Computational Science Simulations based on Web Services

Paul Chew[1], Nikos Chrisochoides[5], S. Gopalsamy[4], Gerd Heber[6], Tony Ingraffea[2], Edward Luke[3], Joaquim Neto[2], Keshav Pingali[1], Alan Shih[4], Bharat Soni[4], Paul Stodghill[1], David Thompson[3], Steve Vavasis[1], Paul Wawrzynek[2]

[1] Department of Computer Science, Cornell University
[2] Department of Civil Engineering, Cornell University
[3] Engineering Research Center, Mississippi State University
[4] School of Engineering, University of Alabama at Birmingham
[5] College of William and Mary
[6] Cornell Theory Center

Abstract. We describe the software architecture of a system for doing multi-physics simulation of a coupled fluid, thermal, and mechanical fracture problem. The system is organized as a collection of geographically-distributed software components in which each component provides a web service, and uses standard web-service protocols to interact with other components. The resulting system incorporates many features such as componentization and geographical distribution which we believe are vital to adaptive and dynamic data-driven application systems (DDDAS).

1 Introduction

Dynamic Data Driven Application Systems (DDDAS) ([10]) are systems in which computational simulation is coupled with real-world experimental data. One example of a DDDAS system is a weather forecasting simulation that periodically uses field observations for corrections. To build a true DDDAS system, we must be able to build systems that adapt continuously to input from sensors and observers, and to changes in computational requirements as the simulation progresses.

One of the goals of the Adaptive Software Project (ASP)[7] is to lay a foundation for building such adaptive systems. In our work, we have identified three levels at which adaptivity occurs in computational science simulations.

Application-level A number of mathematical models (discrete, continuum, etc.) may be available to describe the science of a given problem. A simulation code may find it advantageous to switch adaptively between such models to trade off accuracy for computational time and resources.

Algorithm-level There may be many algorithms for implementing a desired functionality (*e.g.*, direct and iterative solvers for linear systems), and it may be advantageous to switch between algorithms.

[7] Additional information about the ASP project can be found at `http://www.asp.cornell.edu/`.

System-level Changes in the computational environment such as hardware failures may require a simulation system to be adaptive at the system-level.

In this paper, we describe the ASP system which exhibits these three types of adaptivity. In Section 2, we introduce a multi-physics fracture problem from the aerospace domain that we use as a challenge problem. In Section 3, we describe our simulation system, which is organized as a collection of components that interact using web-services implemented on top of standard web-service protocols. We also present the pros and cons of such an architecture. In Section 4, we present some preliminary performance measurements that show that the performance overhead of using web-services is relatively small. We discuss related work in Section 5, and conclude in Section 6.

2 Overview of the Problem

The applications engineers on our research team work in computational fluid mechanics and solid mechanics, so we decided to tackle a problem involving high Reynolds number, chemically reacting gas flow coupled with linear elastic fracture mechanics.

The geometry shown in Figure 1 represents 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 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 (see Figure 2), causing this defect to propagate.

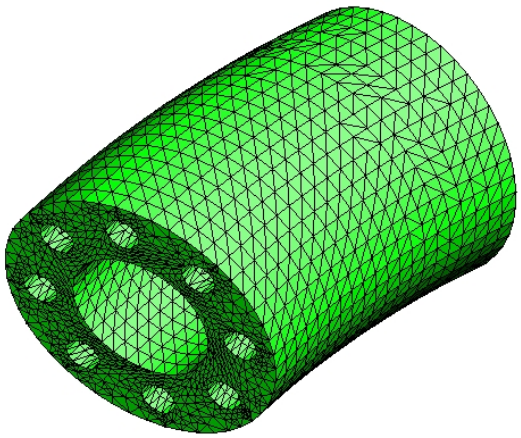

Fig. 1. The Pipe

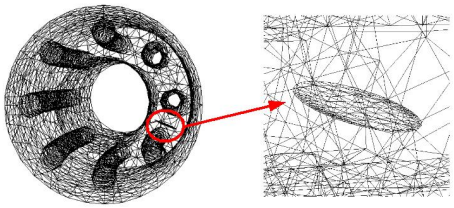

Fig. 2. The Pipe with Crack

2.1 The Workflow

The workflow of a single time step of the simulation is shown in Figure 3. In this figure, the components of our system appear like this, the intermediate data sets appear like *this*, and the "human in the loop" is appear like this. In our current workflow, the only data that is passed from one timestep to the next is the geometric model of the pipe, which is updated in each time step as the defect is inserted and grown.

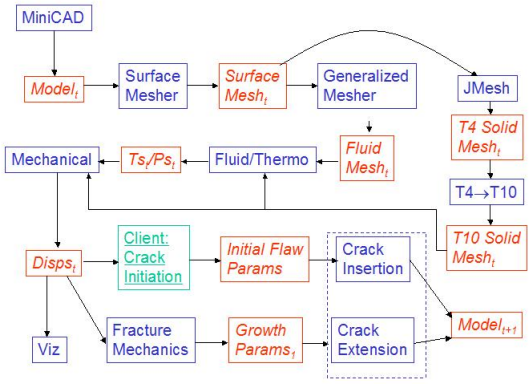

Fig. 3. Workflow for the Pipe problem

Crack initiation is an active area of research in Fracture Mechanics, and, at present, is not understood well enough to do automatically. Hence, in our present system, we require a knowledgeable user to manually determine the Initial Flaw Parameters by studying the displacement field at the end of time step $t = 0$. This is shown as a component labeled `Client: Crack Initiation` in Figure 3. In subsequent timesteps,

t=1,2,..., state of the art fracture mechanics techniques are used to predict growth parameters that determine how the crack defect will grow. At present, a complete simulation of the Pipe Problem consists of three time steps because at the end of the third step, we find that the crack reaches the surface of the component, possibly leading to failure.

2.2 Components

We now describe the components used in the simulation. Many of these components were written as part of the ASP project.

MiniCAD MiniCAD is an integrated environment for creating geometry and topology. It uses Nonuniform Rational B-Splines (NURBS) to represent geometry surfaces. MiniCAD offers several advantages over traditional CAD systems, including being able to guarantee the "watertightness" of the models that it produces. MiniCAD's output is a geometric model in an XML-based representation developed by our project [8].

Surface Mesher Once the geometry model is produced, it is passed to the surface mesher component which produces triangular meshes for each of the model's geometric surfaces. This component produces surface meshes with certain quality guarantees [7].

Generalized Mesher Meshes and grids employed for simulating viscous fluid flows require highly anisotropic elements in regions near no-slip boundaries, *i.e.* boundary layers. For such problems, topological adaptivity - using cell types that are locally appropriate for the region being discretized - offers an attractive alternative to traditional structured multi-block grids or unstructured tetrahedral meshes. Chalasani *et al* [6,5] have exploited this idea to generate high quality meshes consisting of extruded triangular prisms, tetrahedral elements, and generalized prisms.

Jmesh Jmesh [4] generates unstructured tetrahedral meshes for arbitrarily shaped three-dimensional regions, and was designed to handle the unique geometric problems that occur in fracture mechanics. The input for Jmesh is a triangular surface mesh, which describes the domain to be meshed, and the output is a tetrahedral mesh for the solid part of the pipe.

T4 to T10 The "T4 to T10" component converts the volume meshes produces by Jmesh, which use four-noded tetrahedra, into equivalent meshes of ten-noded tetrahedrons.

Fluid/Thermal Simulation Loci [15,14] is a framework for intra-application coordination of fine-grained numerical kernels and methods. The CHEM code [15,16] is a library of Loci rules (fine-grained components) and a front end that generates a component that simulates 3-D chemically reacting flows of thermally perfect, calorically imperfect gases.

Mechanical Simulation The mechanical solver solves the equations of linear elasticity to determine the deformation of the pipe due to different loading conditions (e.g. pressure on the inner pipe) and thermal expansion.

Fracture Mechanics The fracture mechanics component takes as input the volume mesh for the solid part of the pipe and the the nodal displacements computed by the mechanical simulation. It computes the new crack front. This component, as well as a number of other components, uses GGTK [1], a library for manipulating geometric models and for performing geometric operations.

Visualization We have developed an innovative real-time visualization tool for interactive exploration of large-scale 3D solid models and underlying engineering data. Users guide dynamic data extraction by manipulating visual probes and selectors. These choices are automatically translated by the visualization system into SQL queries, which are sent to a parallel database server cluster and which return new features to the user's display.

3 Using Web Services for Simulation

In this section, we describe how the components in our system communicate with each other. Briefly, each component provides web services, implemented using standard web-service protocols, and these services are invoked by any other component that needs to interact with that component.

3.1 Why components?

Components and component frameworks are critical for adaptive and DDDAS systems. Consider, for example, instances of algorithmic adaptivity in which the application switches from one technique to another. If the two implementations of these techniques did not have clearly defined interfaces or did not use similar parameter (or data) types, then it would be impossible to switch between these implementations dynamically.

3.2 Why distributed?

It is likely that many instruments and visualization tools that would be used in a production DDDAS system are geographically distributed. Another reason is that in a multi-disciplinary, multi-institutional project like ours, there are likely to be many different architectures and operating systems in use. It would be a tremendous burden if every developer had to port their code to every other platform. Ideally, a component would be implemented on just one platform, and it would be invoked as needed by project partners. In other words, components should be *write once, run* from *anywhere*.

This has a number of advantages. For the component developer, intellectual property issues become less critical because source code does not have to be released. For component users, the advantage is that they do not have to download source code and install components, nor do they have to find sufficient computational resources to run the components.

3.3 Distributed components lead to web services

The W3C Web Services Architecture Working Group defines the term "web service" as ([12]),

> A web service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.

We have found that frameworks for building and deploying web services are suitable for developing our adaptive simulation systems. In particular, the current system for solving the pipe problem has been developed and deployed using the following web services frameworks.

Microsoft .NET On our Windows platforms, we use Microsoft .NET ([9]), which provides a "holistic" approach to distributed applications.

SOAP::Clean On our UNIX and Linux platforms, we use SOAP::Clean ([20]), a Perl module for exposing legacy applications as web services. Compared with .NET, SOAP::Clean provides a "minimalistic" approach to distributed applications. It is designed to allow existing command-line applications to be exposed as web services after writing a handful of lines of "glue" code, . It also provides client tools that make remote web services appear as local command-line programs.

4 Performance

We have implemented the workflow shown in Figure 3 by deploying each component as a web service on a number of different computers at Cornell Computer Science (CU CS), the Cornell Theory Center (CTC), the Engineering Research Center at Mississippi State University (MSU ERC) and the College of William and Mary (CW&M), as shown in Table 1. The entries that are boxed denote the specific instances of components that were used in these experiments.

4.1 Results

Table 2 shows the individual and total execution times taken by running each of the components using a number of different methods. All times are given in minutes. The columns denote the following execution methods,

Local, non-XML Some of our components consist of application programs that read and write non-standard ASCII file formats. In order to make them interoperable, we "wrapped" them with code that converted between these non-standard formats and our standard XML-based formats. This column contains the execution times of the original application programs running directly on a single machine (i.e., no web services involved). These times represent the expected performance of these modules within a traditional monolithic simulation system.

Component	CU	CS	CTC	MSU	ERC	CW&M
Surface Mesher	yes		no	no		yes
Jmesh	yes		no	no		yes
T4_to_T10	yes		no	no		no
Generalized	yes		no	yes		no
Fluid/Thermal	yes		no	no		no
Mechanical	no		yes	no		no
Fracture Mechanics	yes		no	no		no
Crack Growth	yes		no	no		no

Table 1. Component deployment

	Local, non-XML	Local, XML	Local, WS	Polling freq	Intra-campus	Interstate
Surface Mesher	1.10	1.10	1.15	0.17	1.33	1.43
Jmesh	17.08	16.92	16.98	5.00	20.27	20.67
T4_to_T10	n.a.	0.70	0.77	0.17	1.02	2.22
Generalized(*)	n.a.	0.57	0.57	0.17	0.67	1.35
Fluid/Thermal	23.00	24.13	28.05	5.00	25.42	28.92
Mechanical(*)	n.a.	16.87	17.45	1.00	18.75	n.a.
Fracture Mechanics	n.a.	0.65	0.72	0.17	1.05	n.a.
Crack Growth	n.a.	0.00	0.08	0.00	0.15	n.a.
Total Execution	-	60.97	65.82	-	68.73	-
Overhead	-	0%	8%	-	13%	-

Table 2. Execution times for the Pipe problem

Local, XML This column contains the execution times of all of the component programs, which all use our standard XML-based file formats. Again, these times are from directly running the component programs on a single machine without using web services. This column is used for the base times when computing overheads.

LOCAL, WS This column contains the execution times for invoking the components using web services. For all but two of the components, the web services client and server were run on the same machine, so it was possible for the two to communicate without having to go through the HTTP server. It was not possible to run the Generalized Mesher and Mechanical Simulation in this way, so these two were run according to the "Intra-campus" column.

Intra-campus This column contains the execution times obtained by putting the client on a different machine, within the same campus (Cornell) network, than the server. In this case, all of the communication between the client and the server has to go over a LAN and through an HTTP server. In order to prevent network connections from timing out, the client uses polling to determine when the server has completed execution of the component. The column marked "Polling freq" contains the polling frequency. The Generalized Mesher component resided at MSU ERC, so its time includes a slightly longer network delay.

Interstate This column contains executions times obtained by putting the client on a machine in a different state (Alabama) from all of the servers (Mississippi and New York). The same polling frequencies were used as in the "Intra-campus" cases. These times are incomplete, because we have not completed the experiments, but it should be clear from the "Jmesh" and "Fluid/Thermal" times that they are comparable to the "Intra-campus" times.

There are several observations that we can make from these results. First, we would argue that the total overhead of 13% in the "Intra-campus" case and similar overheads in the "Interstate" case are not excessive. In particular, paying the price of this overhead gives us all of the advantages of web services discussed earlier. Second, the bulk of this overhead is from using the web services frameworks (8%) and not Internet communication (5%) . This is very encouraging, because there are many places in our system (in SOAP::Clean in particular) where there is room for aggressive optimization. Third, we expect that as we scale our experiments to solve larger problems, this overhead should go down.

5 Related Work

A number of frameworks and standards have been proposed for developing component-based systems. Perhaps the best known are CORBA ([19]) and COM ([18]). We investigated using these frameworks, but they did not meet our needs. Using these frameworks required major reengineering of our existing applications to incorporate them into the frameworks. We also found that the existing frameworks were primarily designed for deploying applications within a single machine. DCOM ([17]) is one exception to this. It is also interesting to note that existing component frameworks are evolving towards interoperability with web services (witness .NET subsuming COM and DCOM, and the OMG's adoption of a specification on CORBA-WSDL/SOAP Interworking).

Perhaps the most widely know paradigm for distributed scientific computing is Grid Computing [11] and the associated Globus Toolkit [2]. We did not follow this approach for a number of reasons. The most fundamental reason was that none of the Grid software provided server-side functionality under Microsoft Windows. Given that the Cornell Theory Center has many hundreds of cluster nodes running Windows, this was a show-stopper[8]. Just as importantly, standards for Grid frameworks are only now being developed whereas the web services community has already adopted a number of standards (e.g., XML, SOAP, WSDL) that enable frameworks from different vendors to interoperate.

This is not to say that we will not use Grid computing frameworks at some point in the future. The Core Grid Functionality ([13]) contains many features that we have not implemented within our web services frameworks, and leveraging them seems prudent. Furthermore, the Grid standards being proposed leverage web services functionality such as WSDL. We see Grid and web services converging in some way in the future, although in what form we are not sure.

6 Conclusions

We have argued that component design and distributed computing are fundamental to adaptive and DDDAS simulations, and we have argued that web services provide a natural means of achieving these characteristics. We have described a multi-physics, adaptive simulation system developed along these lines. Preliminary performance results indicate that the overhead of using web services for distributed simulation is not unacceptably high, and certainly worth the benefits and flexibility they give us.

References

1. GGTK home page. Available at "`http://www.erc.msstate.edu/ccs/research/GGTK/`". Accessed February 13, 2003.
2. The Globus Project. Available at "`http://www.globus.org`". Accessed February 13, 2003.
3. Globus Project, Microsoft expand partnership. Available at "`http://www.anl.gov/OPA/whatsnew/020315globus.htm`". Accessed February 13, 2003.
4. J.B. Cavalcante-Neto, P.A. Wawrzynek, M.T.M. Carvalho, L.F. Martha, and A.R. Ingraffea. An algorithm for three-dimensional mesh generation for arbitrary regions with cracks. *Engineering with Computers*, 17:75–91, 2001.
5. S. Chalasani and D. Thompson. Quality improvements in extruded meshes using topologically adaptive generalized elements. *International Journal for Numerical Methods in Engineering*, (submitted).
6. S. Chalasani, D. Thompson, and B. Soni. Topological adaptivity for mesh quality improvement. In *Proceedings of the 8th International Conference on Numerical Grid Generation in Computational Field Simulations*, Honolulu, HI, June 2002.
7. L. P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the Ninth Symposium on Computational Geometry*, pages 274–280. ACM Press, 1993.

[8] This issue is being addressed[3]

8. L. Paul Chew, Stephen Vavasis, S. Gopalsamy, TzuYi Yu, and Bharat Soni. A concise representation of geometry suitable for mesh generation. In *Proceedings, 11th International Meshing Roundtable*, pages pp.275–284, Ithaca, New York, USA, September 15-18 2002. Available at "`http://www.imr.sandia.gov/papers/imr11/chew.pdf`".
9. Microsoft Corporation. Microsoft .NET. Available at "`http://microsoft.com/net/`". Accessed February 11, 2003.
10. Craig Douglas, Abhi Deshmukh, et al. Report from the March 8-10, 2000 NSF sponsored workshop on Dynamic Data Driven Application Systems. Available at "`http://www.cise.nsf.gov/eia/dddas/dddas-workshop-report.htm`". Accessed February 8, 2003.
11. Global Grid Forum. Global Grid Forum home page. Available at "`http://www.ggf.org/`". Accessed February 13, 2003.
12. W3C Web Services Architecture Working Group. Web services glossary W3C working draft 14 November 2002. Available at "`http://www.w3.org/TR/ws-gloss/`". Allen Brown and Hugo Haas, eds. Accessed February 8, 2003.
13. W. Johnston and J. Brooke. Core Grid Functions: A minimal Architecture for Grids. GGF Draft Document. Available at "`http://www.ggf.org/meetings/ggf6/ggf6\protect\unhbox\voidb@x\kern.06em\vbox{\hrulewidth.3em}wg\protect\unhbox\voidb@x\kern.06em\vbox{\hrulewidth.3em}papers/CoreGridFunctions.v3.pdf`". Accessed February 13, 2003.
14. E. Luke. Loci: A deductive framework for graph-based algorithms. In S. Matsuoka, R. Oldehoeft, and M. Tholburn, editors, *Third International Symposium on Computing in Object-Oriented Parallel Environments*, number 1732 in Lecture Notes in Computer Science, pages 142–153. Springer-Verlag, December 1999.
15. E. A. Luke. *A Rule-Based Specification System for Computational Fluid Dynamics*. PhD thesis, Mississippi State University, 1999.
16. E. A. Luke, X.L. Tong, J. Wu, L. Tang, and P. Cinnella. A step towards "shape-shifting" algorithms: Reacting flow simulations using generalized grids. In *Proceedings of the 39th AIAA Aerospace Sciences Meeting and Exhibit*. AIAA, 2001. AIAA-2001-0897.
17. Microsoft, Inc. Distributed component object model (DCOM). Available at "`http://www.microsoft.com/com/tech/dcom.asp`". Accessed February 13, 2003.
18. Microsoft, Inc. Microsoft COM technologies. Available at "`http://www.microsoft.com/com/`". Accessed February 13, 2003.
19. Object Management Group, Inc. Welcome to the OMG's CORBA website. Available at "`http://www.corba.org/`". Accessed February 13, 2003.
20. Paul Stodghill. SOAP::Clean, a Perl module for exposing legacy applications as web services. Available at "`http://www.asp.cornell.edu/SOAP-Clean/`". Accessed February 11, 2003.