Community Grids

Geoffrey Fox[1,2,5], Ozgur Balsoy[1,3], Shrideep Pallickara[1], Ahmet Uyar[1,4]
Dennis Gannon[2], and Aleksander Slominski[2]
[1]Community Grid Computing Laboratory, Indiana University
gcf@indiana.edu, spallick@indiana.edu
[2]Computer Science Department, Indiana University
gannon@cs.indiana.edu, aslom@cs.indiana.edu
[3]Computer Science Department, Florida State University
ozgur@csit.fsu.edu
[4]EECS Department, Syracuse University
auyar@syr.edu
[5]School of Informatics and Physics Department, Indiana University

Abstract. We describe Community Grids built around Integration of technologies from the peer-to-peer and Grid fields. We focus on the implications of Web Service ideas built around powerful event services using uniform XML interfaces. We go through collaborative systems in detail showing how one can build an environment that can use either P2P approaches like JXTA or more conventional client-server models.

1. Introduction

The Grid [1-5] has made dramatic progress recently with impressive technology and several large important applications initiated in high-energy physics [6,7], earth science [8,9] and other areas [29,30]. At the same time, there have been equally impressive advances in broadly deployed Internet technology. We can cite the dramatic growth in the use of XML, the "disruptive" impact of peer-to-peer (P2P) approaches [10,11] and the more orderly but still widespread adoption of a universal Web Service approach to Web based applications [12-14]. We have discussed this recently [15,16] with an emphasis on programming environments for the Grid [17]. In particular we described the important opportunities opened up by using Web service ideas as the basis of a component model for scientific computing [18-22]. This builds on the DoE Common Component Architecture (CCA). This paper also discussed the implications for portals and computational science applications on the Grid. In the following, we look at other facets of the integration of Grid with P2P and Web Service technology. In particular we discuss the overall architecture in section 2 with attention to the implications of adopting a powerful event service as a key building block [23-24]. Web services are discussed in section 3 with special focus on the possibility of building science and Engineering as a Web Service – what can be termed e-Science. P2P technologies are very relevant for collaboration [25,26] and we discuss this in section 4; an area addressed for the Grid [31], including a seminal paper by Foster and collaborators [27] addressing broad support for communities. Section 5 gives more detail on our proposed event model, which integrates both P2P and more traditional models – in particular, that of the commercial Java Message Service [28].

2. Architecture

We view the "world" as built of three categories of distributed system components: raw resources, clients and servers shown in fig. 1. These describe the different roles of machines in our distributed system. Clients provide user interfaces; raw resources

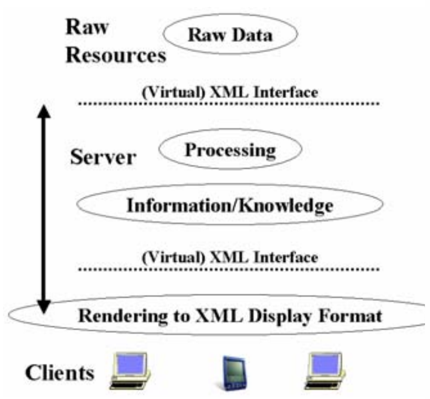


Fig. 1: XML-based Architecture

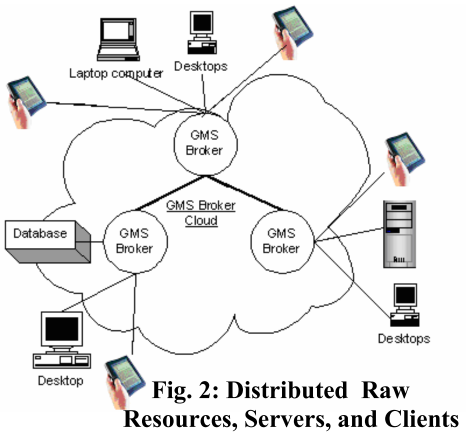


Fig. 2: Distributed Raw Resources, Servers, and Clients

provide "raw data" either from simulations or data sources; servers map between clients and raw resources and are specified by two XML specified interfaces; that between raw resource and server and that between client and server. Actually exploding the "server" layer inside fig. 1 finds an interlinked set of servers each of which linkage is itself described by XML interfaces. Note that the three functions can be thought of as roles and a given computer can have one, two or all of these three roles. Our architecture then should be termed as a three-role model rather than the more traditional three-tier model used in many current systems. We need to view our system in this way because in a peer-to-peer (P2P) system [11,23], one does not see the clear identification of machine and roles found in a classic Grid application involving say a workstation client going through some middleware to clearly identified back-end supercomputers.

The components of our system of whatever role are linked by message passing infrastructure shown in fig. 2. This we also term the event-bus and it has significant features, which we will elaborate later. We assume that all messages will be defined in XML and the message infrastructure – called GMS (Grid Message Service) in fig.

2 – can support the publish-subscribe mechanism. Messages are queued by GMS from "publishers" and then clients subscribe to them. XML tag values are used to define the "topics" or "properties" that label the queues. We can simplify and abstract the system as shown in figs. 3 and 4. We have divided what is normally called Middleware into two. There are routers and/or brokers whose function is to distribute messages between the raw resources, clients and servers of the system. We consider that the servers provide services (perhaps defined in the WSDL [12] and related XML standards) and do NOT distinguish at this level between what is provided (a service) and what is providing it (a server). Actually the situation is even simpler as shown in fig. 4. All entities in the system are resources labeled in the spirit of W3C [32,33] by URI's of form *gxos://category/someheader/blah/…/blah/foo/bar/leaf* and resources communicate by events. We do not distinguish between events and messages; an event is defined by some XML Schema including a time-stamp but the latter can of course be absent to allow a simple message to be thought of as an event. Note an event is itself a resource and might be archived in a database raw resource. Routers and brokers actually provide a service – the management of (queued events) and so these can themselves be considered as the servers corresponding to the event or message service. Note that in fig. 1, we call the XML Interfaces "virtual". This signifies that the interface is logically defined by an XML Schema but could in fact be implemented differently. As a trivial example, one might use a different syntax with say <*sender*>*meoryou*</*sender*> replaced by *sender:meoryou* which is an easier to parse but less powerful notation. Such simpler syntax seems a good idea for "flat" Schemas that can be mapped into it. Less trivially, we could define a linear algebra web service in WSDL but compile it into method calls to a Scalapack routine for high performance implementation. This compilation step would replace the XML SOAP based messaging [34] with serialized method arguments of the default remote invocation of this service by the natural in memory stack based use of pointers to binary representations of the arguments.

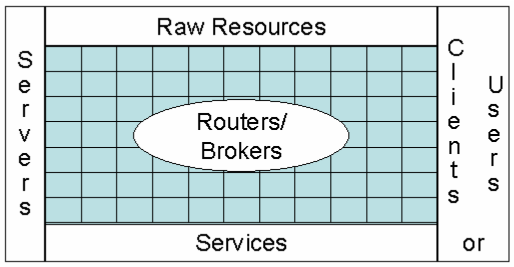


Fig. 3: One View of System Components

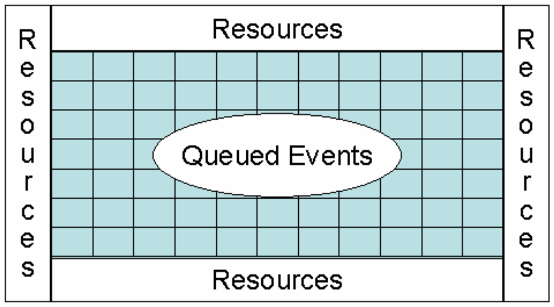


Fig. 4: Simplest View of System Components

In the next four subsections we summarize some features and issues for the four components of the architecture events/messages, clients/users, servers/services and raw resources.

2.1 Event and Message Subsystem

We discuss the event or message service further in Sec. 5 but we elaborate first on our choice of this as an essential feature. We see several interesting developments in this area where we can give four examples: there is SOAP messaging [34]; the JXTA peer-to-peer protocols [10]; the commercial JMS message service [28]; and finally a growing interest in SIP [35] and its use in instant messenger standards [36]. All these approaches define messaging principles but not always at the same level of the OSI stack; further they have features that sometimes can be compared but often they make implicit architecture and implementation assumptions that hamper interoperability and functionality. We suggest breaking such frameworks into subsystem capabilities describing common core primitives. This will allow us to compose them into flexible systems, which support a range of functionality without major change in application interfaces. Here SOAP defines a message structure and is already a "core primitive" as described above; it is "only" XML but as discussed above, a message specified in XML could be "compiled to other forms such as RMI either for higher performance or "just" because the message was linking two Java programs. In some of our work, we use publish-subscribe messaging mechanisms but of course this is often unnecessary and indeed occurs unacceptable overhead. However it does appears useful to define an event architecture such as that of fig. 5, allowing communication channels between Web services which can either be direct or pass through some mechanism allowing various services on the events. These could be low-level such as routing between known source and destination or the higher-level publish-subscribe mechanism that identifies the destinations for a given published event. Some routing mechanisms in peer-to-peer systems in fact use dynamic routing mechanisms that merge these high and low level approaches to communication. We use the term virtual queue in fig. 5 because again we can in many cases preprocess (or "compile") away the queue and transmit messages directly. As an example, consider an audio-video conferencing web service. It would use a simple publish/subscribe mechanism to advertise the availability of some video feed. A client interested in receiving the video would negotiate (using the SIP protocol perhaps) the transmission details. The video could either be sent directly from publisher to subscriber; alternatively from publisher to web service and then from web service to subscriber; as a third option, we could send from the web service to the client but passing through a filter that converted one codec into another if required. In the last case, the location of the filter would be negotiated based on computer/network performance issues – it might also involve proprietary software

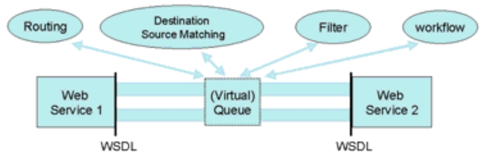


Fig 5: Communication Model showing Sub-services of Event Service

only available at special locations. The choice and details of these three different video transport and filtering strategies would be chosen at the initial negotiation and one would at this stage "compile" a generic interface to its chosen form. One could of course allow dynamic "run-time compilation" when the event processing strategy needs to change during a particular stream. This scenario is not meant to be innovative but rather to illustrate the purpose of our architecture building blocks in a homely example. Web services are particularly attractive due to their support of interoperability, which allows the choices described. One could argue that the complexity of fig. 5 is unnecessary as its "luxury" features are an unacceptable overhead. However as the performance of networks and computers increase, this "luxurious" approach can be used more and more broadly. For instance, we have shown that JMS (the Java Message Service which is a simple but highly robust commercial publish-subscribe mechanism) can be used to support real-time synchronous collaboration. Note this application only requires latencies of milliseconds and not the microseconds needed by say MPI for parallel computing. Thus JMS with a message-processing overhead of a millisecond in good implementations can be used here. As explained in Sec. 5, we have developed a system that allows general XML-based selection on our message queues and this generalizes the simple topic and property model of JMS. The XML selection can specify that the message be passed though a general Web service, and so the "subscription" mechanism supports both event selection and filtering of the messages. In the collaboration application, this mechanism allows the same event service to support multiple clients – say a hand-held device and a high-end workstation – which would need different views (versions) of an event.

2.2 Clients

In the "pure view" of the architecture of the previous two sections, the traditional workstation (desktop) client has at least two roles – rendering and providing services. There has been a trend away from sophisticated clients (Java Applets and complex JavaScript) towards server based dynamic pages using technologies like ASP and JSP (Active and Java Server Pages). This reflects both the increased performance of networks and the more robust modular systems that can be built in the server-based model. There are several good XML standards for rendering – XHTML [37] and SVG [38] define traditional browser and 2D vector graphics respectively. These are the XML display formats of fig. 1; their support of the W3C Document Object Model [39] (DOM) allows both more portable client-side animation and shared export collaboration systems described in sec. 4. We do not expect this to replace server side control of "macroscopic" dynamic pages but built in animation of SVG (which can be considered a portable text version of Flash technology) and scripted event control will allow important client side animation. XML standards on the client should also allow universal access and customization for hand-held devices -- possibly using the WML [40] standard for PDA's and VoiceXML [41] for (Cell)-phones. The 3D graphics standard X3D [42] is likely to be another important XML rendering standard.

2.3 Servers and Services

Servers are the most important feature of our community grid architecture. They "host" all application and system services ranging in principle from Microsoft word through a 1024 node parallel simulation. They have multiple input and output ports defined by (virtual) XML Interfaces. There has been substantial work on many system Services in both the Grid and broader communities. We find Object registration, lookup and persistence; security, fault tolerance, information and collaboration. There are also the set of services common in computing environments such as Job submission, transparent login, accounting, performance, file access, parameter specification, monitoring, and visualization. An online education system using this architecture would have curriculum authoring, course scheduling and delivery, registration, testing, grading, homework submission, knowledge discovery, assessment, and learning paths as some of the services. We see current typically monolithic systems being broken up into small Web services and this will enable easier delivery of capabilities customized to particular communities. As one makes the basic services "smaller", flexibility increases but typically performance suffers as communication overhead increases. Here efficient "compilation" techniques will be important to use the optimal implementation of communication between the ports (see fig. 5) of linked web services. Static "compilation" can be supplemented by dynamic choice of communication mechanism/stub that will use the optimal solution (based on many criteria such as bandwidth, security, etc.). Looking at peer-to-peer technology, we see important issues as to the appropriate implementation infrastructure. Is it to be based on a relatively large servers or a horde of smaller peers.

2.4 Raw Resources

The XML interfaces exhibited by servers represented "knowledge" or processed data and have typically "universal" form. The raw resources – databases, sensors, and supercomputers – also use XML interfaces but these can reflect nitty gritty detail. One sees approaches like JDBC (databases), SLE (spacecraft sensors [43]), HDF (scientific data) and MathML. The first two are true interfaces, the last two "raw" data format.

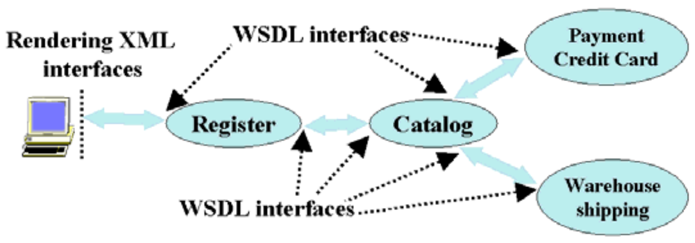


Fig. 6. An Online Shopping system with component Web Services

3. Web Services

Web Services are being developed actively by many major companies (Ariba, IBM, Microsoft, Oracle, Sun) with the idea typified in fig. 6 of componentizing Business to Business and Business to Customers applications.
We suggest that a similar approach is useful in both Grid system services shown in table 1 but also more generally to develop „Science as a Web Service" – one could term the latter e-Science.

Table 1. Basic Grid services

Security Services	Authorization, authentication, privacy
Scheduling	Advance reservations, resource co-scheduling
Data Services	Data object name-space management, file staging, data stream management, caching
User Services	Trouble tickets, problem resolution
App Services	Application tracking, performance analysis
Monitoring Service	Keep-alive meta-services

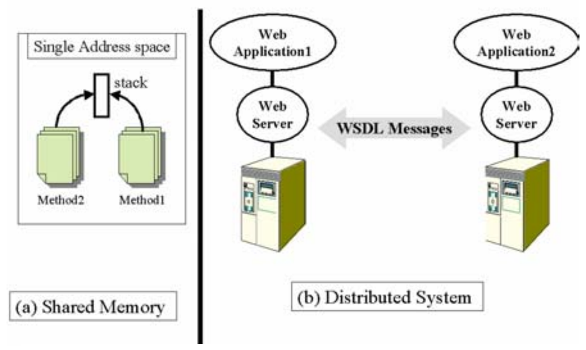


Fig. 7. Schematic of two bindings of a Web Service (a) Single address Space and (b) Distributed

We see WSDL [12-14] – the Web Services Definition Language – as a well thought through proposal. It is incomplete in some ways and more research is needed to decide how best to enhance it in such areas as the integration of multiple services together to form composite systems. Figure 6 shows 4 component Web services, which integrate to form an online shopping Web Service. WSFL [44] and WSCL [45] are candidate integration standards but another possibility is to build a programming environment on top of basic XML (for data) and WSDL (for methods). Then integration of services could be specified by scripts in this environment. There are several interesting research projects in this area [49,50]. WSDL has (at least) two important features:

1) An XML specification of properties and methods of the Web Service. This is an XML „equivalent" of IDL in CORBA or Java in RMI.
2) A distinction between the abstract application interface and its realization gotten by binding to particular transport (such as HTTP) and message (such as SOAP) formats.

The result is a model for Services with ports communicating by messages with a general XML specified structure. WSDL allows multiple bindings of a given interface and so supports the "compilation model" described in section 2. As an extreme example, fig. 7 illustrates that one could use WSDL to specify purely local methods and allow implementations that are either distributed or confined within a given address space. This carefully defined model can be integrated with the ideas of DoE's common component architecture (CCA) project [18-20] and this leads to an interesting "Grid programming model for high performance applications". Substantial research is needed into the optimization ("compiling") of Web services but we seem to finally have a promising approach to integrating the best approaches of the high performance computing and commodity software communities. In fig. 8, we show how one can potentially define (composite) Web services with both commodity (SOAP, IIOP, RMI) messaging and high performance parallel MPI standards. For instance the Gateway [46-48] approach to integrating distributed object and high performance modules, builds a wrapper for the latter and implements a very loose coupling between a backend "parallel raw resource" and its proxy in the commodity layer. WSDL allows a tighter integration and this could lead to better commodity interfaces to high performance services. In the same vein as fig. 7, some variant of WSDL could provide a portable framework for linking shared and distributed memory parallel programming models. A WSDL formulation of MPI could involve two types of ports; firstly high performance with "native bindings" where basic data transfer would implemented; however one could use "commodity ports" for the less compute intensive MPI calls and for resetting some of the overall MPI parameters.

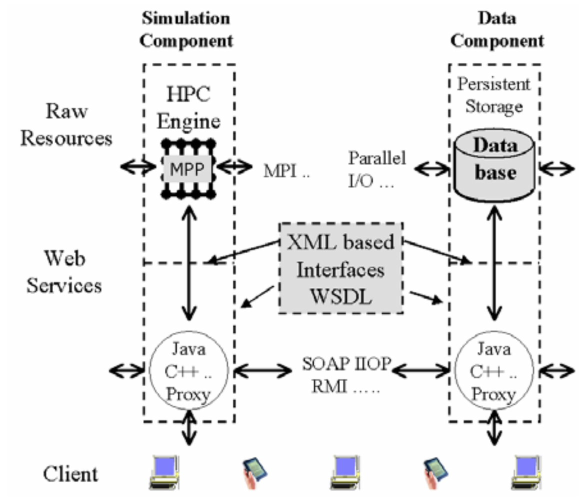


Fig. 8. An approach to integrating high performance and commodity systems

It is very important to note that Web Services are and should be composable. However as long as composed service can be exposed through WSDL it does not matter how they were composed. Therefore all those composition mechanisms are really interchangeable by means of WSDL - this is layer of abstraction that adds up to the robustness of Web Services technology. There are other Web Service technologies -- UDDI [51] and WSIL [52] are two approaches for registering and lookup of such services. This is a critical service but current approach seems incomplete. The matching of syntax of Web Service port interfaces is addressed but not their semantics. Further the field of XML meta-data registering and look up is much broader than Web Services. It seems likely that future versions of UDDI should be built on terms of more general XML Object infrastructure for searching. Possibly developments like the Semantic Web [53,54] will be relevant.

We expect there to be many major efforts to build Web Services throughout a broad range of academic and commercial problem domains. One will define web services in a hierarchical fashion. There will be a set of broadly defined services such as those defined in table 2. These (WSDL) standards will presumably be defined either through a body like W3C or through de facto standards developed by the commodity industry.

Table 2. General Collaboration, Planning and Knowledge Grid Services

People Collaboration	Access Grid - Desktop AV
Resource Collaboration	P2P based document Sharing, WebDAV, News groups, channels, instant messenger, whiteboards, annotation systems
Decision Making Services	Surveys, consensus, group mediation
Knowledge Discovery Service	Data mining, indexes (myGoogle: directory based or unstructured), metadata indexes, digital library services
Workflow Services	Support flow of information (approval) through some process, secure authentication of this flow. Planning and documentation
Authoring Services	Multi-fragment pages, Charts, Multimedia
Universal Access	From PDA/Phone to disabilities

Table 3. Science and Engineering Generic Services

Authoring and Rendering	Storage Rendering and Authoring of Mathematics, scientific whiteboards, nD (n=2,3) support, GIS, Virtual worlds
Multidisciplinary Services	Optimization (NEOS), image processing, netsolve, ninf, Matlab as a collaborative Grid Service
Education Services	Authoring, curriculum specification, assessment and evaluation, self paced learning (from K-12 to Lifelong)

Although critical for e-Science, one will build Science and engineering Services on the top of those in table 2. e-Science itself will define its own generic services as in table 3 and then refine them into areas like research (table 4) and education. Further hierarchical services would be developed on the basis of particular disciplines or perhaps in terms of approaches such as theory and experiment.

Table 4. Science and Engineering Research

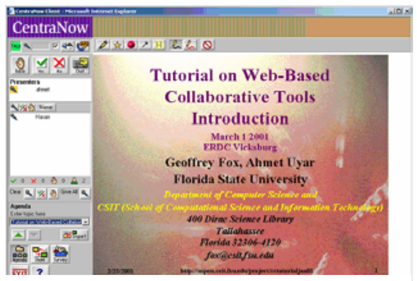
Portal Services	Job control/submission, scheduling, visualization, parameter specification
Legacy Code Support	Wrapping, application Integration, version control, monitoring
Scientific Data Services	High Performance, special formats, virtual data as in Griphyn, scientific journal publication, Geographical Information Systems
Research Support Services	Scientific notebook/whiteboard, brainstorming, seminars, theorem proving
Experiment Support	Virtual Control Rooms (accelerator to satellite), Data analysis, virtual instruments, sensors (Satellites to field work to wireless to video to medical instruments (Telemedicine Grid Service)
Outreach	Multi-cultural customization, multi-level presentations

4. Collaboration

One of the general services introduced in the earlier sections 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 this paper allows one to develop a powerful collaborative model. We originally built a collaborative system TangoInteractive [55,56], which 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 [57,58]. Our new system Garnet has been built from scratch around the model of section 2. In particular Garnet views all collaboration as mediated by the universal event brokering and distribution system described in sections 2.1 and 5.

One of the attractive features of the web and distributed objects is the natural support for asynchronous collaboration. One can post a web page or host a Web Service and then others can access it on their own time. Asynchronous collaboration as enabled by basic web infrastructure, must be supplemented by synchronous or real-time interactions between community members. The field of synchronous collaboration is very active at the moment 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 presentation,), as well shared visualization, maps, or data streaming from sensor.



Tutorial on Web-Based Collaborative Tools Introduction

March 1 2001
ERDC Vicksburg

Geoffrey Fox, Ahmet Uyar
Florida State University
Department of Computer Science and
CSIT (School of Computational Science and Information Technology)
400 Dirac Science Library
Tallahassee
Florida 32306-4120
fox@csit.fsu.edu

Fig. 9. Typical Shared Document System from Centra commercial collaboration system

(3) Audio-video conferencing illustrated by both commercial systems and the recent high-end Access Grid from Argonne [59] shown in fig. 5.

There are several commercial tools that support (1) and (2) – Interwise, Centra, Placeware and WebEx are best known [60-63]. They look to the user similar to the screen in fig. 9 – 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:

Shared Display: The master system brings up an application and the system shares the bitmap defining display window of this application [64]. 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

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 shared display – this or some variant of it (see below) can be the only viable approach if some clients have poor network connectivity. This approach has been developed commercially by Groove Networks using COM objects. It appears interesting to look at this model with Web services as the underlying object model.

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 [38], X3D [42], Java3D or whatever formats ones 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, and 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 clients (these could be URL's, 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 can be done as we described 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 shared display, one can choose to resize images on some machines – this we do for a palmtop device with a low-resolution screen sharing a display from a desktop.

Real-time collaborative systems can be used as a tool in Science in different modes:

(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 (such as earthquakes, unexpected results in a phsics experiment on-line data system, need to respond to 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.

We are developing in Garnet a shared SVG browser in the shared export model. The new SVG standard has some very attractive features [38]. 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 have built a filter that automates the PowerPoint to SVG conversion. The work on SVG can viewed as a special case of a shared W3C DOM [39] environment and it can be extended to sharing any browser (such as XHTML [37]) supporting this document object model.

We mentioned audio-video conferencing earlier in this section where we have used a variety of commercial and research tools with the Access Grid [59] as high-end capability. We are investigating using the Web Service ideas of the previous sections to build a Audio Video Conferencing Web Service with clients using publish-subscribe metaphor to stream audio-video data to the port of the web service that integrates the different systems using the H323 and SIP standards. More generally we expect shared Web Services to be an attractive framework for future work in Garnet.

5. Event Service

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 [26] have rather structured ways of forming communities and controlling them with centralized servers. The P2P approach [23] exemplified by Napster, Gnutella and JXTA [10] uses search techniques with "waves of agents" establishing communities and finding resources As described in section 2, Peer-to-Peer Grids [16] should be built around Web services whose collaborative capabilities support the P2P metaphors.

As one approach to this, we are generalizing the design of the Garnet collaboration system described in the previous section. Currently this uses a central publish-subscribe server for coordinating the collaboration with the current implementation built around the commercial JMS (Java Message Service) [28] 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 our previous TangoInteractive environment.

Originally we realized that Garnet's requirements for a publish/subscribe model were rather different than that for which JMS was developed. Thus we designed some extensions, which we have prototyped in Narada [25,66,67] – a system first described in the PhD thesis of Pallickara [68]. Narada was designed to support 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.
- Narada servers form a distributed network with servers created and terminated as needed to get high performance fault tolerant delivery.

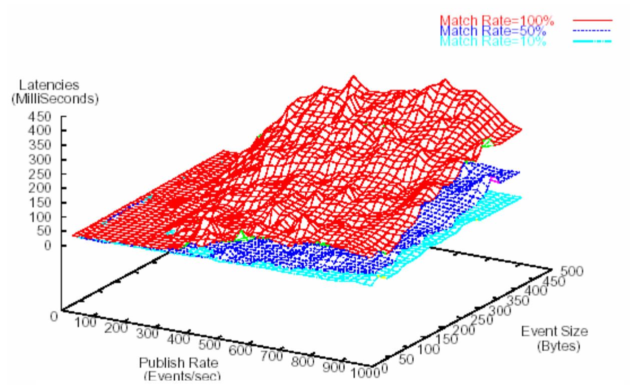


Fig. 10. Message Transit times (labeled latencies) for Narada Event Infrastructure as a function of Event size and publish rate for three different subscription rates

The Narada server network was illustrated in fig. 2 where each cluster of clients instantiates a Narada server. The servers communicate with each other while peer-to-peer methods are used within a client subgroup. Fig. 10 illustrates some results from our initial research where we studied the message delivery time as a function of load. The results are from a system with the event brokering service supported by 22 server processes and with the "measuring" subscriber 10 hops away from publisher. The three matching values correspond to the percentages of subscribing clients to which messages are delivered. We found that the distributed network scaled well with adequate latency (a few milliseconds) unless the system became saturated. On the average, the time per hop between brokers was about 2 milliseconds. We expect this to decrease by a factor of about three in an "optimized production system". Nevertheless, our current pure Java system has adequate performance for several applications. The distributed cluster architecture allows Narada to support large heterogeneous client configurations that scale to arbitrary size. Now we are evolving these ideas to explicitly include both P2P and Web service ideas. We have already extended Narada so that it can function in a mode equivalent to JMS and are currently comparing it with our commercial JMS (from SonicMQ [69]) when used in Garnet and other circumstances. Soon we will integrate Narada with JXTA so that it can interpolate between "central server" (JMS) and distributed P2P (JXTA) models. Web services designed to use the event model of section 2 will then be able to run "seamlessly" with the same API to either classic 3-tier (client-server) or distributed P2P architectures.

Garnet supports PDA access and for this uses a modification – HHMS (Hand Held Message Service) – of the event service. Currently a conventional client subscribes to events of interest to the PDA on the JMS server. This special client acts as an adaptor and exchanges HHMS messages with one or more PDA's. It seems that PDA's and Cell-phones are a especially good example on the opposite end of the scale to clusters and supercomputers. They also require specialized protocols because of performance and size considerations but here because they are slow and constrained devices. In these cases an optimized protocol is not a luxury, it is requirement and a pure XML approach does not perform well enough. However we can still describe what is to be done in XML/WSDL and then translate it into binary (by an adaptor as described above). This is another example of run-time compilation.

We emphasis that it is unlikely that there will be a single event service standard and in this case using XML for events and messages will prove very important in gaining interoperability on the Grid. As long as there are enough similarities between the event systems, the XML specified messages can be automatically transformed by use of an event system adapter that can run as web service and allow for seamless integration of event services as part of middle tier (perhaps this is a filter in Fig. 5).

References

1. The Grid Forum http://www.gridforum.org
2. GridForum Grid Computing Environment working group(http://www.computingportals.org) and survey of existing grid portal projects. http:www.computingportals.org/cbp.html
3. 27 Papers on Grid Computing Environments http://aspen.ucs.indiana.edu/gce/index.html
4. "The Grid: Blueprint for a New Computing Infrastructure", Ian Foster and Carl Kesselman (Eds.), Morgan-Kaufman, 1998. See especially D. Gannon, and A. Grimshaw, "Object-Based Approaches", pp. 205-236, of this book.

5. Globus Grid Project http://www.globus.org
6. GriPhyN Particle Physics Grid Project Site, http://www.griphyn.org/
7. International Virtual Data Grid Laboratory at http://www.ivdgl.org/
8. NEES Earthquake Engineering Grid, http://www.neesgrid.org/
9. SCEC Earthquake Science Grid http://www.scec.org
10. Sun Microsystems JXTA Peer to Peer technology. http://www.jxta.org.
11. "Peer-To-Peer: Harnessing the Benefits of a Disruptive Technology", edited by Andy Oram, O'Reilly Press March 2001.
12. Web Services Description Language (WSDL) 1.1 http://www.w3.org/TR/wsdl.
13. Definition of Web Services and Components http://www.stencilgroup.com/ideas_scope_200106wsdefined.html#whatare
14. Presentation on Web Services by Francesco Curbera of IBM at DoE Components Workshop July 23-25, 2001. Livermore, California. http://www.llnl.gov/CASC/workshops/ components_2001/viewgraphs/FranciscoCurbera.ppt
15. Dennis Gannon, Randall Bramley, Geoffrey Fox, Shava Smallen, Al Rossi, Rachana Ananthakrishnan, Felipe Bertrand, Ken Chiu, Matt Farrellee, Madhu Govindaraju, Sriram Krishnan, Lavanya Ramakrishnan, Yogesh Simmhan, Alek Slominski, Yu Ma, Caroline Olariu, Nicolas Rey-Cenvaz; "Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications" GRID 2001, 2nd International Workshop on Grid Computing, Denver November 12 2001.
16. Geoffrey Fox. and Dennis Gannon, "Computational Grids," *Computing in Science & Engineering*, vol. 3, no. 4, July 2001
17. GrADS Testbed: Grid application development software project. http://hipersoft.cs.rice.edu.
18. R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. Mcinnes, S. Parker, and B. Smolinski, "Toward a Common Component Architecture for High Performance Scientific Computing," *High Performance Distributed Computing Conference*, 1999. See http://z.ca.sandia.gov/~cca-forum.
19. B. A. Allan, R. C. Armstrong, A. P. Wolfe, J. Ray, D. E. Bernholdt and J. A. Kohl, "The CCA Core Specification In a Distributed Memory SPMD Framework," to be published in *Concurrency and Computation : Practice and Experience*
20. Talk by CCA (Common Component architecture) lead Rob Armstrong of Sandia at LLNL July 23-25 2001 meeting on Software Components http://www.llnl.gov/CASC/workshops/components_2001/viewgraphs/RobArmstrong.ppt
21. R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko, M. Yechuri, "A Component Based Services Architecture for Building Distributed Applications," *Proceedings of HPDC*, 2000.
22. S. Krishnan, R. Bramley, D. Gannon, M. Govindaraju, R. Indurkar, A. Slominski, B. Temko, R. Alkire, T. Drews, E. Webb, and J. Alameda, "The XCAT Science Portal," Proceedings of SC2001, 2001.
23. Geoffrey Fox, "Peer-to-Peer Networks," *Computing in Science & Engineering*, vol. 3, no. 3, May2001.
24. Geoffrey Fox, Marlon Pierce et al., *Grid Services for Earthquake Science*, to be published in Concurrency and Computation: Practice and Experience in ACES Special Issue, Spring 2002. http://aspen.ucs.indiana.edu/gemmauisummer2001/resources/gemandit7.doc
25. Geoffrey Fox and Shrideep Pallickara, An Event Service to Support Grid Computational Environments, to be published in Concurrency and Computation: Practice and Experience, Special Issue on Grid Computing Environments.
26. Fox, G. Report on Architecture and Implementation of a Collaborative Computing and Education Portal. http://aspen.csit.fsu.edu/collabtools/updatejuly01/erdcgarnet.pdf. 2001.

27. Ian Foster, Carl Kesselman, Steven Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations http://www.globus.org/research/papers/anatomy.pdf
28. Sun Microsystems. Java Message Service. http://java.sun.com/products/jms
29. W. Johnston, D. Gannon, B. Nitzberg, A. Woo, B. Thigpen, L. Tanner, "Computing and Data Grids for Science and Engineering," Proceedings of SC2000.
30. DoE Fusion Grid at http://www.fusiongrid.org
31. V. Mann and M. Parashar, "Middleware Support for Global Access to Integrated Computational Collaboratories", Proc. of the 10th IEEE symposium on High Performance Distributed Computing (HPDC-10), San Francisco, CA, August 2001.
32. W3C and IETF Standards for Universal Resource Identifiers URI's: http://www.w3.org/Addressing/
33. Resource Description Framework (RDF). http://www.w3.org/TR/REC-rdf-syntax
34. XML based messaging and protocol specifications SOAP. http://www.w3.org/2000/xp/.
35. SIP Session Initiation Protocol standard http://www.ietf.org/html.charters/sip-charter.html
36. SIP for Instant Messaging and Presence Leveraging Extensions (simple) http://www.ietf.org/html.charters/simple-charter.html
37. XHTML Browser Standard in XML http://www.w3.org/TR/xhtml1/
38. W3C Scalable Vector Graphics Standard SVG. http://www.w3.org/Graphics/SVG
39. W3C Document Object Model DOM http://www.w3.org/DOM/
40. Wireless application Protocol (includes WML) http://www.wapforum.org/
41. Voice Extensible Markup Language http://www.voicexml.org
42. X3D 3D Graphics standard http://www.web3d.org/x3d.html
43. Space Link Extension SLE developed by The Consultative Committee for Space Data Systems http://www.ccsds.org
44. Frank Laymann (IBM), Web services Flow Language WSFL, http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
45. Harumi Kuno, Mike Lemon, Alan Karp and Dorothea Beringer, (WSCL – Web services Conversational Language), "Conversations + Interfaces == Business logic" , http://www.hpl.hp.com/techreports/2001/HPL-2001-127.html
46. Gateway Computational Portal, http://www.gatewayportal.org;
47. Marlon. E. Pierce, Choonhan Youn, Geoffrey C. Fox, The Gateway Computational Web Portal http://aspen.ucs.indiana.edu/gce/C543pierce/c543gateway.pdf, to be published in Concurrency and Computation: Practice and Experience in Grid Computing environments Special Issue 2002;
48. Fox, G., Haupt, T., Akarsu E., Kalinichenko, A., Kim, K., Sheethalnath, P., and Youn, C. The Gateway System: Uniform Web Based Access to Remote Resources. 1999. *ACM Java Grande Conference*.
49. IPG Group at NASA Ames, CRADLE Workflow project.
50. Mehrotra, P and colleagues. Arcade Computational Portal. http://www.cs.odu.edu/~ppvm
51. Universal Description, Discovery and Integration Project UDDI, http://www.uddi.org/
52. Peter Brittenham, An Overview of the Web Services Inspection Language (WSIL), http://www-106.ibm.com/developerworks/webservices/library/ws-wsilover/
53. Semantic Web from W3C to describe self organizing Intelligence from enhanced web resources. http://www.w3.org/2001/sw/
54. Berners-Lee, T., Hendler, J., and Lassila, O., "The Semantic Web," Scientific American, May2001.
55. Beca, L., Cheng, G., Fox, G., Jurga, T., Olszewski, K., Podgorny, M., Sokolowski, P., and Walczak, K., "Java Enabling Collaborative Education Health Care and Computing," Concurrency: Practice and Experience, vol. 9, no. 6, pp. 521-533, May1997.
56. Beca, L., Cheng, G., Fox, G., Jurga, T., Olszewski, K., Podgorny, M., and Walczak, K. Web Technologies for Collaborative Visualization and Simulation. 1997. Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing.

57. Fox, G. and Podgorny, M. Real Time Training and Integration of Simulation and Planning using the TANGO Interactive Collaborative System. 1998. International Test and Evaluation Workshop on High Performance Computing.
58. Fox, G., Scavo, T., Bernholdt, D., Markowski, R., McCracken, N., Podgorny, M., Mitra, D., and Malluhi, Q. Synchronous Learning at a Distance: Experiences with TANGO Interactive. 1998. Proceedings of Supercomputing 98 Conference.
59. Argonne National Laboratory. Access Grid. http://www.mcs.anl.gov/fl/accessgrid.
60. Interwise Enterprise Communication Platform http://www.interwise.com/
61. Centra Collaboration Environment. http://www.centra.com.
62. Placeware Collaboration Environment. http://www.placeware.com.
63. WebEx Collaboration Environment. http://www.webex.com.
64. Virtual Network Computing System (VNC). http://www.uk.research.att.com/vnc.
65. openp2p P2P Web Site from O'Reilly http://www.openp2p.com.
66. Geoffrey C. Fox and Shrideep Pallickara , A Scalable Durable Grid Event Service. under review *IEEE Internet Computing*. Special Issue on Usability and the World Wide Web.
67. Geoffrey C. Fox and Shrideep Pallickara , An Approach to High Performance Distributed Web Brokering ACM Ubiquity Volume2 Issue 38. November 2001.
68. Pallickara, S., "A Grid Event Service." PhD Syracuse University, 2001.
69. SonicMQ JMS Server http://www.sonicsoftware.com/