O'SOAP - A Web Services Framework for DDDAS Applications*

Keshav Pingali[1] and Paul Stodghill[1]

{pingali,stodghil}@cs.cornell.edu
Department of Computer Science
Cornell University, Ithaca, NY 14853, USA

Abstract. Because of the continued development of web services protocols and the apparent convergence of Grid services with web services, it is becoming evident that web services will be an important enabling technology for future computational science applications. This is especially true for Dynamic Data-Driven Application Systems (DDDAS's). In this paper, we argue that the current systems for web services development are ill-suited for DDDAS applications. We describe O'SOAP, a new framework for web service applications that addresses the specific needs of computation science. We discuss a multi-physics simulation developed using O'SOAP and show that O'SOAP is able to deliver excellent performance for a range of problem sizes.

1 Introduction

There are certain classes of Dynamic Data-Driven Application Systems (DDDAS) that are, by their very nature, distributed systems. Some obvious examples include applications that incorporate geographically distributed instruments, such as VLA radio telescopes or sensor nets. Other, less obvious, examples include large-scale loosely-coupled applications[1] developed by multi-institutional teams. Elsewhere [2,19], we have described the Adaptive Software Project (ASP) [11], a multi-institutional project that is developing multi-disciplinary simulation systems. What is unique about our systems is that they have been deployed as a geographically distributed set of application components.

There are many advantages to this over the conventional approach of building monolithic applications. First, component developers only have to deploy and maintain their codes on a single local platform. This saves the developer time and enables codes with intellectual property constraints to be used by other project members. Second, the loosely-coupled nature of distributed components facilitates their reuse in the development of new simulations. It also enables their simultaneous use in any number of research projects.

* This research is partially supported by NSF grants EIA-9726388, EIA-9972853, and ACIR-0085969.

[1] We consider an application to be loosely-coupled if its components communication infrequently, as opposed to tightly-coupled, in which communication is frequent, or embarrassingly parallel, in which communication is absent.

What has made this approach possible is the standardization of protocols for the interoperability of distributed components. In particular, our infrastructure is based on the standard web service protocols (i.e., XML [21], SOAP [10], and WSDL[4]). While original developed for business applications, we have found that web services are ideally suited for building computations science applications as well[2].

Because these standardization efforts have led to the development of many interoperable systems for the deployment and use of web services, we believe that many future computational science and DDDAS applications will use web services to varying degrees. Such web services could be particularly useful for DDDAS applications, where runtime dynamic invocation of components is required based on additional injected data, proximity of data repositories, or when the simulation is used to control measurement processes.

As a result, one of the key technologies that will lead to the wide spread deployment of future DDDAS applications will be web services frameworks that make it relatively easy to build distributed component-based applications. Hence, it is critical that computational scientists be able to prototype DDDAS applications without an enormous development effort.

In Section 2, we will discuss existing web services systems and why they do not directly meet the needs of DDDAS application programmers. In Section 3, we will discuss O'SOAP, a new web services framework that is designed to address these needs. In Section 4, we discuss some performance result using O'SOAP. In Section 5, we discuss our conclusions and future directions of O'SOAP development.

2 Motivation

Many systems have been developed for deploying web services. These range from large systems, such Apache Axis [8], Microsoft .NET [5] and Globus [7] to more modest frameworks like, SOAP::Lite [12], SOAPpy [14] and GSOAP [6]. Unfortunately, these systems present a relatively high entry point for DDDAS application developers.

Let's consider what is required to deploy an existing application using these systems. First, the developer must write code to interface the application with a web services framework. While this code is often short, it presents a learning curve that can discourage computational scientists from experimenting with DDDAS systems.

The second difficulty is that there are many issues that arise in deploying an existing application in a web services environment that do not arise in the traditional interactive environment. As a result, the DDDAS application developer must consider:

- Generating WSDL - WSDL is the means for documenting a web service's interface. Some web service frameworks provide tools for generating WSDL documents automatically, but many require that the developer write these documents by hand.
- Data management - Data sets in computational science applications vary greatly in size. While small data sets can be included in the SOAP envelopes, other mechanisms must be used for larger data sets. Also, the developer must manage intermediate and result files that are generated by the application.

[2] Since the emerging standards for Grid computing [20,9] are based upon the basic web services protocols, we consider Grid services to be part of the more general class of web services.

– Asynchronous interactions - SOAP over HTTP is essentially a synchronous protocol. That is, the client sends a SOAP request to the server and then waits to receive a SOAP response. However, many computational science applications can take a very long time to execute, which can result in the remote client timing out before receiving the results. This must be considered when writing the code for interfacing with the application.
– Authentication, Authorization and Accounting (AAA) - The developer will certainly wish to restrict which remote users are able to use the web service.
– Job scheduling - Very often, the machine that is hosting the web service is not the same as that on which the application will run. Very often, the web service will have to interact with a job scheduling system in order to run the application.
– Performance - High performance is an important consideration for many computational science applications. It is likely more so for DDDAS applications.

The existing tools offer a blank slate for the programmer. This enables the experienced and knowledgeable web services developer to write efficient and robust solutions for each application. For the novice web service developer, this presents a tremendous hurdle that will only be tackled if absolutely necessary.

To summarize, the very general nature of existing web and Grid service tools makes deploying web services a very costly undertaking for a novice DDDAS application developer. This cost makes it unlikely that computational scientists will try to build DDDAS systems unless there is an absolute need to do so. What is needed is a new web services framework that is designed to address the needs of the DDDAS application developer. Ideally, this framework would enable a computational scientist to deploy new and existing applications as web services with little or no interfacing code. This framework must also address the considerations listed above.

3 Overview of O'SOAP

O'SOAP [17] is a web services framework that is designed to enable a non-expert to quickly deploy and use legacy applications as fully functional web services without sacrificing performance. The primary benefits of O'SOAP over other frameworks is the manner in which it builds upon the basic SOAP protocol to enable efficient interactions between distributed scientific components.

3.1 Deploying Applications as Distributed Components

On the server side, O'SOAP enables existing, command-line oriented applications to be made into web services without any modification. The user only needs to write a small CGI script that calls O'SOAP server-side applications. Placed in the appropriate directory on a web server, this script will execute when the client accesses its URL. An example of such a script is shown in Figure 1.

The `oids_server` program, which is provided by the O'SOAP framework, processes the client's SOAP request. The `-n`, `-N`, and `-U` parameters specify the short name, full name, and namespace, respectively, of the web service. What appears after `--` is a template of the command line that is to be used to run the legacy program,

```
#! /bin/bash

oids_server \
    -n arithmetic-test -U urn:test -N 'Arithmetic Server' \
    -- ./add.sh '[in val x:int]' '[in val y:int]' \
          '>' '[out file result:int]'
```

Fig. 1. Sample O'SOAP Server

`add.sh`. The text that appears within `[...]` describes the arguments to the legacy program. Each argument specification includes at least four properties,

- The directionality of the parameter, i.e., "in", "out", or "in_out".
- Whether the parameter value should appear directly on the command line ("val") or whether the parameter value should be placed in a file whose name appears on the command line ("file").
- The name of the parameter, e.g., "x", "y" and "result".
- The type of the parameter value, i.e., "int", "float", "string", "raw" (arbitrary binary file), "xml" (a structured XML file).

A component implemented using O'SOAP will expose a number of methods, discussed below, that can be invoked using the SOAP protocol. O'SOAP also automatically generates a WSDL document that describes these methods, their arguments, and additional binding information.

On the client-side, O'SOAP provides two tools for accessing remote web services. The `osoap_tool` program provides a command-line interface to remote web services. In addition, the `wsdl2ml` program generates stub code for invoking web services from O'Caml [15] programs.

To summarize, O'SOAP is a framework that hides most of the details of the SOAP protocol from the client and server programs. With this in place, we can now discuss how the interactions between the clients and servers can be organized to support distributed computational science applications.

3.2 Asynchronous interactions

As previously mentioned, SOAP over HTTP was designed for synchronous communication. To accommodate long running computational science applications, O'SOAP's server-side programs provide basic job management by exposing a number of methods to the client. The "spawn" method invokes the application on the server and returns a job id to the client. The client can then pass this job id as the argument to the "running" method to discover whether or not the application has finished execution. Once completed, the client uses the "results" method to retrieve the results. There are additional methods, such as "kill", for remotely managing the application process.

Since the server is able to generate a response for these methods almost immediately, the synchronous SOAP protocol can be used for such method invocations. Also, since a new network connection is established for each method invocation, detached

execution and fault recovery are possible without additional system support (e.g., to re-establish network connections).

3.3 Support for small and large data sizes

In computational science applications, data set sizes can vary greatly. Small data sets can be included within the SOAP envelope that is passed between the client and the server. This eliminates the need for a second round of communication to retrieve the data. However, there are several reasons why embedding large data sets in SOAP envelopes is problematic. One reason that has been observed by others [3,16] is that translating binary data into ASCII for inclusion in the SOAP envelope can add a large overhead to a system. The second reason is that many SOAP implementations have preset limits on the size of SOAP envelopes. Many of our data sets exceed these limits.

For these reasons, O'SOAP enables data sets to be separated from the SOAP request and response envelopes. If a data set is included, it is encoded using XML or Base64 (called, "pass by value"). If it is not included, then a URL to the data set is included (called "pass by reference"). Furthermore, O'SOAP enables clients and servers to dynamically specify whether a data set will be passed by value or reference.

O'SOAP manages a pool of disk space that is used for storing data sets downloaded from the client and data sets generated by the application that will be accessed remotely. O'SOAP currently supports the HTTP, FTP, and SMTP protocols, and we have plans to provide support for IBP [13].

4 Performance

In the previous section, we discussed how O'SOAP generates WSDL automatically, provides mechanisms for transferring large data sets, and enables asynchronous interactions with long running applications. In this section, we will discuss the performance of O'SOAP for a real application.

The Pipe Problem application simulates an idealized segment of a rocket engine modeled after actual NASA experimental spacecraft hardware. The object is a curved, cooled pipe segment that transmits a chemically-reacting, high-pressure, high-velocity gas through the inner, large diameter passage, and a cooling fluid through the outer array of smaller diameter passages. The curve in the pipe segment causes a non-uniform flow field that creates steady-state but non-uniform temperature and pressure distributions on the inner passage surface. These temperature and pressure distributions couple with non-uniform thermomechanical stress and deformation fields within the pipe segment. In turn, the thermomechanical fields act on an initial crack-like defect in the pipe wall, causing this defect to propagate.

The components of this system were deployed on servers at Cornell Computer Science and the Engineering Research Center at Mississippi State University. All components were deployed using O'SOAP, except for one, which used SOAP::Clean [18,2], a predecessor of O'SOAP. All clients were developed using O'SOAP.

To understand how increasing the problem size changes the performance of our system, we ran experiments for three different sizes of the Pipe Problem. The sizes of the meshes for the solid and interior volumes of the Pipe are shown in Table 1.

Problem Size	Solid Mesh vertices	triangles	tet's	Interior Mesh vertices	tri's/quad's	tet's/prisms
1	4,835	4,979	22,045	19,242	3,065	38,220
2	16,832	10,322	83,609	41,216	5,232	85,183
3	54,849	21,127	289,500	79,407	9,074	170,179

Table 1. Pipe Problem Sizes

Problem Size	Local runtime (secs.)	CU Client runtime (secs.)	overhead	UAB Client runtime (secs.)	overhead
1	1630.89	1719.44	5.43%	1695.24	3.95%
2	5593.66	5776.55	3.27%	5745.78	2.72%
3	22202.73	22901.39	3.15%	22222.49	0.09%

Table 2. Pipe Problem Runtimes

Table 2 shows the total running time, in seconds, for the Pipe Problem application. The column labeled "Local runtime" shows the running time when each component it is executed directly, without using the web services infrastructure. These times correspond to the performance of a monolithic application and overheads are measured relative to these times. The columns labeled "CU Client" and "UAB Client" show the running times when the client is run on different machines than the components. The "CU Client" client runs on a machine at Cornell on the same LAN as the Cornell server, and the "UAB Client" client runs on a machine at the University of Alabama at Birmingham. Overall, the total overhead for both clients falls as the problem size increases. The overhead for the largest problem size is 3.2% and 0.1% for the "CU Client" and "UAB Client" clients, respectively.

These results are in marked contrast to the studies in the literature [3,16] that concluded that the use of SOAP and XML adds enormous overhead to computational science applications. Our conclusion is that the organization of a distributed simulation system makes more of a difference to its performance than the underlying web services infrastructure. Tightly-coupled applications appear to perform poorly for large problem sizes, even when a highly optimized web services infrastructure is used. However, loosely-coupled applications, such as ours, appear to perform very well.

A more complete description of the setup, results, and analysis of these experiments can be found in [19].

5 Conclusions and Future Work

In this paper, we have described O'SOAP, a framework that enables legacy applications to be deployed as feature-rich web services without any interface programming. O'SOAP provides automatic WSDL generation, mechanisms for efficient data set transport, and asynchronous client-server interactions. This makes it ideally suited for computational scientists that are not web services programmers to develop distributed component-based and DDDAS applications.

Just as importantly, our experiments have demonstrated that applications can be developed using O'SOAP-based web services without sacrificing performance. In fact, we have shown that, for the Pipe Problem application, the overhead introduced by the O'SOAP framework decreases as the problem size increases. While there will certainly be some DDDAS applications for which O'SOAP is not appropriate, we believe that O'SOAP is perfectly suited for a very large class of DDDAS applications.

While we are greatly encouraged by the features and performance of O'SOAP to date, there are a number of improvements that we plan to make. First, we need to address the two remaining items from the list of requirements given in Section 2. We have implemented WS-Security [1] in SOAP::Clean, the predecessor of O'SOAP; it remains to fold this code into O'SOAP. Also, while O'SOAP has hooks for interfacing with conventional job submission systems, it remains to provide the interfaces for commonly used systems.

One other direction in which O'SOAP can be developed is to expand support for commonly used protocols. For instance, the SOAP standard specifies a means of sending envelops using SMTP (i.e., email). While this transport mechanism is unlikely to deliver the same level of performance as HTTP or TCP, it dramatically lowers the entry point for the DDDAS application developer, because it enables applications to be deployed without the need for any system administrator support. This will make it much easier to develop prototype or "one-off" DDDAS applications.

The Open Grid Services Infrastructure (OGSI) specification [20] has been developed by the Global Grid Forum (GGF) to define basic low-level protocols that Grid services will need to interoperate. OGSI is currently supported by a number of Grid toolkits. IBM, HP and the Globus Project have recently proposed the Web Services Resource Framework (WSRF), a new set of Grid protocols designed to be more compatible with existing web services specifications. At the moment, the evolution of the Grid protocols is a little murky, but once it becomes clearer, O'SOAP can be extended to support the protocols the GGF adopts.

Last, but not least, there are a number of ways in which the performance of O'SOAP can be improved. One way would be to enable O'SOAP-based web services to be deployed as servlets (e.g., using `mod_ocaml` and Apache). While servlets require much more system administrator involvement and support, there are certainly some DDDAS application developers who are willing to incur this cost in return for the performance improvements.

References

1. Bob Atkinson et al. Web services security (WS-Security), version 1.0. Available at `http://www-106.ibm.com/developerworks/webservices/library/ws-secure/`, April 5 2002.
2. Paul Chew, Nikos Chrisochoides, S. Gopalsamy, Gerd Heber, Tony Ingraffea, Edward Luke, Joaquim Neto, Keshav Pingali, Alan Shih, Bharat Soni, Paul Stodghill, David Thompson, Steve Vavasis, and Paul Wawrzynek. Computational science simulations based on web services. In *International Conference on Computational Science 2003*, June 2003.
3. Kenneth Chiu, Madhusudhan Govindaraju, and Randall Bramley. Investigating the limits of soap performance for scientific computing. In *Proceedings of the Eleventh IEEE International Symposium on High Performance Distributed Computing (HPDC'02)*, July 2002.

4. Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (wsdl) 1.1. Available at `http://www.w3.org/TR/wsdl`, March 15 2001.
5. Microsoft Corporation. Microsoft .NET. Accessed February 11, 2003.
6. Robert A. Van Engelen and Kyle A. Gallivan. The gSOAP toolkit for web services and peer-to-peer computing networks. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, page 128, Berlin, Germany, May 21 – 24 2002.
7. I. Foster and C. Kesselman. The globus project: A status report. In *IPPS/SPDP '98 Heterogeneous Computing Workshop*, pages 4–18, 1998.
8. The Apache Foundation. Webservices - axis. `http://ws.apache.org/axis/`.
9. Globus Alliance. The WS-Resource framework. Available at `http://www.globus.org/wsrf/`, January 24 2004.
10. Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. Soap version 1.2 part 1: Messaging framework. Available at `http://www.w3.org/TR/SOAP/`, June 24 2003.
11. The itr/acs adaptive software project for field-driven simulation. Available at `http://www.asp.cornell.edu/`.
12. Paul Kulchenko. Web services for perl (soap::lite, xmlrpc::lite, and uddi::lite). Accessed on June 3, 2003.
13. James S. Plank, Micah Beck, Wael R. Elwasif, Terry Moore, Martin Swany, and Rich Wolski. The internet backplane protocol: Storage in the network. In *NetStore99: The Network Storage Symposium*, Seattle, WA, USA, 1999.
14. Python web services. `http://pywebsvcs.sourceforge.net/`.
15. Didier Rémy and Jérôme Vouillon. Objective ML: An effective object-oriented extension to ML. *In Theory And Practice of Objects Systems*, 4(1):27–50, 1998.
16. Satoshi Shirasuna, Hidemoto Nakada, Satoshi Matsuoka, and Satoshi Sekiguchi. Evaluating web services based implementations of gridrpc. In *Proceedings of the Eleventh IEEE International Symposium on High Performance Distributed Computing (HPDC'02)*, 2002.
17. Paul Stodghill. O'SOAP - a web services framework in O'Caml. `http://www.asp.cornell.edu/osoap/`.
18. Paul Stodghill. SOAP::Clean, a Perl module for exposing legacy applications as web services. Accessed February 11, 2003.
19. Paul Stodghill, Rob Cronin, Keshav Pingali, and Gerd Heber. Performance analysis of the pipe problem, a multi-physics simulation based on web services. Computing and Information Science Technical Report TR2004-1929, Cornell University, Ithaca, New York 14853, February 16 2004.
20. Steve Tuecke et al. Open grid services infrastructure (OGSI) version 1.0. Available at `https://forge.gridforum.org/projects/ogsi-wg/document/Final_OGSI_Specification_V1.0/en/1`, June 27 2003.
21. World Wide Web Consortium. Extensible markup language (xml) 1.0 (second edition). W3C Recommendation, October 6 2000.