiple DNA sequences against each other or against standard publicly available data sets. The particular infrastructure used for its calculation is irrelevant. The tool developers would rather focus on the algorithms and the appropriate interfaces to present to the scientific end-users. In [9], Foster introduces the term *Service-Oriented Science*. In his words, "Grid technologies can accelerate the development and adoption of service-oriented science by enabling a separation of concerns between discipline-specific content and domain-independent software and hardware infrastructure." Enabling access to scientific applications and tools using a service-oriented approach allows the developers of scientific tools to focus on the domain science, and delegate the management of the complex back-end resources to others who are more proficient in Grid middleware.

3. Provide users access to virtualized resources: Users

accessing the Cyberinfrastructure should not be provided with the raw resources – rather, the resource access should be *virtualized*. Resources include data files, computational cycles, or even application services. For instance, providing individual user accounts on computation resources does not scale very well with the number of users, or the number of computational clusters. Instead, individual application services running on the virtualized back-end computational resources may be provided to authenticated users. The services can hide the disparity between the heterogeneous resources in the back-end, and provide a uniform interface to the users.

Similarly, data files should be accessed through generic object identifiers or uniform resource identifiers, and not through direct file paths on hosts. This enables the Cyberinfrastructure to perform optimizations such as replication or data migration for better fault tolerance, and higher availability. The use of virtualized resources through a layer of indirection allows the system developers to add or move resources as needed without any interruption. For example, additional clusters can be added, data moved, while the end-user is completely oblivious to such details.

4. Do not impose a single user interface on scientific users: There are many classes of users using the Cyberinfrastructure. On one end of the spectrum, we have the beginners or casual users who may wish to have simplified interfaces with Web-based graphical user interfaces. On the other hand, power users, typically find that graphical interfaces are more of a hindrance, and prefer command-line tools that can be scripted. From a technical standpoint, it is also difficult for Web-based GUIs to represent many of the complex functionality of today's scientific applications. Desktop applications can provide better visualization and interaction than Web-based tools, but they are harder to develop and require platform-specific considerations.

For all these reasons, it is more and more difficult to imagine a single user interface satisfying all the different users for any particular Cyberinfrastructure. Hence, we advocate application services that can be used by a number of end-user tools, including Web-based portals, rich desktop applications, command-line tools and language-specific libraries.

5. Leverage the right tools: There are a variety of software choices for building SOAs. Web service technologies are the de facto standard – however, even they come in various flavors. Your choice of tools will depend on the type of applications that you would like to support. For instance, WSRF presents a way to model state-ful *transient* Web services. If your services happen to be transient, then the WSRF model would be appropriate. However, in general, it is a good idea to choose the simplest possible technologies to implement your SOA. Vanilla Web services have an advantage that there is support in most languages, such as Java, Python, Perl, JavaScript, etc. WSRF implementations are behind the curve in that respect, but are catching up fast.

In our experience, open-source community-driven projects have been the right choice for building Cyberinfrastructures. Being open-source and committed to the community has resulted in software of high quality that not only addresses the requirements of the community, but also provides the longevity required for long-term scientific projects.

3. Challenges

3.1. Web Services Tooling

In Services-oriented Science, scientific functionality must be exposed via programmatic interfaces. This implies wrapping of scientific applications as Web services, and providing science-oriented APIs for tool developers to leverage. The Web services themselves can be implemented in a number of languages, e.g. using Apache Axis in Java, or the ZSI toolkit in Python. However, in theory, they can be developed in two distinct ways.

The Web services may all be hand-written in a custom fashion. This involves defining an application-specific API for a scientific application using WSDL. For instance, a Blast service may contain an operation *searchDatabase*, which accepts an input sequence and Blast-specific options. The implementation of the Web service would receive the input options, and run the Blast job on Grid resources transparently on behalf of the user. The inputs and outputs may be *strongly-typed*, and described in detail using XML schemas. This approach is quite flexible; however, it does not scale very well as the number of applications in the system increases.

On the other hand, generic Web services wrapper toolkits may be used to expose scientific applications as Web services. This approach scales very well with the number of applications, but also has its cons. First, the implementations are usually not very flexible. Since the wrapper toolkits do not have any knowledge of the application behavior, they typically cannot do any processing that is application specific. Second, description of application I/O and incorporation into the service WSDL is quite challenging. Typically, the service WSDLs are not as expressive as they would be if they are written by hand. Depending upon the needs of the system, one of the two approaches described above may be chosen.

Once the services are defined, they must be discovered by clients. This is done with the help of *registries* such as Universal Description, Discovery and Integration (UDDI), or via semantic discovery techniques [23].

3.2. Data Interoperability

Composing multiple applications together into a scientific workflow can be extremely difficult because of the numerous data translations that may need to be performed from one format to another. There are different approaches to enable data mediation between such applications. On one end of the spectrum, everyone is forced to use the same data formats and representations. On another, everyone uses their own data formats, and there is some mechanism to mediate between them, as need arises. In general, mediation can be performed centrally, or by the individual entities. In the former, a central service performs all the translations, while in the latter the individual parties are responsible for the translations themselves [6].

While there are pros and cons to each other, the approach depends on the circumstances. From a software engineering standpoint, it is much easier to deal with common standardized formats. From a social standpoint, it is tough to enforce standards, especially if different groups have different requirements. In cases where it is possible to do so, a single ontology model works best. If this is not possible, different groups may have different ontology models, and semantic mediation approaches may be used to interoperate between them.

Representation of the standard data formats and ontologies is a challenge as well. Although the use of XML is the de facto standard way of representing data and metadata in the Web, the use of XML is not very efficient for large amounts of scientific data. Alternatives such as binary XML and the Data Format Description Language (DFDL) [2] can provide a more compact representation of scientific data.

3.3. Security

Security is important if you wish to restrict access to the Web services to a certain set of users. In this context, the keys goals are credential management, authentication, and authorization.

In the Grid world, GSI-based [12] security is the de facto standard. GSI is a public key system that uses X509-based user and host certificates signed by trusted Certificate Authorities (CAs). Every user is typically assigned a GSI *certificate*, which can be used to create limited lifetime *delegated proxy* certificates that form the basis of authentication, access control and logging. Although GSI-based systems are universally adopted in Grid systems, they are difficult to deploy and use. However, there are several tools that simplify the management and retrieval of credentials, such as GAMA [7].

Authentication is the process by which the client and the service attempt to confirm their identities of the other prior to any message exchange. This can be done at the transport-level, which relies on the creation of a secure point-to-point connection between the client and the server, and at the message-level, which relies on signing and/or encrypting the SOAP messages between the client and server. 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, addresses both of these problems. However, it suffers from severe performance problems. In general, transport-level security is better for performance, while message-level security is better for greater flexibility.

Authorization is the process where a decision is made whether a particular user has the permissions required to perform a particular operation. In the most basic form, every resource can use an access control list for authorization. However, managing authorization at a per-user level may not scale very well with the number of users. It may be necessary to define roles where a certain set of users belonging to a certain role may have more privileges than others (e.g. developers versus users, PIs versus students, etc). Information about the roles can be embedded into the X509 certificates in the form of assertions using a mechanism such as the Security Assertion Markup Language (SAML). The resource provider can extract the assertions before making an authorization decision.

3.4. Fault Tolerance and Scalability

The Web services middleware should support fault tolerance and scalability at multiple levels. First, there is the Web services container hosting the services themselves, and then there are the resources on which the jobs are being scheduled. For hosting the Web services, there are several containers available that provide robust implementations that can contribute to a high degree of fault tolerance. For instance, Jakarta Tomcat is an open-source container that is most commonly used in projects in the world of Grid computing, and provides mechanisms for clustering multiple instances for redundancy and load-balancing purposes.

For the back-end resources, Grid-based schedulers, such as the Sun Grid Engine (SGE) and Condor, provide scalability by launching jobs on a large number of distributed resources, and also provide mechanisms for fault tolerance, such as restart on failures. Furthermore, meta-schedulers such as the Community Scheduler Framework (CSF4) are capable of scheduling jobs across multiple clusters running different schedulers. Care must be taken not to reinvent the work on fault tolerance and scalability at the Web services level, and rather use the support provided by the back-end

scheduler frameworks, where possible.

3.5. Accounting and QoS

Accounting is the process by keeping track of service usage, and enforcing allocations per user. For instance, it is unacceptable for users to be stuck in a queue behind a user who happens to be running thousands of jobs. It should be possible to limit the number of jobs for a particular user to enforce fair share scheduling of jobs on the back-end resources.

Qualities of services and Service Level Agreements (SLA) are related to user accounting and allocations. It should be possible for users and service providers to agree on service quality metrics, and associated penalties if these metrics are not satisfied. For example, one particular metric could be that the average turnaround time for Blast jobs of a particular size be less than 2 minutes. The SOA should be able to specify such metrics, and implement this capability. Different users may have different priorities and different qualities of service - it is a challenge to build a system that maximizes the adherence to the service level agreements, and minimizes the penalties if they are indeed broken.

4. Case Study: The NBCR SOA

4.1. Goals

Our goals for the NBCR user community are twofold. First, we wish to provide transparent access to the emerging Grid-based computational infrastructure by "grid-enabling" biomedical codes and providing access to distributed biological and biomedical databases. This will allow biomedical researchers to harness the computational power and securely access very large resources and specialized instruments available in the emerging and distributed Grid environment.

Secondly, our goal is to enable integration of applications across different scales (e.g. atomic to macromolecular, molecular to cellular, tissue to organ). For instance, a user may wish to utilize the highly accurate quantum models to calculate atomic charges, but opt for the less accurate but significantly more computationally tractable classical molecular dynamics approach for determining protein-ligand docking [17]. Each of these capabilities has been developed by independent developer communities over long periods of time sometimes decades and in some cases by different but related domain communities. Currently, integrating these capabilities requires users to learn the intricacies of each software implementations and perform time-consuming data format conversions and/or other code restructuring. Our goal is to provide a mechanism for scientific users to discover and leverage these capabilities, in a standard easy-to-user manner, possibly with the use of commodity freely available software tools.

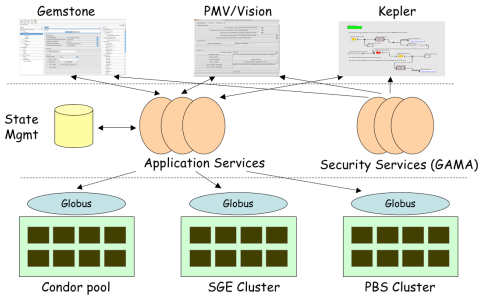

Figure 1. NBCR SOA Architecture

4.2. Architecture Overview

Web service technologies have become the de facto standard for accessing applications on the Grid, and we use the same to build our SOA. Web services are language and platform independent; this enables access via a multitude of user interfaces.

Figure 1 shows our multi-tiered architecture for enabling easy access to scientific applications on Grid resources. The bottom tier consists of the Grid resources where the jobs are scheduled. Accessing these resources via Grid schedulers can be quite complicated for the end-user. Hence, in the middle tier, we enable programmatic access to scientific applications and security services running on Grid resources via simple Web service wrappers. Various client tools, available in the top tier, leverage the easy-to-use Web services APIs to securely access these applications running on the complex back-end infrastructure. Our Web services architecture is described in greater detail in [20].

The current set of deployed services consists of key scientific software, plus numerous secondary resources and utilities, and includes a combination of hand-written and auto-generated services based on the Opal toolkit:

1. *GAMESS*, a general atomic molecular electronic structure package which uses general ab initio quantum chemistry techniques,
2. *APBS*, a classical modeling package for electrostatics computations, important in several areas of biomolecular simulation,
3. *AutoDock* and *AutoGrid*, a suite of automated molecular docking software tools used in applications such as X-ray crystallography and structure-based drug design,
4. *SIESTA*, a macromolecular plane wave density functional software which is used for performing electronic structure calculations and ab initio molecular dynamics simulations of molecules and solids,

5. *MolPrep*, a tool enabling manipulation of molecular structures such as rotational conformations,

6. *PDB2PQR*, a utility for converting protein files from the Protein Data Bank (PDB) format to PQR format used by applications such as APBS,

7. *Babel*, a toolkit that enables users to search, convert, analyze, or store data from molecular modeling, chemistry, solid-state materials, biochemistry, or related areas,

8. *MEME* and *MAST*, software tools for discovery and search of motifs (highly conserved regions) in groups of related DNA or protein sequences,

9. *Blast*, the Basic Local Alignment Search Tool which enables users to compare a query biological sequence with a database of sequences, and identify any resemblance,

10. *HMMER*, an implementation for profiling hidden Markov Models (HMM) for biological sequence analysis.

4.3. Software Tools

Apart from the application-specific services that we have built for the NBCR SOA, we have also developed tools to help build generic SOAs for a scientific communities. In particular, the Opal toolkit [21] was developed to wrap legacy applications with Web services and integrate the back-end cluster and security services. It has been downloaded and used by a variety of external projects, and has been extended in a number of ways, e.g. to provide WSRF compatibility [16]. Similarly, the GAMA security infrastructure [7] (co-developed by UCSD's Telescience and GEON Grid projects) directly supports services infrastructures. GAMA is being used in production for a variety of UCSD-based and external Cyberinfrastructure projects.

1. The Opal Toolkit: The Opal toolkit enables application providers to automatically wrap legacy scientific applications as Web services. Application providers configure the Web service wrapper using configuration files, and the Opal toolkit automatically deploys the application as a Web service fully integrated with the local cluster scheduler and with GSI-based authentication (if desired). This enables the rapid deployment of legacy applications as Web services on the Grid without having to write a single line of source code. The 1.0 version of the Opal toolkit was released in Sept 2006, and has been used effectively by a number of projects in the Grid community apart from NBCR, e.g. the CAMERA project [1] sponsored by the Moore Foundation which addresses marine microbial genomics.

2. Data Schemas and Strongly-typed Services: In Section 3.1, we described the benefits of hand-written Web services, viz., stronger data typing and greater flexibility. In particular, strong data typing is beneficial to the creation of complex scientific workflows. If the data is strongly typed, it is easy to extract information from the output of a particular application, in order to couple it with another application. With this in mind, we have developed a standard molecule definition using XML schemas, which is used across applications in computational chemistry and continuum electrostatics. When such codes are connected together into a workflow, e.g. in order to perform protein-ligand interaction studies ([17]), it becomes straightforward to share data across them. If these applications used their own legacy file formats, doing so would be difficult without writing messy error-prone data translators that perform conversions from one file format to another. We are working actively with developers in several scientific communities in order to standardize common data types that may be shared between applications. The standardized data types will be in XML format so that it can be easily imported by the Web service wrappers, and used by standard workflow tools.

3. GAMA Security: The Grid Account Management Architecture (GAMA) provides the security infrastructure for the SOA. In general, Grid systems use a GSI-based security mechanism to administer resources. However, they are known to be difficult for administrators to deploy, and end-users to use. GAMA provides a simple mechanism for administrators to deploy the security services in the form of Rocks Rolls [19], and a Web services API for user interfaces (on various platforms) to interact with. Users have the option of retrieving the Grid credentials from the GAMA Web services for use in a stand-alone mode. However, in most cases, lightweight GAMA clients are incorporated into various clients shown in the top tier of Figure 1.

4. Client Interfaces: In Section 2, we describe how access to scientific applications should be enabled via a multitude of interfaces. With that in mind, our architecture does not impose a single user interface on the end-users. Although a detailed description of the UIs being used is beyond the scope of this paper, some of them are as follows:

1. *Gemstone*, a rich client application [17] built on top of the Mozilla Firefox platform. Gemstone runs on the user's desktop, and provides an interface for discovering and accessing remote application Web services.

2. *PMV*, the Python Molecular Viewer, and the accompanying visual programming tool *Vision* [4], being developed at The Scripps Research Institute (TSRI). These tools provide many pluggable commands ranging from displaying molecular surface to running molecular dynamics simulation and energy minimization calculations.

3. *AutoDockTools* [4], also being developed at TSRI, which is a graphical front-end for setting up and running *AutoGrid* and *AutoDock*.

4. *Kepler* [14], which is a scientific workflow toolkit being developed at the University of California, San Diego (UCSD). Kepler provides a visual interface to create a scientific pipeline, and provides a standard set of tools to interact seamlessly with emerging Grid-based resources.

5. *GridSphere* [3], which provides an open-source port-

Figure 2. SOA Usage Map

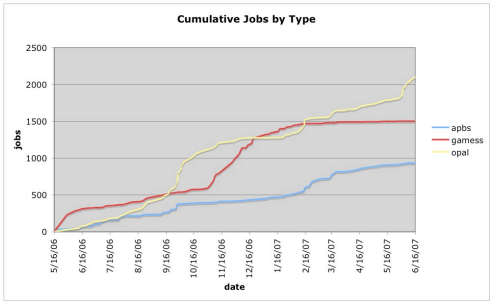

Figure 3. Cumulative jobs run on the SOA

let based Web portal framework, also being developed at UCSD. The portlet model gives users a flexible easy-to-use interface, and it gives portal developers a model to create pluggable and dynamic application support.

4.4. System Usage

The SOA infrastructure is hosted at various locations, such as clusters maintained by the Advanced Cyberinfrastructure Laboratory at SDSC, the NBCR at the University of California at San Diego, and the University of Zurich.

In this section, we use the statistics from the SDSC cluster to illustrate the use of our system. The head node is a 3.2GHz Intel Xeon with 3GB memory. There are 7 compute nodes, which are 4-CPU Intel Xeons varying from 2.8GHz to 3.2GHz, and from 3GB to 6GB of memory. The Web services are hosted within a Jakarta Tomcat container version 5.0.30, in both secure and non-secure fashion, as appropriate. Apache Axis 1.2.1 is used as the SOAP implementation for the Web services.

Figure 2 shows the geographical location of the users who have accessed our Web services. Data from May 2006 to June 2007 was collected from Tomcat logs which record the IP address and request path for each access. These statistics show that over the past year, our services have been accessed by users having nearly 300 unique IP address, spread over 35 countries and 20 US states. Most of our users are clustered around Europe, and the United States.

Figure 3 shows the job executions per application over the same time period. In total roughly 4500 jobs were run, with around 1500 GAMESS jobs, 1000 APBS jobs and slightly over 2000 other jobs (deployed using the Opal toolkit). On average, this works out to around 75 jobs per week, or 15 jobs per day (excluding week-ends).

4.5. Experiences

In Section 4.1, we mentioned that our goals were 1) to provide seamless access to Grid resources through a multitude of user interfaces, and 2) to enable novel scientific pipelines across different scales. We have successfully demonstrated success in addressing both of these goals with the help of our SOA. With the help of several intuitive user interfaces described above, scientific end-users have been accessing several scientific applications without having to deal with the complexity of the back-end Grid middleware. Furthermore, with the help of workflow toolkits like Kepler, and rich problem solving environments such as Gemstone, users have been able to integrate multiple applications across different scales.

As appropriate, we have used an automatic Web services wrapper toolkit (Opal), and hand-written services to help bring scientific applications into our SOA framework. We have addressed a number of the technical challenges listed in Section 3. In particular, we have tackled issues of data standardization and interoperability, and Web services tooling for scientific applications, security, and scalability. However, several issues still remain.

In the area of Web services tooling, we are working towards improving on the Opal toolkit, and making it more generic and flexible. For instance, at this time, the Opal toolkit uses a generic WSDL API for representing application inputs and outputs. While this is useful, it is impossible to figure out the input and output types for the scientific applications by just looking at the service WSDL. It will be beneficial for the toolkit to be able to import application input and output schemas into the WSDL, when available. This will provide easier inspection of the WSDL, and incorporation into Web services workflow tools. Furthermore, we are also planning on using alternate protocols for application inputs and outputs for better performance, as need be. We are investigating issues of automatic user interface generation from the application schemas, and support for different styles of scientific applications (currently limited to serial and parallel). We are planning on adding these features to Opal 2, which is currently in design phase. We are also working with developers in different communities to define standard schemas and ontologies, to enable interoperability and integration between different applications.

While the current GAMA toolkit provides a simple Web

services API for credential management, we are working on improving it at different levels. From a software engineering standpoint, we are working on making GAMA more modular, so that different back-end credential management systems can be plugged in seamlessly. For instance, different projects might choose to install different Certificate Authority (CA) software, or follow a unique combination of steps for account creation. Furthermore, we are investigating authorization frameworks such as CAS [18], which provide mechanisms for role-based and coarse grained authorization via assertions embedded within user credentials.

In our current system, we do not deal with accounting and quality of service constraints, which must be addressed as our resource utilization grows. Although our framework is fault tolerant by design, we have not invested a significant amount of time in investigating the limits at which it breaks down. We plan on addressing this in our future efforts.

5. Conclusions

In this paper, we presented Service-oriented Architectures (SOA) from the perspective of scientific applications. We presented some of our positions about SOAs and Cyberinfrastructures, viz. (1) not everyone needs an SOA, (2) focus on application-level services, (3) provide access to virtualized resources (4) via a multitude of user interfaces, and (5) leverage the right software tools for the job. We also discussed the technical challenges to be addressed while building such an SOA, viz. Web services tooling, data interoperability, security, fault tolerance, scalability, accounting, and quality of service. As a case study, we described the NBCR SOA for biomedical applications, presented the individual components, and illustrated how we have addressed these technical challenges. These positions and challenges are relevant to other scientific Cyberinfrastructures as well, as demonstrated by similar projects such as CAMERA.

The work on the NBCR SOA has been funded by the NIH through the National Center for Research Resources program grant P41RR08605. We wish to thank Wilfred Li and Peter Arzberger for their leadership and vision for NBCR, and the Gemstone, PMV, CAMERA and Kepler teams for their work on applications, user interfaces and workflows leveraging our SOA.

References

[1] Community Cyberinfrastructure for Advanced Marine Microbial Ecology Research and Analysis (CAMERA). http://camera.calit2.net.

[2] Data Format Description Language (DFDL). http://forge.gridforum.org/projects/dfdl-wg/.

[3] Gridsphere Portal Framework. http://www.gridsphere.org.

[4] Python-based software development at MGL - AutoDockTools, PMV, and Vision. http://mgltools.scripps.edu/.

[5] The National Biomedical Computation Resource (NBCR). http://nbcr.net.

[6] M. K. Bergman. Models of Semantic Interoperability. http://mkbergman.com/?p=240, 2006.

[7] K. Bhatia, S. Chandra, and K. Mueller. GAMA: Grid Account Management Architecture. *1st IEEE International Conference on e-Science and Grid Computing*, 2005.

[8] D.E. Atkins et al. Revolutionizing Science and Engineering Through Cyberinfrastructure: Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure. Technical report, National Science Foundation, 2003.

[9] I. Foster. Service-Oriented Science, 2005. American Association for the Advancement of Science.

[10] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit, 1997.

[11] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *Computer 35(6)*, 2002.

[12] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *ACM Conference on Computers and Security*, 1998.

[13] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In *International Journal of Supercomputer Applications*, volume 15(3), 2001.

[14] I. Altintas et al. Kepler: An Extensible System for Design and Execution of Scientific Workflows. In *16th International Conference on Scientific and Statistical Database Management (SSDBM'04)*, 2004.

[15] K. Czajkowski et al. WS-Resource Framework. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf, 2004.

[16] K. Ichikawa et al. Opal OP: An Extensible Grid-Enabling Wrapping Approach for Legacy Applications. In *3rd Workshop on Grid Computing and Applications (GCA07)*, 2007.

[17] K. K. Baldridge et al. GEMSTONE: Grid-Enabled Molecular Sciences through Online Networked Environments. In *Life Sciences Grid Workshop (Satellite of Grid Asia)*, 2005.

[18] L. Pearlman et al. A Community Authorization Service for Group Collaboration. In *IEEE 3rd International Workshop on Policies for Distributed Systems and Network*, 2002.

[19] P. Papadopoulos, M. Katz, and G. Bruno. NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters. In *Concurrency and Computation: Practice and Experience Special Issue*, 2001.

[20] S. Krishnan et al. An End-to-end Web Services-based Infrastructure for Biomedical Applications. In *6th IEEE/ACM International Workshop on Grid Computing*, 2005.

[21] S. Krishnan et al. Opal: Simple Web Services Wrappers for Scientific Applications. In *IEEE International Conference on Web Services*, 2006.

[22] N. Shadbolt, W. Hall, and T. Berners-Lee. The semantic Web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[23] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, 1(1):27–46, 2003.