Grid services for earthquake science

Geoffrey Fox[1,*,†], Sung-Hoon Ko[2], Marlon Pierce[2], Ozgur Balsoy[3], Jake Kim[3], Sangmi Lee[3], Kangseok Kim[4], Sangyoon Oh[4], Xi Rao[4], Mustafa Varank[4], Hasan Bulut[5], Gurhan Gunduz[5], Xiaohong Qiu[5], Shrideep Pallickara[5], Ahmet Uyar[5] and Choonhan Youn[5]

[1]*Departments of Computer Science and Physics, School of Informatics, Community Grids Laboratory, Indiana University, IN, U.S.A.*
[2]*School of Computational Science and Information Technology, Florida State University, FL, U.S.A.*
[3]*Computer Science Department, Florida State University, FL, U.S.A.*
[4]*Computer Science Department, Indiana University, IN, U.S.A.*
[5]*Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, U.S.A.*

SUMMARY

We describe an information system architecture for the ACES (Asia–Pacific Cooperation for Earthquake Simulation) community. It addresses several key features of the field—simulations at multiple scales that need to be coupled together; real-time and archival observational data, which needs to be analyzed for patterns and linked to the simulations; a variety of important algorithms including partial differential equation solvers, particle dynamics, signal processing and data analysis; a natural three-dimensional space (plus time) setting for both visualization and observations; the linkage of field to real-time events both as an aid to crisis management and to scientific discovery. We also address the need to support education and research for a field whose computational sophistication is rapidly increasing and spans a broad range. The information system assumes that all significant data is defined by an XML layer which could be virtual, but whose existence ensures that all data is object-based and can be accessed and searched in this form. The various capabilities needed by ACES are defined as grid services, which are conformant with emerging standards and implemented with different levels of fidelity and performance appropriate to the application. Grid Services can be composed in a hierarchical fashion to address complex problems. The real-time needs of the field are addressed by high-performance implementation of data transfer and simulation services. Further, the environment is linked to real-time collaboration to support interactions between scientists in geographically distant locations.

KEY WORDS: Web services; collaboration; computational portal; earthquake science

1. ACES GRID AND .OPENNET GRID ARCHITECTURE

We consider an ACES [1] computational environment (ACESCE) built in terms of a Web-based user interface accessing service, which is built in a broker-based fashion [2]. The client machine contacts a server that acts as an intermediary to back-end resources and also as a conduit for clients to access services. One can also view the brokers as middleware wrappers that allow a heterogeneous collection of resources to be accessed in a relatively uniform fashion. In the simplest technology, these brokers or wrappers would be implemented as a Perl CGI program running on a Web server. As discussed later, there are more sophisticated approaches but the basic model is correct; ACESCE consists of Web clients connecting to a collection of Web servers, which host a collection of resources. In Figure 1, we illustrate this with a particular set of resources; ground and satellite sensors, field data, computers, software and compiled geophysical data such as the positions of faults. The user uses a portal (described in Section 2) to access a set of services, which roughly correspond to the servers of the simple model described above—one server per resource [3]. The overall environment can be termed as a Web or a grid. The services available to users can be divided into two. Firstly, we have the 'system' or general services such as security (authentication, authorization and communication encryption) and collaboration, which are important to most application areas. Then we have the more application-specific services such as those of Figure 1. Here, some services are very specific to this application area (field data and geophysical fault data), some are very general (such as simulation), while others are specializations of general services. In Figure 1, we show a general sensor service used by two application-specific sensors for which it must be specialized. Again the application software service would be specialized into those especially important for ACES. These could be a Green's function solver or a finite-element solver service linked to earthquake-specific kernels. Visualization and information services are also general capabilities which could be specialized to this field.

The host computer and software services would be invoked by other services—especially the simulation service. This process is described in Section 2 and is itself further broken up into other services corresponding to parameter specification, login, execution, job status, etc. Any interesting task typically involves multiple services—for example, the visualization service might access the sensor data, the geophysical database and the simulation service. We do not show all the possible links in Figure 1 and they are left implicit. ACES has the opportunity to develop a next generation computational environment built around such interacting Web services. Each service can be thought of as a component in the general software engineering sense and more specifically as a component such as is defined by the major DoE-led Common Component Architecture (CCA). The CCA, described in [4], is developing high-performance components aimed at scientific computing. We expect the CCA to be compatible with the WSDL (Web Services Definition Language), which is the current industry standard for Web-based components. The WSDL [5,6] is being developed as an XML-based framework that can describe distributed objects built from any of the major approaches (SOAP, CORBA, Java) and allows one to define input and output data streams with a mix of transport protocols. Thus, it enables one to build networks of heterogeneous services, which interoperate with well-defined interfaces. This is the library or component model for Grid or Web programming. The WSDL is augmented by other important and still-developing technologies. For example UDDI [7] allows registration and discovery services and the WSFL [5] describes the linking of services together [8]. The W3C standard SOAP protocol is becoming very popular as a generic XML transport layer to be used in Web services when performance is not critical [9]. A key feature of the WSDL is its support of multiple transport protocols

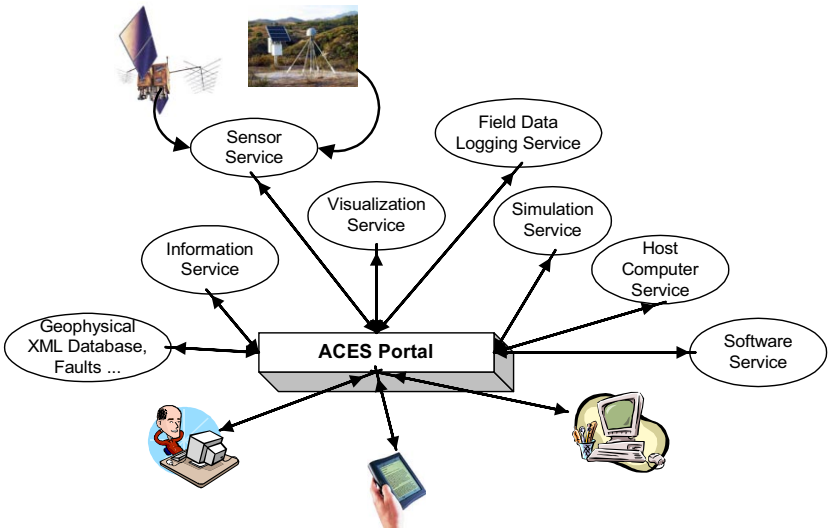

Figure 1. Service model for ACESCE.

with a common application interface; this way we can choose between, say, the flexibility of SOAP and the performance of GridFTP [10].

This architecture of interlinked Web modules has some generally attractive features—all components have Web views making it easier to document them, while the universality of the Web allows us to implement this model on essentially any distributed system. One must break an application into modules (components) carefully. Smaller components are easier to maintain, but all components must also interact via communication channels. Typically these communication channels correspond to an overhead that increases as the modules get smaller and the ratio of edge (communication) to volume (computation) increases. The situation is further exacerbated for high-performance problems where parallel algorithms usually require low latency; the bandwidth of Internet and intranet connections is rapidly increasing, but the latency of Web service component communication is likely to be in the 200 microsecond to one millisecond range—one hundred times slower than that of a shared memory or dedicated parallel computing system. Thus, one should carefully evaluate where to break one's system into Web components and keep these reasonably coarse grained. So in ACES, one probably would not make an adaptive mesh as a Web service, but rather bundle it with the solver as a parallel finite-element solver Web service. However, one would take separate simulations (say particle dynamics and fast multipole Green's function solver) and make these separate services. Similarly, pattern dynamics analysis would be a Web service that can be used either on empirical data or on the results of a simulation. We would design a standard interface for such data analysis systems and so allow different users to build and test modules with this functionality. Image processing modules would be treated in a similar way; there will be a generic image processing Web service, which is subclassed for different

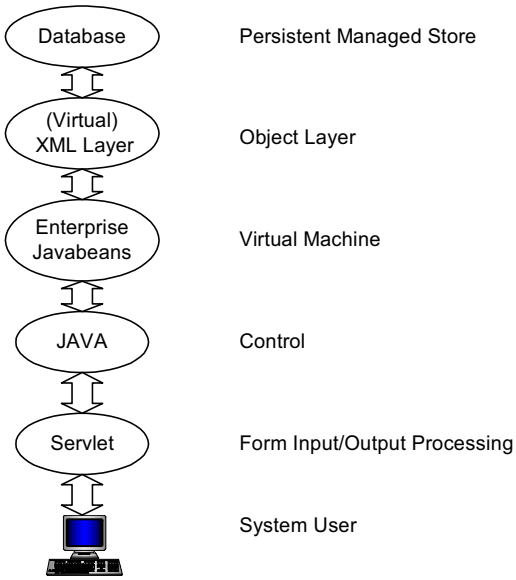

Figure 2. Six-layer architecture for GEM software.

algorithms. Analysis of a particular image could require piping it through multiple such services. We need research to see how far one can go—for instance, can a friction model be made a Web component?

If we look at the special features of the ACES applications, we see the need for multi-scale and multi-disciplinary simulations. The service model naturally supports the multi-disciplinary requirement as one builds complex applications out of, say, separate particle dynamic and finite-element components. Multi-scale simulations can also exploit this feature and the availability of general services (like visualization), which can be shared by multiple simulations. One can build a simulation out of the different types of services needed and then substitute in different components corresponding to say different approaches with different algorithms or different resolutions. This capability of supporting different 'plug-and-play' versions is also important in education as discussed in the next section. One can substitute smaller data sets or simpler software to enable a classroom version of an ACES simulation.

In Figure 2, we show the key features of a typical implementation of what we sometimes call .opennet—the collection of open Web technologies which can be used to build robust multi-tier systems. The simple client–broker–resource triplet is a three-tier model; however, once we link multiple services and build hierarchical service bundles we get a general multi-tier model. The model of Figure 2 builds modularity into the software model. Databases are used to store and support access and search of data, but they do not define the structure. The data structures are defined in XML, which has the important implication that all data is now viewed as an object. Later, in Section 4, we discuss in detail the potential use of XML in ACES. We term the XML layer in Figure 2 as virtual because we do

not need to turn all data into an XML syntax—that would often be very inefficient. Rather, we need to be able to reference the data with XML query languages and manipulate it as though it had the XML form. In our implementations of this architecture, we use Castor to automatically generate Java classes equivalent to the XML Schema object specification. As is discussed in Section 4, we suggest that the earthquake community develop appropriate XML schema to describe those quantities that are characteristic of their field. This should be built on activities in related fields and on relevant general standards.

Section 2 describes the ACES portal and how it can support both research and education while in Section 3 we describe how one can share resources and build a collaborative environment. Section 4 describes the way XML can be used by ACES. Note that we can see two facets of interoperability in ACESCE; macroscopically the grid service distributed object architecture supports this while 'in the small' the use of XML to define object properties is the key enabling technology. Systematic use of Java to build the middleware gives ACESCE good software engineering and portability features. Conclusions are given in Section 5.

2. COMPUTATIONAL WEB PORTALS

A computing Web portal, as shown in Figure 3, is designed to simplify remote access to computing resources. Typically, high-performance computing centers are interested in outreach to potential new users. The problem faced in doing this is that many of these users are unfamiliar with the peripheral details of using these machines: using the Unix operating system, creating and submitting batch scripts to queuing systems, transferring files, etc. All of this is in addition to problems associated with learning to use a new code. These difficulties are further compounded by the introduction of grid technologies for distributing jobs among several institutions. None of these problems singularly is insurmountable, but taken together they can be very frustrating for new users and force them to become experts in particular computer operating systems instead of allowing them to focus on scientific and engineering tasks.

These usage problems apply equally well to the educational community. Computing techniques have become important in a wide range of disciplines and high-quality commercial and academic codes are available. Instructors must, however, devote time to teaching students esoteric operating system details. The limited student–instructor interaction time would be better spent teaching the students about the different computational techniques that are available, the appropriate problem domain for each technique, and the actual business of solving problems with the correct application. Matlab is a well known portal to areas like linear algebra and signal processing and illustrates some of the basic ideas, which Web portals try to generalize [11].

One solution that computing centers have chosen in order to simplify access is the development of computational Web portals. Typically these can be grouped as either *system portals* or *application portals*. The former are geared toward assisting users to remotely login and use general resources at the computing center through a browser interface. The latter are more specialized browser portals devoted to particular codes. Typical services provided by system portals include:

(1) secure login, access control and authorization;
(2) information services describing available host computers and applications;

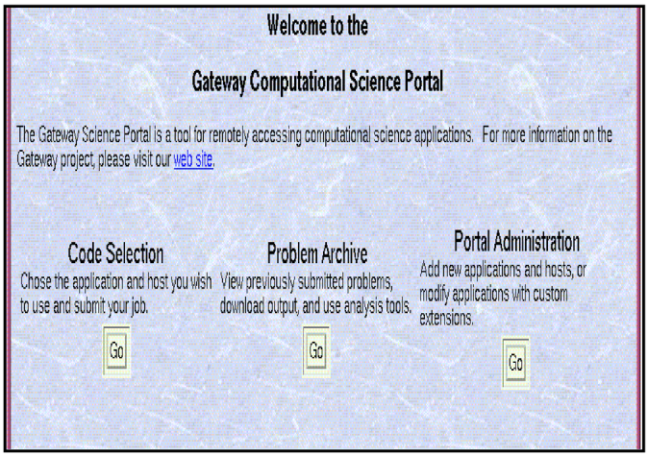

Figure 3. Gateway's 'welcome page' provides an entry point for users and administrators.

(3) job submission and monitoring;
(4) file transfer;
(5) remote file access and manipulation; and
(6) session archiving.

By session archiving, we refer to the ability of the user to revisit old sessions, edit the parameters of that session and resubmit that job. A simple interface for a session archive is shown in Figure 4. Application portals might provide all of these services plus additional services specific to the code, such as input file creation.

We have developed a system portal, called Gateway [12–14], for the Department of Defense's High Performance Computing Modernization Program. Several similar projects are under development at many computing centers, and descriptions and additional references may be found at the Grid Computing Environments Web site [15].

These portals can play an obvious role in education. Because they hide the details of using remote computers with a particular operating system behind a browser-based user interface, students can choose applications, submit jobs and analyze output by using a simple point-and-click interface. These portals can also play an important role in distance education, simplifying access for students taking the class remotely. The browser interface can be easily augmented with online documentation and examples. A more sophisticated interface may provide expertise in helping students choose the correct codes for their particular problem.

Application portals can be built on top of the basic services of system portals. For example, Gateway has been designed to be application-neutral, making it simple to add new applications to the portal. Gateway tools are also modular with well-defined interfaces, so developers wishing to add more sophisticated user interfaces to create application portals can easily integrate these Web pages

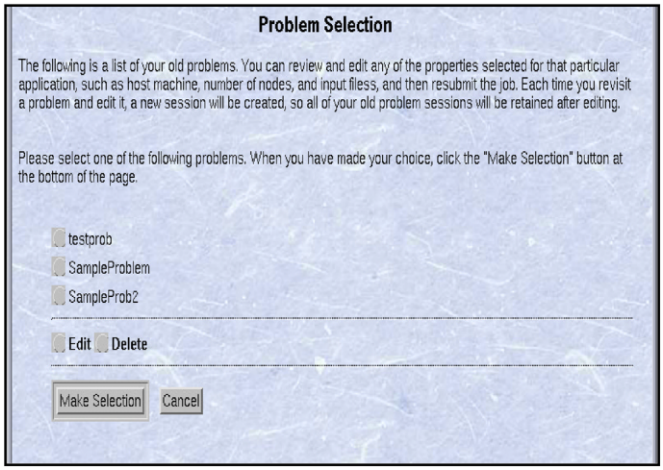

Figure 4. Users can access old problem sessions for editing and resubmission.

into the system portal. Other portal projects, such as NPACI's HotPage [16], similarly provide base functionality that can be extended for specific applications [17].

Computing portals for education possess a slightly different focus than computing portals for working scientists and researchers. First, collaboration and shared control of the input pages are important. When giving initial instructions on setting up input decks and running codes, instructors will need to be able to share displays (in the fashion described below) with all students (especially remote students) to show them the steps involved. For post processing and visualization, instructors and students will want to share visualization so that typical problems, such as common mistakes in input decks that produce invalid results, can be identified. Secondly, the portal must have multiple user privilege levels. The instructor, for instance, will need to be able to examine the students' problem archives and assume control over applications started by students, but students should not be allowed to access instructor areas. Thirdly, problem archiving acquires a new usage and would benefit from different access permission levels. Instructors, for example, will want to create a series of sample input problems for the students to run and modify.

3. COLLABORATIVE PORTAL

One of the general services introduced in Section 1 was collaboration. This is the capability for geographically-distributed users to share information and work together on a single problem. The basic distributed object and Web service model described in Section 1 allows one to develop a powerful collaborative model. In fact, one of the attractive features of the Web and distributed objects is the natural support of asynchronous collaboration. One can post a Web-page or host a Web service and then others can access it on their own time. Search and registration capabilities such as those provided

Figure 5. Access grid at Indiana with some of the authors.

by UDDI are key to a good asynchronous environment. XML is also an important technology as it can build metadata to describe resources. This metadata will enable more precise search methods as envisaged by the Semantic Web [18,19].

Asynchronous collaboration, as enabled by the basic Web infrastructure of Section 1, must be supplemented by synchronous or real-time interactions between the ACES community members. The field of synchronous collaboration is very active at present and we can identify several important areas:

(1) basic interactive tools including text chat, instant messenger and white boards;
(2) shared resources including shared documents (e.g. PowerPoint presentations), as well shared visualization, earthquake maps, or data streaming from sensors; and
(3) audio–video conferencing illustrated by both commercial systems and the recent high-end access grid from Argonne [20] shown in Figure 5.

There are several commercial tools that support (1) and (2)—Centra, Placeware and WebEx are best known [21–23]. They look similar to the screen in Figure 6—a shared document window surrounded by windows and control panels supporting the collaborative function. All clients are presented the same or a similar view and this is ensured by an event service that transmits messages whenever an object is updated. There are several ways objects can be shared as we now describe.

Shared display. The master system brings up an application and the system shares the bitmap-defining display window of this application [24]. This approach has the advantage that essentially all applications can be shared and the application does not need any modification. The disadvantage is that faithful sharing of dynamic windows can be CPU intensive (on the client holding the frame-buffer). If the display changes rapidly, it may not be possible to accurately track this and, further, the network traffic could be excessive as this application requires relatively large messages to record the object changes.

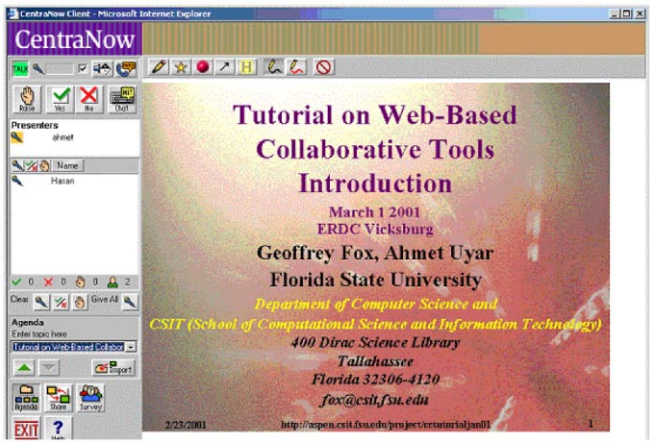

Figure 6. Typical shared document system from Centra.

Native shared object. Here, one changes the object to be shared so that it generates messages defining its state changes. These messages are received by collaborating clients and used to maintain consistency between the shared object's representations on the different machines. In some cases this is essentially impossible, as one has no access to the code or data-structures defining the object. In general, developing a native shared object is a time consuming and difficult process. It is an approach used if you can both access the relevant code and if the shared display option has the problems alluded to earlier. Usually this approach produces much smaller messages and lower network traffic than the shared display—this or some variant of it (see below) can be the only viable approach if some clients have poor network connectivity.

Shared export. This applies the above approach, but chooses a client form that can be used by several applications. Development of this client is still hard, but worth the cost if useable in many applications. For example, one could export applications to the Web and build a general shared Web browser, which in its simplest form just shares the defining URL of the page. The effort in building a shared browser can be amortized over many applications. We have built quite complex systems around this concept—these systems track frames, changes in HTML forms, JSP (Java Server Page) and other events. Note the characteristic of this approach—the required sharing bandwidth is very low but one now needs each client to use the shared URL and access common (or set of mirrored) servers. The need for each client to access servers to fetch the object can lead to substantial bandwidth requirements, which are addressed by the static shared archive model described below. Other natural shared export models are PDF, SVG, Java3D or whatever formats the users scientific visualization system uses.

Static shared archive. This is an important special case of shared export that can be used when one knows ahead of time what objects are to be shared. All that changes in the presentation is the choice of object and not the state within the object. The system downloads copies of the objects to participating

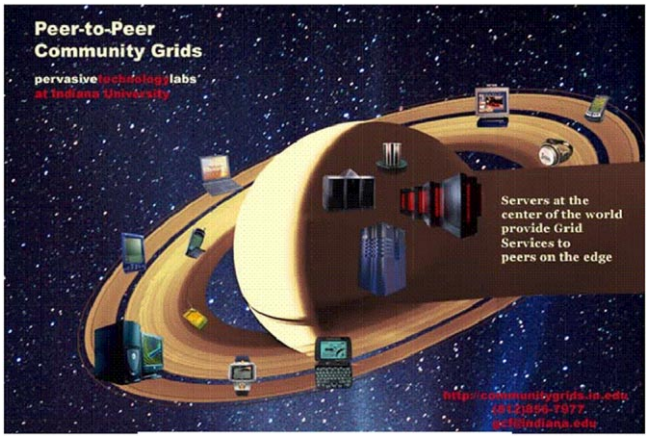

Figure 7. A NASA P2P grid.

clients (these could be URLs, PowerPoint foils or Word documents). Sharing requires synchronous notification as to which of the objects to view. This is the least flexible approach but gives in real-time, the highest quality with negligible real-time network bandwidth. This approach requires substantially more bandwidth for the archive download—for example, exporting a PowerPoint foil to JPEG or Windows Meta File (WMF) format increases the total size but as we described, can be done before the real-time session.

It can be noted that in all four approaches, sharing objects does not require identical representations on all the collaborating systems. Even for the shared display, one can choose to resize images on some machines—we do this for a palmtop device with a low-resolution screen sharing a display from a desktop. In Figure 7 we illustrate this, showing a collection of clients (peers) supported by central servers, which provide grid resources and control the collaborative synchronization process. Real-time collaborative systems can be used as a tool in earthquake science in three different modes as follows.

(a) Traditional scientific interactions—seminars, brainstorming, conferences—but done at a distance. Here the easiest to implement are structured sessions such as seminars.
(b) Interactions driven by events (earthquakes, need to respond to an error-condition in a sensor) that require collaborative scientific interactions, which must be at a distance to respond to a non-planned event in a timely fashion. Note, this type of use suggests the importance of collaborating with diverse clients—a key expert may be needed in a session, but he or she may only have access through a PDA.
(c) As well as scientific interactions in an earthquake, collaborative technology can be and is used to manage and enhance the response to the crisis. The first collaborative system that we built TangoInteractive [25,26] was in fact designed for command and control operations, which is the military equivalent of crisis management. It was later evolved to address scientific collaboration and distance education [27,28].

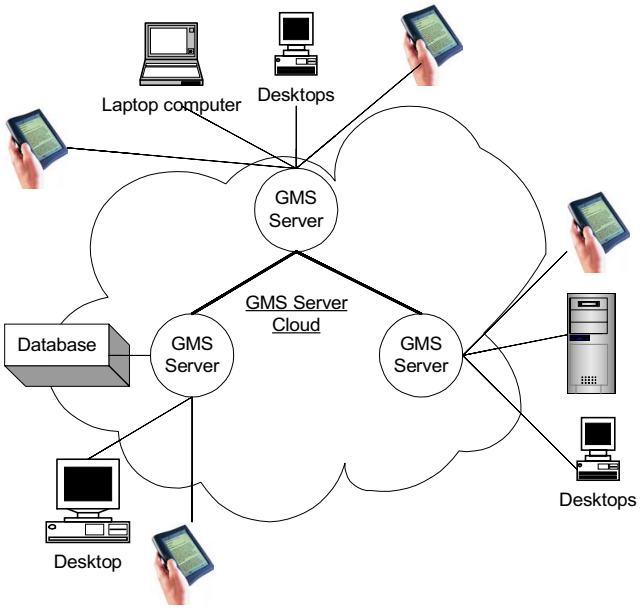

Figure 8. Distributed GMS architecture.

Areas (b) and (c) are characteristic for this field while (a) and (b) are relevant for this paper. ACES has some special needs that would suggest custom collaborative applications—for instance, special native shared event or shared export applications. We need to share geographical information systems (GIS) or equivalent 2D and 3D approaches for representing maps and related data. This could involve either a detailed sharing at something like the openGIS level [29] or, in a less custom fashion, sharing of the export of a GIS to a standard visualization format [30,31]. We are developing a shared SVG browser as the new SVG standard has some very attractive features [30]. It is a 2D vector graphics standard, which allows hyperlinked 2D canvases with a full range of graphics support—Adobe Illustrator supports it well. SVG is a natural export format for 2D maps on which one can overlay simulations and sensor data. As well as its use in 2D scientific visualization, SVG is a natural framework for high-quality educational material—we are building a filter that automates the PowerPoint to SVG conversion and already one can achieve this by the complex PowerPoint to WMF (Windows Metafile) to Illustrator to SVG export pipeline.

There are some important new developments in collaboration that come from the Peer-to-Peer (P2P) networking field [32]. Traditional systems such as TangoInteractive and our current Garnet environment [33] have rather structured ways of forming communities and controlling them with centralized servers. The P2P approach [34], exemplified by Napster, Gnutella and JXTA [35], uses search techniques with ‘waves of agents’ establishing communities and finding resources. P2P and Grid ideas can be usefully combined as a Peer-to-Peer Grid [36] shown in Figures 7 and 8. We expect

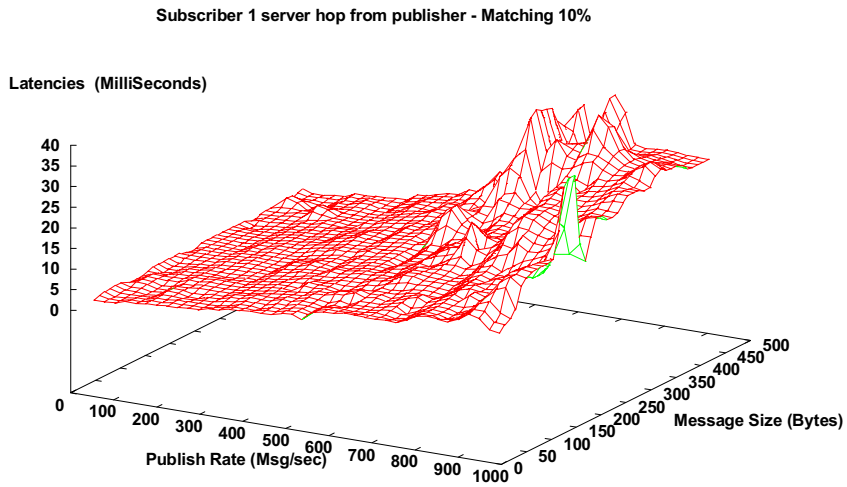

Figure 9. Latencies for a GMS prototype.

these developments to be important in all scientific areas, with the application to real-time communities centered on earthquake events as particularly important for ACES.

Our Garnet system uses a central publish–subscribe server for coordinating the collaboration with the current implementation using a commercial JMS (Java message service) [37] system. This has proved very successful, with JMS allowing the integration of real-time and asynchronous collaboration with a more flexible implementation than the custom Java server used in TangoInteractive.

However, our use of the publish/subscribe model is rather different than that for which JMS was developed and we have proposed some extensions which we have prototyped in GMS—the Grid message or event service [38]. GMS was first described in the PhD thesis of Pallickara [39]. We suggest that GMS needs the following capabilities.

- The matching of published messages with subscribers is based on the comparison of XML based publisher topics or advertisements (in a JXTA parlance) with XML based subscriber profiles.
- The matching involves software agents and not just SQL-like property comparisons at the server, as used by JMS.
- GMS servers form a distributed network with servers created and terminated as needed to get high-performance fault-tolerant delivery.

The GMS server network is illustrated in Figure 8 where each cluster of clients instantiates a GMS server. The servers communicate with each other while P2P methods are used within a client subgroup. Figure 9 illustrates some results from our initial research where we studied the message delivery latency as a function of load. We found that the distributed network scaled well with adequate latency (a few milliseconds) unless the system became saturated. The distributed cluster architecture allows the GMS service to support large heterogeneous client configurations that scale to arbitrary size.

We mentioned audio–video conferencing earlier in this section, where we have used a variety of commercial and research tools with the access grid as the preferred high-end system (Figure 5). We are investigating using the Grid service ideas of Section 2 to build a grid conferencing service with audio–video systems using the publish–subscribe metaphor to post to a Web service that integrates the different systems using standards like H323 and SIP.

4. XML DESCRIPTORS OF DATA STRUCTURES

A crucial problem for developing information technology-based tools for earthquake science is the definition of data structures that describe and organize the metadata associated with the field. Here, it is important to distinguish between the raw data generated either by codes or by scientific instruments and the metadata that describes the raw data. The metadata is appropriately described by a specialized XML dialect. XML has the advantage of being human-readable and hierarchically organized, but is verbose and thus not ideal for very large datasets. Instead it is more often useful to have the XML metadata description point to the location of the data and describe how that data is formatted, compressed and to be handled. This is related to the Virtual XML architecture described in Figure 2. Let us first consider an example of using XML data from computing portals and then examine some of the specific issues that will need to be addressed by the earthquake science community.

4.1. XML use in Gateway

Computational Web portals are described in more detail in Section 2 in this paper, but one may consider them in summary to be browser-based systems for accessing computing resources for composing and submitting jobs and monitoring their progress. Numerous supporting services to this basic concept can be defined, such as security, file transfer, resource monitoring and selection and session archiving. Many computing portal projects are underway and a partial listing can be found at the Grid Computing Environments Web site [15]. The Gateway Web portal is one such project.

XML metadata descriptions form the basis for Gateway and are used to describe static data about host machines and codes. These data, in turn, can be used to generate browser forms in the user interface and to construct requests for backend resources. Here, static data means data that should remain relatively constant. This is somewhat idealized but is distinguished from dynamic data, which by definition will change every time a user accesses the Web portal. For example, the location of the executable for a particular code on a particular machine is static data, but the actual code and machine a user selects in a particular session, as well as his or her input file and code parameters, are dynamic.

Let us now examine this in practice. For Gateway, we have defined three sets of static data: code descriptions, host descriptions and service descriptions. For the first two we have chosen to use XSIL, an XML dialect for the description of scientific data. We determined that this approach had sufficient flexibility to be extended to the description of codes that would use scientific data, as well as the data itself.

XSIL: A convenient XML dialect

In developing our XML descriptions for Gateway we were motivated by a desire to move quickly and so we decided to adopt XSIL (eXtensible Scientific Interchange Language) developed by Roy Williams

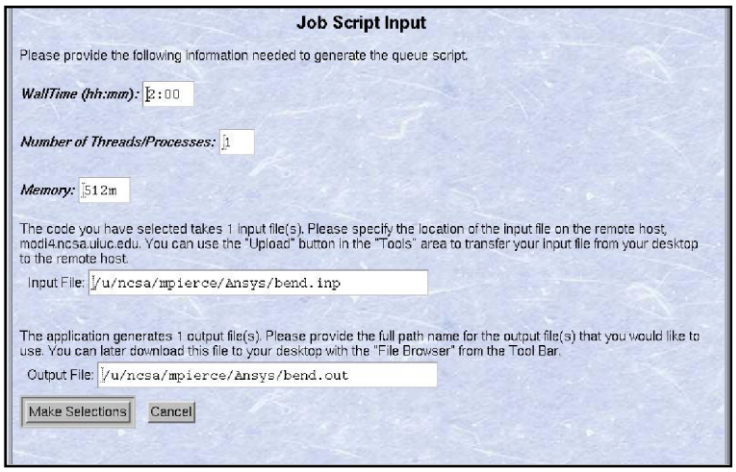

Figure 10. Job input forms are generated using the HPC and application descriptors.

at CalTech [40]. XSIL is primarily designed to describe scientific data, but we found it to be generally useful and to provide a single solution for both scientific and non-scientific data. XSIL comes with software (in Java) for parsing documents and extracting name–value pairs from the XML data. XSIL also allows you to identify in the XML the piece of Java code that you wish to handle a particular set of tags, which we found to be quite useful. There are other important approaches to the description of scientific data, including the ICE project at the Army Research Laboratory [41]. Likewise, the Castor project described in Section 1 can be used to automatically generate the XML-handling code.

Application description

First, we should clarify our use of the word application. We use this term to refer specifically to third party codes, whatever they may be (scientific and engineering codes such as Gaussian, visual analysis tools such as Gnuplot or MatLab and so on). All of these have common characteristics for running on a command line, so in our application description we seek to capture this information in a XML data record. Dynamically generated Web forms, such as the one shown in Figure 10, can then be generated from this descriptor. The code for generating the pages (in this case, Java code in a JavaServer page) can be reused to create pages for many different codes.

For a particular application, we need to capture at least the following to run it:

1. the number of input files the code takes;
2. the number of input parameters the code takes;
3. the number of output files the code generates;
4. the number of output parameters the code generates (for symmetry); and
5. the input/output style the code uses.

By input and output files, we refer specifically to data files. Parameters are anything else that you might need to pass to the code, such as the version of the code to use, the number of nodes to use in parallel computation, a user-written Fortran subroutine to dynamically link and so on. I/O style is typically either by standard Unix redirects, < and >, or C-style command line arguments.

The following is the application description for ANSYS, a structural mechanics code:

```
<XSIL Name="ANSYS" Type="csm.parseXMLDesc">
    <Param Name="NumberOfInParams">0</Param>
    <Param Name="NumberOfInFiles">1</Param>
    <Param Name="NumberOfOutParams">0</Param>
    <Param Name="NumberOfOutFiles">1</Param>
    <Param Name="IOStyle">StandardIO</Param>
        ...
```

The 'Type' attribute of the <XSIL> tag specifies the code that extracts this information from the XML file and makes it available to other components. In this example, it is parseXMLDesc, a custom written Java class that extracts the name/value pairs from the XML document and defines accessor (getter) methods to be used by other components of the portal to retrieve the information in the descriptor.

We have not attempted to be complete in this description, but rather are motivated by the requirements of the codes we currently need to support. One of the advantages of using XSILs 'shallow' tree structure is that it is simple to add further parameter tags as required. Code command line flags are an obvious additional parameter we would want to provide. This is simply a parameter again and the parseXMLDesc code is general and does not care what name and value we provide.

HPC description

We have developed a description of HPC systems using the same viewpoint as our application description. We primarily want to capture enough information to generate a queue script so that the code can run on a particular machine. For each application, we need a further description of all the host machines on which that application can run and the details for executing the code on that particular platform. This again is stored in an XML descriptor file that can be used to automatically generate Web forms. For example, as shown in Figure 11, this record can be used to generate a list of codes and hosts that are available in the portal.

Let us now examine the minimal contents of a host descriptor. We take as an example the ANSYS application on Modi4 at NCSA. This can be described by the following descriptor.

```
<XSIL Name="Modi4 Type="csm.parseXMLHost">
    <Param Name="HostName">modi4.ncsa.uiuc.edu</Param>
    <Param Name="QueueType">LSF</Param>
    <Param Name="ExecPath">/usr/apps/fe/bin/ansys57</Param>
    <Param Name="WorkDir">/scratch</Param>
    <Param Name="QsubPath">/usr/local/bin/bsub</Param>
...
```

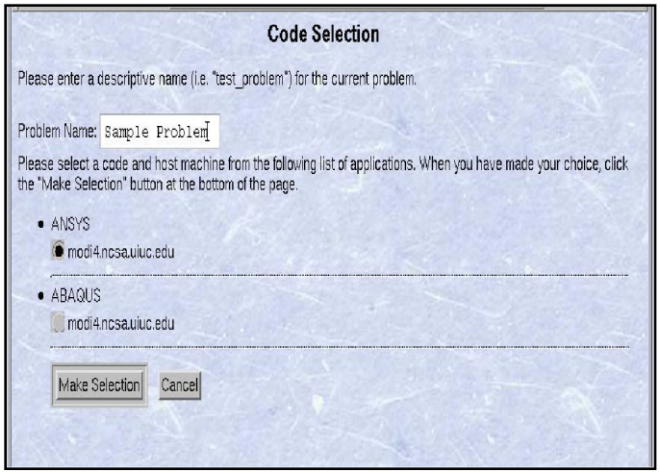

Figure 11. The 'code selection' page lists available applications and hosts.

Again, we use a shallow tree description. The handler code (parseXMLHost) does not care what name/value pairs we give for a particular parameter, so we can add as many additional parameters to our list as required. For example, if an application needs to have a number of environment variables set in its queue script file before it can run on a particular host, we can add these to the description list.

Service description

We have identified a number of generic services that we wish to implement in our portal, such as job submission and file transfer. These are implemented using WebFlow (Java and CORBA-based middleware). However, we believe the services to be general and so the interface to a particular service should be independent of the implementation. Thus, all computational portals could potentially use the same interface description for a particular set of services and any particular portal could radically redesign its middleware without changing the user interface. This will be possible once the community develops WSDL-based portal standards and is the first step towards portal interoperability.

The following is an example of the XML interface we use for job submission. For WebFlow, this must be translated into CORBA's Interface Definition Language (IDL), which motivated our tag naming.

```
<interface name="submitJob" extends="BeanContextChild">
  <method return="void" name="test"></method>
    <method return="string" name="execLocalCommand">
      <arg in="string">command</arg>
    </method>
    <method return="string" name="execRemoteCommand">
```

```
      <arg in="string">host</arg>
      <arg in="string">user</arg>
      <arg in="string">command</arg>
      <arg in="string">carrier</arg>
    </method>
    <method return="string" name="copyFileFromBackend">
      <arg in="string">options</arg>
      <arg in="string">user</arg>
      <arg in="string">host</arg>
      <arg in="string">remoteFile</arg>
      <arg in="string">localFile</arg>
      <arg in="string">carrier</arg>
    </method>
    <method return="string" name="copyFileToBackend">
      <arg in="string">options</arg>
      <arg in="string">localFile</arg>
      <arg in="string">user</arg>
      <arg in="string">host</arg>
      <arg in="string">remoteFile</arg>
      <arg in="string">carrier</arg>
    </method>
</interface>
```

4.2. XML descriptors for earthquake science

Successful XML schema [42] development is a community process that is best done under the auspices of standards-setting organizations within a particular field. Problems exist with this approach because there are often multiple stake-holding organizations, introducing the possibility of multiple, incompatible 'standards'. Federating these groups presents an additional challenge: large, multiple group consortia often lack the 'nimbleness' to quickly develop and test straw man schemas. Smaller groups may possess the required nimbleness, but lack the authority to see their schemas widely adopted.

Consider the problem of developing schemas for earthquake science [43]. Stake-holding groups include, but are not limited to, ACES, EarthScope [44,45], GEM [46], IRIS [45], the Southern California Earthquake Center [47] and the United States Geological Survey [48]. Any and all of these organizations may develop schemas, but for interoperability and data sharing, these efforts must eventually be standardized. However, a consortium of these groups potentially suffers from the problems outlined above in developing schemas. Perhaps the better procedure is to have smaller, more focused groups develop rapid prototype schemas that they can test and refine. This prototype can then serve as the basis for later, official standards. It is also important that related schemas be considered and adopted if appropriate. For example, for the case of earthquake science, related efforts include GML, the Geography Markup Language [49] and XMML, the Exploration and Mining Markup Language [50]. It is important that the new schemas standards build upon earlier efforts and avoid duplication.

Some of the capabilities of XML schemas [42,51] can simplify the process. First, XML namespaces can be used to resolve potential future conflicts in the tag naming process. For example, if the GEM group decides to develop a prototype schema, it can define its own namespace, say 'GEMRP' for GEM Rapid Prototype. All tag definitions within this schema then fall within this namespace. Thus, conflicts with other definitions can be automatically resolved. It is also perhaps politically expedient, since it immediately tells anyone viewing marked-up data that this is the GEM group's attempt at a definition of, for example, strainmeter data, so confusion with other groups' efforts at a standard will be avoided and there is no presumption that this is the standard definition of strainmeter data. Successful tag definitions can later be promoted to a more official namespace.

Namespaces also have the advantage of allowing other work to be folded into a particular XML data description. For example, developers of a rapid prototype schema for earthquake science will find tag definitions in other schemas such as GML that they will want to use. Namespaces allow these tags to be directly imported into the prototype data descriptions.

Another advantage of using XML schemas for data definitions is their simple inheritance model. This simplifies the prototyping process because tag definitions do not have to be complete. The prototype version can be general, with specific biases towards the developing group's area of interest. As the schema is refined and moves towards becoming a standard, refined tag definitions can inherit, from the prototype, definitions without invalidating data described in the prototype's language. Furthermore, subgroups needing more specialized tag definitions can extend the general schema definitions to adequately represent their more specialized description requirements.

Now we will consider some specific data that must be described. First we will take a holistic approach and consider everything of potential interest [43]. This organizational structure can be mapped into an XML data tree as follows.

- Researchers
- Publications
- Institutions
  - Universities
    - Jet Propulsion Laboratory
  - Government Agencies
    - US Geological Survey
  - Research Organizations
    - Jet Propulsion Laboratory
    - Los Alamos National Laboratory
  - Collaborative Groups
    - ACES
    - GEM
  - Scientific Societies
  - For-Profit Corporations
- Data
  - Units
  - Observational Data
    - Seismic
      - Siesmicity

- Standard Processed Data
- Reprocessed Data
- Focal Mechanisms
- Waveforms
- Paleoseismic

- Geodetic
  - GPS
  - INSAR
  - VLBI
  - Surveying
    - Leveling
    - Triangulation
    - Trilateration
  - Creepmeeters
- Stress–strain
  - Strainmeter Data
  - Stress Measurements
- Gravity

- Simulation Data
  - Seismic
    - Seismicity
    - Waveforms
    - Focal Mechanism
    - Paleoseismic
  - Geodetic
    - Displacement and velocity fields
    - Fault Slip Rates
  - Stress–Strain
  - Gravity

- Earth
  - Proper Geographic Names
    - Regions
    - Countries
    - States
    - Cities
    - Geologic Entities with Proper Names
      - Faults
      - Volcanoes
      - Rivers
      - Mountains
      - Basins
    - Earth Structures
      - Point Entities
        - Hypocenters

- Epicenters
- Linear Entities
- Surfaces
  - Faults
  - Strata Boundaries
  - Seismic Discontinuities
- Volume Entities
  - Seismic Velocity
  - Seismic Attenuation
  - Density
  - Pore Pressure
  - Electrical Conductivity
  - Magnetic Properties
  - Rock Type

- Geological Events
  - Earthquakes
  - Tsunamis
  - Volcanic Eruptions

- Devices
  - Computer resources
  - Instruments
    - Earth sensors
      - Seismic graphs
      - GPS Receivers
      - VLPB Antennae
      - Creepmeters
      - Strainmeters
    - Laboratory
      - Rock Mechanics
      - Analog Models
  - Observatories
    - Boreholes
- Computing applications
  - Simulation Methods
    - Finite Element Methods
    - Finite Difference Methods
    - Boundary Element Methods
    - Mesh definitions
  - Data Analysis
  - Visualization

Given the expansiveness of information that needs to be described, the next step is to decide the appropriate scope of the prototype schema. The first points to eliminate are those that have been covered

by other groups. Several groups have developed descriptions of people and institutions—one example is the IMS project for education [52,53], publications can be described using XML standards such as the Dublin Core [54] and RDF [55] and many groups have described computing resources and applications (such as is described in Section 4.1 of this paper). Suggested areas of concentration then are the areas specific to earthquake science, particularly the data and devices sections above.

As a gauge for determining what is in scope and what is out, it will also be useful to have specific applications in mind. For example, a potential application might be to use observational data within a specific set of analysis and visualization tools. In this case, a common data format is needed to serve as a middle ground between measured data and applications. New measurements records may be written into this format directly and application tools may be modified to accept the standard format. However, legacy formats will have to be supported, so the common data format will need support tools for conversion between it and legacy data representations and input formats. A related use to consider is the coarse-grained coupling of applications, in which the output of one code can be formatted and used as the input for another code. Here the common data format and conversion tools serve as the glue for the coupling and future versions of the codes can be redesigned to use the new data format.

5. CONCLUSIONS

We have postulated in Section 1 distributed service components for ACESCE, which satisfy among other things, the following requirements: a common metadata description language that can be used to describe the services and how they are to be accessed; a service lookup and discovery system so that clients can find appropriate services; a way for describing workflow that links together various service components into a single meta-service; and wire protocols for accessing remote objects. As described above, standards and associated software development kits are being developed for each of these areas. The Web Services Description Language (WSDL) is the industry standard for describing services. SOAP is the universal transport protocol to be enhanced when it becomes necessary to achieve high performance. Workflow (WSFL) and discovery (UDDI) capabilities are less far along, but powerful systems in these areas will surely emerge.

This macroscopic framework is joined by ‘in the small’ technologies XML and Java to produce powerful interoperable modular systems. XML needs special attention from the ACES community to define discipline-specific standards and to participate in the evolution of related standards such as those in the GIS field. In Section 3, we showed how the integration of resources inherent in the Grid can be enhanced by the integration of people or the construction of community grids.

We note that there are several important problems in Web services for scientific use that are not being addressed by the commercial world, even though the development of WSDL is being driven by IBM and Microsoft for e-commerce applications. For example, what happens if a user’s data file is 10 gigabytes or larger in size? It may not be a good idea to use a visualization service in Australia (or anywhere else) if that data sits at JPL. This type of consideration impacts the way we integrate services together. Also, supercomputers are fragile and go down often (as do networks) so some robustness (or quality of service) is very important; perhaps even more so than in business-to-business Web services.

Scientific instruments can be on the Web service grid and these have many interesting requirements for our Grid Web services system. For example, earthquake events are rare but important, so the Web service grid needs a good event model. Or researchers may want to use instruments in real-

time computation and visualization, producing a situation similar to synchronous collaboration. This is different from the more conventional time-independent view of Web services. These are some of the research issues we will be addressing.

ACKNOWLEDGEMENTS

Developments of the collaborative portal and Gateway computational Web portal were partially funded by the High Performance Computing Modernization programs and we gratefully acknowledge their support. Funding from the Jet Propulsion Laboratory and the NSF through the PACI Partnership program at NCSA partially supported the ACES architecture and collaboration work.

REFERENCES

1. ACES Asia-Pacific Cooperation for Earthquake Simulation. http://www.quakes.uq.edu.au/ACES/ [2001].
2. Fox GC, Hurst K, Donnellan A, Parker J. Introducing a new paradigm for computational earth science—A Web-object-based approach to earthquake simulations. *GeoComplexity and the Physics of Earthquakes*, Rundle J, Turcotte D, Klein W (eds.). 2000; 219–245.
3. Fox GC. Portals and frameworks for Web based education and computational science. *Proceedings of the Second International Conference on the Practical Application of Java*, Manchester, 2000.
4. Presentation on Common Component Architecture by Robert Armstrong of Sandia at DoE Components Workshop July 23–25, 2001. Livermore, CA. http://www.llnl.gov/CASC/workshops/components_2001/viewgraphs/RobArms trong.ppt.
5. Presentation on Web Services by Francesco Curbera of IBM at DoE Components Workshop July 23–25, 2001. Livermore, CA. http://www.llnl.gov/CASC/workshops/components_2001/viewgraphs/Francis coCurbera.ppt.
6. WSDL Web Service Framework. http://www.w3.org/TR/wsdl.
7. UDDI Universal Description and Discovery Framework. http://www.uddi.org.
8. Mehrotra P and colleagues. Arcade Computational Portal. http://www.cs.odu.edu/~ppvm.
9. XML based messaging and protocol specifications SOAP. http://www.w3.org/2000/xp.
10. Gannon D, Bramley R. Research on multi-protocol Web Services. http://www.extreme.indiana.edu.
11. The Mathworks Corporation. Matlab Computational framework. http://www.mathworks.com.
12. The Gateway Computational Web Portal. http://www.gatewayportal.org.
13. Fox G, Haupt T, Akarsu E, Kalinichenko A, Kim K, Sheethalnath P, Youn C. The Gateway system: Uniform Web based access to remote resources. *ACM Java Grande Conference*. ACM Press: New York, 1999.
14. Pierce M, Fox G, Youn C. The Gateway computational Web portal. *Concurrency and Computation: Practice and Experience* 2002. To be published.
15. Grid Computing Environments. http://www.computingportals.org.
16. NPACI HotPage. https://hotpage.npaci.edu.
17. Thomas M, Mock S, Dahan M, Mueller K, Sutton D, Boisseau J. The GridPort toolkit: A system for building grid portals. *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing*, August 2001. IEEE Press: New York, 2001.
18. Semantic Web from W3C to describe self organizing Intelligence from enhanced Web resources. http://www.w3.org/2001/sw.
19. Berners-Lee T, Hendler J, Lassila O. The semantic Web. *Scientific American* May 2001.
20. Argonne National Laboratory. Access Grid. http://www.mcs.anl.gov/fl/accessgrid.
21. Centra Collaboration Environment. http://www.centra.com.
22. Placeware Collaboration Environment. http://www.placeware.com.
23. WebEx Collaboration Environment. http://www.webex.com.
24. Virtual Network Computing System (VNC). http://www.uk.research.att.com/vnc.
25. Beca L, Cheng G, Fox G, Jurga T, Olszewski K, Podgorny M, Sokolowski P, Walczak K. Java enabling collaborative education health care and computing. *Concurrency and Computation: Practice and Experience* 1997; **9**(6):521–533.
26. Beca L, Cheng G, Fox G, Jurga T, Olszewski K, Podgorny M, Walczak K. Web technologies for collaborative visualization and simulation. *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM: New York, 1997.

27. Fox G, Podgorny M. Real time training and integration of simulation and planning using the TANGO interactive collaborative system. *International Test and Evaluation Workshop on High Performance Computing*. IEEE Press: New York, 1998.
28. Fox G, Scavo T, Bernholdt D, Markowski R, McCracken N, Podgorny M, Mitra D, Malluhi Q. Synchronous learning at a distance: Experiences with TANGO interative. *Supercomputing 98 Conference*. IEEE Press: New York, 1998.
29. Open GIS Consortium. Open Geodata Interoperability Specification. http://opengis.net/gml/ [2001].
30. W3C Scalable Vector Graphics Standard SVG. http://www.w3.org/Graphics/SVG.
31. X3D. http://www.web3d.org/x3d.html.
32. openp2p. http://www.openp2p.com.
33. Fox G. Report on architecture and implementation of a collaborative computing and education portal. http://aspen.csit.fsu.edu/collabtools/updatejuly01/erdcgarnet.pdf [2001].
34. Fox G. Peer-to-Peer Networks. *Computing in Science & Engineering* 2001; **3**(3):75–77.
35. Sun Microsystems JXTA Peer to Peer technology. http://www.jxta.org.
36. Fox G, Gannon D. Computational grids. *Computing in Science & Engineering* 2001; **3**(4):74–77.
37. Sun Microsystems. Java Message Service. http://java.sun.com/products/jms.
38. Fox G, Pallickara S. An event service to support grid computational environments. *Concurrency and Computation: Practice and Experience* (Special Issue on Grid Computing Environments) 2002. To be published.
39. Pallickara S. A grid event service. *PhD Thesis*, Syracuse University, 2001.
40. XSIL: Extensible Scientific Interchange Language. http://www.cacr.caltech.edu/SDA/xsil.
41. ICE Interdisciplinary framework and eXtensible Data model and Format (XDMF). http://www.arl.hpc.mil/SciVis/dice/ and http://www.arl.hpc.mil/PET/training/SEM77.html.
42. XML Schema. http://www.w3c.org/XML/Schema.
43. Tullis T. Private Communication, August 2001.
44. EarthScope. http://www.earthscope.org.
45. The IRIS Consortium. http://www.iris.edu.
46. General Earthquake Models. http://geodynamics.jpl.nasa.gov/gem/.
47. The Southern California Earthquake Center. http://www.scec.org.
48. United States Geological Survey. http://www.usgs.gov.
49. Geography Markup Language. http://opengis.net/gml/01-029/GML2.html.
50. Exploration and Mining Markup Language. http://www.ned.dem.csiro.au/XMML.
51. XML Schema Primer. http://www.w3c.org/TR/xmlschema-0/.
52. Instructional Management Systems (IMS). http://www.imsproject.org.
53. Advanced Distributed Learning Initiative. http://www.adlnet.org.
54. The Dublin Core Metadata. http://dublincore.org/.
55. Resource Description Framework (RDF). http://www.w3.org/TR/REC-rdf-syntax [2001].
56. Castor Java XML Linkage. http://castor.exolab.org/.