

PROPOSAL

Program Notice: 01-06

Focus Element: Middleware Technology

Proposal Title: IQ-ECho: Interoperability and Quality of Service Across Heterogeneous Hardware/Software Platforms

Institution: Georgia Institute of Technology

PIs: Karsten Schwan, Greg Eisenhauer, Mustaq Ahamad, Calton Pu, and Sudhakar Yalamanchili

Technical Point of Contact:

Karsten Schwan
Georgia Institute of Technology
College of Computing
Atlanta, GA 30032-0280
Phone – 404-894-2589
Fax – 404-385-2295
Email – schwan@cc.gatech.edu

Summary of Costs of Proposed Research:

Year 1 \$215,525
Year 2 \$231,667
Year 3 \$245,120

A. Executive Summary

B. Project Description

B.1 Problem Statement and Solution Approach

Problem statement. This research addresses the problems created by the increased distribution and heterogeneity of the hardware/software platforms on which teams of researchers will conduct future scientific collaborations. The assumptions that drive this work are that on the one hand, it is difficult to constrain the future hardware and software systems such teams will use, especially given the diverse scientific tasks they will be carrying out, and that on the other hand, effective real-time collaboration demands that team members be able to interact with each other and with remote resources as if they were co-located. Two problems arise:

1. Heterogeneous underlying operating systems and software platforms create interoperability problems for the scientific simulations, their input data, and their outputs via which and their team members collaborate.
2. Heterogeneous underlying hardware and networks make it difficult to guarantee the timely transport of data and execution of software required for seamless remote collaboration.

Concerning Problem 1., industry efforts have already resulted in middleware that provides certain levels of interoperation across platforms. Using Corba, IIOP, and SOAP, different operating system and software platforms may interoperate and share data, including Java-based tools. However, these mechanisms are not targeting the high performance domain, as evident, for example, from our own measurements of attainable communication bandwidth for Corba object invocations reported in the last Supercomputing conference[BESW00]. Similarly, the inadequacies of Java's RMI implementations with respect to the latency and bandwidths of parameter transmission are well-known[ZSEC01]. Furthermore, the interoperability mechanisms now provided by industry do not support the degrees of flexibility desired in future collaborations, where end users expect to operate in a dynamic plug-and-play world: it should be easy to switch from one data analysis tool to another; or to in midstream, change the visualizations and views end users employ to gain new insights into the complex scientific processes they are studying; to now 'look over a colleague's shoulders' and next, bring up and study their own views of certain data, regardless of data sizes and often using computationally demanding data analyses. Requirements like these are not met by Corba's relatively static model of interoperation via compile-time stub generation, and even if such stub generation were made dynamic, the diverse nature of today's operating systems and their dynamic linking processes create problems for making processes like these appear seamless across different systems and machine platforms.

Solution approach – the IQ-ECho publish/subscribe middleware. Our approach is to provide end users with rich and efficient new functionality for real-time collaboration, while also leveraging the industry-provided XML standard for component interoperation. Toward this end, we will develop the IQ-ECho middleware, which will implement plug-and-play component interactions via a publish/subscribe model of communication; it will efficiently transport large amounts of data via the logical communication channels it offers; it will in addition, provide the mechanisms for data filtering and transformation required for collaboration across heterogeneous networks, machines, and software platforms. In contrast to industry research on publish/subscribe systems, which tends to focus on a large number of low-end clients (e.g., salespersons placing orders), our emphasis will be on the transport of large scientific data across heterogeneous software and hardware platforms, including Java-based tools. Finally, we will use IQ-ECHO's ability to perform data filtering to address quality of service (QoS) problems like timely data delivery for collaborating end users. This will be done with sample applications that move and display large data across heterogeneous systems and on diverse display platforms.

To summarize, our solution approach is comprised of research that may be divided into three categories:

1. *Scalable interoperation via XML* – our idea is to enable middleware to use the XML standard for the efficient exchange of large-scale data between cooperating components. Industry currently provides ubiquitous support for XML targeted at small data sets. In comparison, our aim is to provide support

for XML-based data exchange for large data, thereby addressing the applications, data generators, analyses, and display tools of importance to the high performance domain.

2. *Quality of Service in middleware* – our goal is to provide middleware-level functionality that permits end users and applications to move data with the timeliness they require, despite dynamic changes in underlying platform resources. The provision of QoS support in middleware is not new, of course. The research we propose differs from previous work in our pursuit of an ‘open systems’ approach for realizing QoS functionality for future high performance computing platforms.
3. *Evaluation with data-intensive applications* – our work will be evaluated with high performance applications available to us via the multi-disciplinary IHPCL research project underway at Georgia Tech. This project seeks to develop Computer Science technologies for high performance applications that currently include combustion modeling, molecular dynamics, materials modeling, and discrete optimization. We will also seek additional collaborations with DOE end users, leveraging our past interactions with DOE researchers via the Globus and Access Grid projects, building upon links already established with DOE researchers at Sandia Labs in Berkeley, and also using links recently established with Oakridge National Labs¹.

B.2 Preliminary Studies

High performance applications. Georgia Tech’s ‘Distributed Computational Laboratories’ project has been addressing distributed and collaborative high performance computing for quite some time, with funding from the National Science Foundation and in part in collaboration with NCSA researchers. As part of these efforts, we have conducted specific research in the HPC domain, and we have created university-wide efforts that support this work. Both are described below. In this context, one specific ongoing project is the Interactive High Performance Computing (IHPC) project [Sch], which is a university-wide project for which an Intel Corporation grant has funded the creation of several high performance cluster computers. These next generation clusters provide a low-cost solution to high performance computing for parallel and distributed scientific applications. The clusters are of significant value to this project both as Unix/Linux-based experimental platforms and as the computational workhorses on which the HPC applications run and their large data are stored that will be used for the remote and collaborative visualizations targeted by this research. Current research conducted in the IHPC project addresses three areas: (1) development of grand challenge applications, (2) technologies for interactive high performance computing, and (3) underlying network support. Grand challenge applications include large-scale optimization problems solved by researchers in Georgia Tech’s School of Industrial and Systems Engineering, molecular dynamics modeling conducted by researchers in Georgia Tech’s School of Physics, and turbulent combustion modeling conducted by the School of Aerospace Engineering. The project name reflects an important goal: that of providing tools through which end users can interact with their applications.

Remote and collaborative visualization of large data. ‘Distributed Computational Laboratories’ is a project initiated with funding from NASA, and then continued with funding from NSF via both a large-scale instrumentation grant and additional funding. Amongst others, these efforts have led to the development of the ECHO middleware[Eisa] leveraged in the project proposed here, since the projects’ purpose was to enable end users to interact with each other and with their computational tools via visual displays, to solve problems collaboratively. Computational tools and collaborators may be local or remote, and tools may be dedicated or shared. In such work, our experiences are that scientists prefer to interact and steer applications from multiple interfaces; from 2D and 3D plots of grid data to graphs summarizing observed behavior. This is one of our reasons for focusing the ECHO middleware described below on the support of plug-and-play functionality for construction of collaborative systems in high performance applications. In the Distributed Laboratories settings with which we have experimented, there are also typically substantial differences in the resources available to users, much like those existing in grid applications: ranging from high end visualization environments like CAVEs to low end environments more suitable for homes like browser-based visualizations. To understand and address these differences, we have developed middleware-level solutions for dynamic load balancing and for filtering data streams as per current user manipulations of their visual displays[IS00], and

¹ The PI is also a member of Georgia Tech’s university-wide effort to create better ties with DOE laboratories like Oakridge.

we have developed the notion of ‘active’ user interfaces to capture some user actions automatically, thus eliminating the need for end users to explicitly state the quality of service and the resolution of data they currently require. Experiments with these solutions in local and wide area settings have demonstrated their ability to provide timely data displays even across international Internet links[IS00].

In general, our research is conducted in computationally diverse hardware environments and with complex, multi-model scientific applications and with multiple client data sinks like data visualization tools. These target hardware/software platforms are controlled through light-weight online steering and monitoring mechanisms, as well as decision mechanisms for controlling and optimizing data flow. The steering and monitoring tools we have developed in past research include a stream-based monitoring and steering framework, Falcon [GESV98, VS95], and its successors that use the ECho middleware and its JECho Java extension[EBS00, BESW00, ZSEC01, Eisa] described below, and an object-based steering framework, termed (J)MOSS [ES98, Eisb], also described in more detail later. Our experiences with such software infrastructures position us well for understanding how to interface IQ-ECho to programming platforms used by DOE researchers, such as CCA and Corba, Globus, MPI, CPA and web portals, etc. Applications used in that research include a parallel atmospheric global transport model coupled with a parallel chemical model. The more complex chemical model enhances the research value of the transport model in that it allows the simulation of reactive species like ozone[PES+98].

The multiple views scientists require and the potential resource limitations they typically experience have driven our research into investigating decision mechanisms for controlling data streams, which include the aforementioned mechanisms for stream control and active interfaces[IS00] and also data stream controls implemented using database cost functions[PS00]. Active interfaces are visual interfaces that offer improved interactions with computational tools by exporting selected state to information providers. For example, an active interface can relay its resource limitations to an information provider, which can then use this information to dynamically tailor the data stream. Extending such functionality, user-specified temporal queries are transformed to further operate on the data stream. Database cost functions which estimate the efficiency of specific operations at the nodes of a query plan dynamically constructed for such queries, are used in the most sophisticated techniques we have developed for dynamically tailoring data streams, as part of the dQUOB layer implemented on top of ECho[PS00].

Object- and event-based middleware. (J)MOSS addresses the strong need to have high performance applications and their user interfaces interact with programs that are available in the commercial domain, written with DCOM or CORBA. One approach to this problem we have explored is to not only ‘wrap’ HPC programs to make them CORBA-compliant, but in addition, to export selected program state and functionality to object systems via CORBA-compliant ‘mirror’ objects. Toward these ends, (J)MOSS mirror objects (1) export relevant application state to potentially remote mirrors that interact with CORBA or CCA applications via IDL interfaces and the IIOP protocols, and (2) they interact with the HPC applications by steering them in response to mirror object invocations. MOSS mirror objects have demonstrated performance in terms of their state updates in response to state changes that is competitive with the performance attained with high performance monitoring systems like Falcon. JMOSS is the Java extensions of MOSS mirrors, which we are currently using to construct a computational ‘workbench’ for mechanical engineers doing parts design[CSR].

The ECho software developed by our group is a publish/subscribe communication infrastructure specifically targeted at the high performance domain. Thus, in contrast to industry efforts that target large numbers of clients each submitting or receiving small amounts of data, ECHO supports large data transfers at speeds exceeding that of high performance frameworks like MPI or PVM. Such performance is achieved due to ECho’s use of efficient ‘wire formats’ for information transport. Underlying ECho and implementing the necessary efficient transport functionality are the PBIO and DataExchange communication layers. Jointly, these communication middleware support the dynamic communications of Distributed Laboratory applications, online program monitoring and steering performed directly or with (J)MOSS, and for collaboration across multiple data visualizations and applications. The principal contributions of the JECho Java middleware are its ability to interface smoothly to ECHO and to support mobility for end users that employ Java-based data generators or viewers.

Program monitoring and steering. The Falcon[GESV98] program steering and monitoring system for parallel machines facilitated the attainment of high performance in parallel programs by offering specific support for program performance monitoring, evaluation, tuning, and steering, and by system integration

support. Information required for steering and performance tuning is easily captured, analyzed, displayed, and then used for program performance improvement. Program monitoring is performed in an identical fashion for both shared and non-shared memory parallel machines, using language-based and graphical user interfaces. Program steering, development, and tuning are performed graphically or via user interfaces that expose the application's parallel structure[VS95]. Falcon was used in the context of large-scale parallel applications being interactively steered by end users, including its deployment at Los Alamos National Labs with parallel stencil codes[VS97]. Its runtime system was implemented on SMP machines based on a configurable and interactive threads package. Falcon's version being used by our group now addresses both the SMP platforms targeted by the initial Falcon system and distributed HPC systems, including cluster machines. In addition, the relatively 'static' ways of attaching Falcon's monitoring and steering facilities in its initial version have been replaced by the highly dynamic attachments provided by the aforementioned ECho and MOSS systems.

The ASAN project. The Active System Area Networks project is developing technologies at the hardware and software levels for extending communication co-processors with application-specific functionality. The use of the term "active" refers to the ability of the network interfaces to perform application-specific as well as system level computations in addition to their traditional role of high performance data transfer. The ASAN project adopts the view that the network infrastructure should be an active computational entity capable of supporting certain classes of computations which would otherwise be performed on the host CPUs. The result is a unique network-wide programming model where computations are dynamically placed within the host CPUs or the network interfaces (NIs) depending upon the quality of service demands and network/CPU resource availability. The projects seeks to demonstrate that such an approach is a better match for data intensive network-based applications and that the advent of low-cost powerful embedded processors and configurable hardware makes such an approach economically viable and desirable. The tangible project goals and results to date include: (1) the demonstration of configurable network interfaces (NIs) comprised of embedded processors on multiple hardware platforms, including last generation NIs supporting 155MB ATM[RSF97] and NIs that have direct access both to disks and to multiple network interfaces (using Intel's I2O boards)[KSR00], (2) experimentation with field programmable gate arrays (FPGAs) to implement data-intensive streaming computations during communication, with particular focus on packet scheduling[WP00], (3) demonstrations of the advantages of using the NIs embedded processors to handle associated meta-information and perform computationally non-intensive operations best performed "close" to the network[RSF97], and (4) the movement of packet scheduling operations to the NIs[KSR00], now followed by the use of the FPGAs to provide practical implementations of computationally demanding QoS scheduling disciplines for real-time communication.

The ASAN project's approach is an experimental one, driven by existing application implementations. Ongoing work in the project is using gigabit Ethernet boards and IXP 1200 routers.

B.3 Research Design and Methods

B.3.1 Solution Approach – Scalable Interoperation via XML

Background. The efficient transport of large-scale scientific data in IQ-ECho will utilize the data encoding and wire formats used in the ECho middleware. Briefly, in ECho, all data is transported as *data events* described by dynamically specified formats, which are essentially type descriptors maintained by the ECho middleware. ECho's ability to transport data events with high performance utilizes such format information to create efficient 'wire' representations (i.e., data layouts 'on the wire') and to encode and decode data events with low overhead. ECho, its wire format and data encodings underlying it are realized by the PBIO library[Eisc, Eis94], all of which are described and evaluated in detail in last year's HPDC and Supercomputing conferences[EBS00, BESW00]. These evaluations show that ECho offers data transport capabilities exceeding even that of common high performance middleware like MPI or PVM, and that its lightweight publish/subscribe mechanism permits end users to create thousands of virtual communication channels at small per-channel costs. Thus, using ECho, it is feasible to export from software tools and simulations the large data sets collaborators and/or analysis tools need to have. Similarly, it is straightforward to dynamically 'attach' and start using certain data sources, like satellites or radars, to analysis codes and simulations, as we have done in previous work in which a large-scale global atmospheric model used

satellite data to periodically adjust its assumptions concerning the vertical transport of certain atmospheric constituents[KSS⁺96, PES⁺98].

Echo's data transport has high performance for two reasons: (1) the 'wire format' into which data is encoded (and from which it is decoded) is designed to minimize data copying at senders and receivers, especially for the heterogeneous machines routinely used when large simulations generate data for display on multiple display machines, and (2) ECHO internally and dynamically optimizes certain routines used during data transport, specifically those that perform data decoding actions. Details of (1) are described in papers on ECHO's P BIO binary format[ED00, EBS00]. Details about (2) appear in our group's recent Supercomputing publication[BESW00].

Efficient movement of XML data – basic idea. End users are typically reluctant to learn about and use new data formats, for reasons of backward compatibility with previous work and/or for fear of wasting time on new formats that may not persist in the future. To attain interoperability, therefore, we must utilize commonly accepted formats and format descriptions. Toward this end, we have chosen XML as the general method by which end users specify the data they wish to exchange across multiple application components, such as between data generators like satellites, the various analysis packages that process it, and the simulations that use it. Similarly, by using XML, it will be easy to interface to software models like the Common Component Architecture (CCA)[AGG⁺99] and to those focused on user interfaces like the Common Portal Architecture (CPA)[GBG⁺00]. Essentially, then, we assume that users 'instrument' their applications by describing relevant inputs and outputs with XML or by exporting from and/or importing to their applications additional data, also specified via XML. At the same time, while using XML as the ubiquitously useful data specification method, we make XML-based 'attachments' available for all software systems mentioned above (e.g., CCA, DCOM) by implementing them at a level of abstraction 'beneath' that of these systems, much like both Corba and MPI finally use TCP/IP as their reliable data transport mechanism. The key idea is to *describe* the data carried by IQ-ECHO channels with XML, but not to actually transport data in XML form. Specifically, from XML-based data descriptions enhanced with semantic information like that used by protocols like SOAP, we will generate at runtime the efficient ECHO-internal wire formats used to transport such data in binary form. Thus, programs using IQ-ECHO efficiently communicate using binary data representations, but the descriptions of such data use the well-known XML standard. When high performance is not important, data can also be exported from or imported into IQ-ECHO (i.e., externalized) in XML form, thereby facilitating interactions with XML-capable software tools and packages.

Efficient movement of XML data – implementation approach. The implementation approach we choose for efficient XML data movement is one that exploits ECHO's internal binary formats, called P BIO. Recall that P BIO formats may be created at runtime. This permits an application, for instance, to declare a new format that contains an array of floats and henceforth, use that format to carry binary data across machines. A receiving machine then uses this format to understand how to unpack the binary data and translate it (when necessary due to machine heterogeneity). In a sense, format definition and use are a dynamic equivalent of what linkers do when compiled programs are being integrated. In contrast to linking, however, at the level of formats, we manipulate relatively 'simpler' entities (i.e., data type definitions), and the implementation of format conversion needs to know only about the machine types being used (i.e., PowerPC vs. I-86 architecture), not about differences in dynamic linking conventions across operating systems. We have found it relatively straightforward to support the relatively small number of machine architectures in current use, currently including I86-based, MIPS, and SPARC machines. For this research, we will also add PowerPC-based architectures to this list.

Performance measurements presented in [BESW00] demonstrate that P BIO and its efficient wire formats outperform existing low-level ubiquitous data representations like XDR and are orders of magnitude more efficient than data representations like XML, both due to differences in 'bulkiness' (i.e., number of bits on the wire) and in the processing overheads experienced on sending and receiving machines. Given dynamic P BIO formats, then, in order to implement XML efficiently, IQ-ECHO will take the following steps whenever an XML description is encountered: (1) it interprets this description as a specification of some data to be sent or received, (2) it translates this specification into an appropriate P BIO format, (3) it sends data in binary form, using P BIO, and finally, (4) data unpacking and translations are done using new stub code also generated dynamically and placed onto whatever machines must perform these actions. Such stub code is created via runtime binary code generation, rudimentary support for which already exists in the ECHO middleware with which IQ-ECHO will be built. When alternative internal formats must be used, such as

XDR, XML descriptions are translated, again dynamically, into such formats rather than into the more efficient PBIO formats ECho currently uses.

Overheads implied by this approach to the transfer of XML-described data do not depend on the sizes of data sets, but instead, they depend on how often the specifications of the data being sent or received change. In fact, the rate of change of such specifications is not high in most of the scientific applications we have experienced. Typically, the new data formats needed are established at ‘connection’ or ‘attachment’ time, whereupon large amounts of data are subsequently moved using these formats.

Scalability of XML-based interoperation. For future large-scale applications, data insertion into and extraction from high performance codes must scale to thousands of machines. This requires technology in addition to lightweight notions of channels and efficient data movement. Specifically, also needed are efficient directory services for resource discovery (e.g., finding suitable channels) and for storing and exchanging intermediate representations (1) of IQ-ECho’s binary data and (2) of their associations with corresponding XML descriptions. Our approach to solving this problem will both use standard directory services like LDAP and also consider novel approaches that combine the request/reply-based protocols of LDAP with cache-based protocols for distributing and maintaining mappings of XML to PBIO and PBIO to XML translations. This is possible in our case because the correctness of system operation does not depend on these mappings always being up-to-date. The system continues to operate correctly (albeit possibly, slowly) as long as these mappings can be discovered in some way.

We will focus our efforts on scalability in cluster computing environments. Specifically, we will assume that cooperating machines are arranged as clusters interconnected via high speed links, with certain clusters or parts of clusters performing tasks like simulation, data capture, data generation, visualization, etc. Heterogeneous network connections primarily exist between clusters and the remote clients who collaborate with clusters and with each other via the high end computations performed by cluster machines. Thus, our studies will be focused on network bandwidth restrictions between clusters and end users, rather than on bandwidth restrictions existing between different components of distributed scientific simulations. The idea is to emulate the network topologies of future laboratories, with computational or storage clusters directly connected to next generation Internet links, which interact with potentially many end users via heterogeneous links, even wireless connections.

Summary and discussion. In summary, IQ-ECho’s implementation of binary data transport for dynamically specified data and channels will employ two key technologies: (1) a well-defined internal representation of dynamically specified data types and (2) runtime binary code generation to create efficient stub code. This approach is general in that it may be applied to software written with Corba, with MPI, or others, and it does not commit developers to expensive investments in new infrastructure, file formats and conversions, etc., because the XML-based APIs they use may result in translations into any number of formats for which translators have been built. For this research, we propose to provide translators that support binary data exchanges via PBIO and XDR and data exchanges via XML.

ECho is a suitable basis for the IQ-ECho work proposed here for two additional reasons. First, ECho already has mechanisms for data filtering and transformation, so that exported data is easily prepared for some new usage, such as being downsampled for visualization on a lower-end client or being transformed for display as an alternate view. These mechanisms are embodied by the novel concept of *deriving* a new communication channel from one that is in current use. Such a derivation implies the creation of a second logical channel in which the data being published is automatically filtered or transformed using the *filter function* specified as part of derivation. Since derivations are performed by data recipients, using this functionality, any recipient can customize the data stream it receives as per its current needs. Such customization will be the basis on which IQ-ECho’s quality of service management support will be built, as well (see Section B.3.2).

The second reason ECho is suitable for the IQ-ECho work proposed here is that its implementation and concepts are orthogonal to systems already in use by high performance end users. That is, for users who already employ systems like Corba, DCOM, Java, MPI, Globus, or the Access Grid, using ECho simply means using another set of libraries in addition to what they currently employ. Since it is commonplace for users to do so anyway when visualizing their data (e.g., to prepare data for visualization with certain tools), learning about and using such libraries does not constitute an onerous or unusual task. Essentially, using IQ-ECho with high performance applications will simply require end users or developers to define and create the data events to be exported from and/or imported to their programs, using XML to describe such data’s types. Once that has been done, their programs can participate in the plug and play interoperation

sought for collaboration in high performance computing, regardless of whether their programs are written with Fortran, implemented using Globus or MPI, or utilize Corba and CCA. Such ease of interoperability is the reason for the ‘I’ in ‘IQ-Echo’.

B.3.2 Solution Approach – Quality of Service in the IQ-Echo Middleware

Problem statement. The second problem we wish to address is that of timely data visualization on heterogeneous clients and across heterogeneous networks. For instance, it should be feasible for an end user operating via a DSL-connected home PC to collaborate with another end user operating on a lab-resident high end graphics machine directly connected to a cluster-based data server. Two problems must be addressed here: (1) visualizations must be available that operate on both types of machines and (2) the network must be able to deliver the data required for such visualizations in a timely fashion. We assume that the first problem will be addressed by end users who will select for their machines one of the many visualization tools already available, such as Viz5D, VizAD, CAVE-based tools, povray, Java-based viewers, etc. For such tools, we offer high performance data transport as long as they are ‘instrumented’ to use XML definitions of the data they import for visualization. It is the second problem, that of dealing with dynamic variations in available network bandwidth, which will be addressed by the proposed research described next.

In order to provide adequate performance over wide area networks, it has become clear that ‘static’ approaches like dedicated networks or reserved network bandwidth are useful for reserving the minimum capacities required by applications, but not for reserving maximum possible capacities. As a result, networking researchers are now developing dynamic methods for providing certain levels of quality for communications, including adaptive methods for network resource management. In order to take advantage of such network-level support, end users must either understand and explicitly manage their applications’ communications, or the middleware they are using must provide such support.

Solution approach – ‘programming’ an ‘open’ platform. Our goal is to substantially enhance the support that middleware provides for runtime resource management. Toward this end, we expect end users to provide application-specific descriptions of their needs and the resource tradeoffs that are admissible, while the IQ-Echo middleware will offer the Quality of Service interfaces and mechanisms – hence the ‘Q’ in ‘IQ-Echo’ – via which (1) such needs and tradeoffs are made known to underlying resource management (at the OS or network levels) and (2) information about platform resources is made known to applications. However, in contrast to previous work on QoS in middleware, such as BBN’s Quo system[ZBS97], our approach takes advantage of the ‘open’ nature of many of the current platforms used in high performance computing, such as Linux-based Beowulf cluster or the programmable network interfaces used in the ASAN project funded by DOE at Georgia Tech (Profs. Yalamanchili and Schwan, PIs). Specifically, rather than asking end users to state their QoS needs with attribute-value pairs as in Quo[ZBS97], for example, or with the payoff or utility functions used in our own prior work[KCS98]), we propose to simply let an application ‘program’ the underlying ‘open’ platform. Such programming has two goals: (1) to help the network or operating system manage resources in a manner meaningful to applications, and (2) to have system levels export to applications resource information that is comprehensible and meaningful.

Figure 1 shows how an open system may be ‘programmed’ at several of its levels, including (1) at the level of IQ-Echo, where data filtering, performance monitoring, and quality management functionality may be freely moved across the machines and address spaces communicating via IQ-Echo’s event channels, (2) at the operating system level, where generalizations of IQ-Echo’s extension functionality will permit end users to place selected quality management functionality into operating system kernels (to be realized for the Linux OS platform), and (3) at the level of network co-processors, where the ASAN project ongoing at Georgia Tech is already developing basic technologies for the runtime placement of application-specific functionality into the firmware (and eventually, into the configurable hardware) of communication co-processors.

Dynamic platform extension. The IQ-Echo middleware will support client-specific (or more generally, receiver-specific) data transport, sampling, and transformation, by permitting applications to create logical communication channels with associated filter functions, using the *derived channel* concept explained in Section B.3.1. These functions may be dynamically defined and run by the data receivers, but receivers can also ‘push’ them into the address spaces of data senders and/or even into underlying operating system kernels or networks. Thus, filter functions *extend* collaborators’ functionality, in a fashion meaningful to their applications. For instance, they inspect or transform application data, perhaps parameterized by the

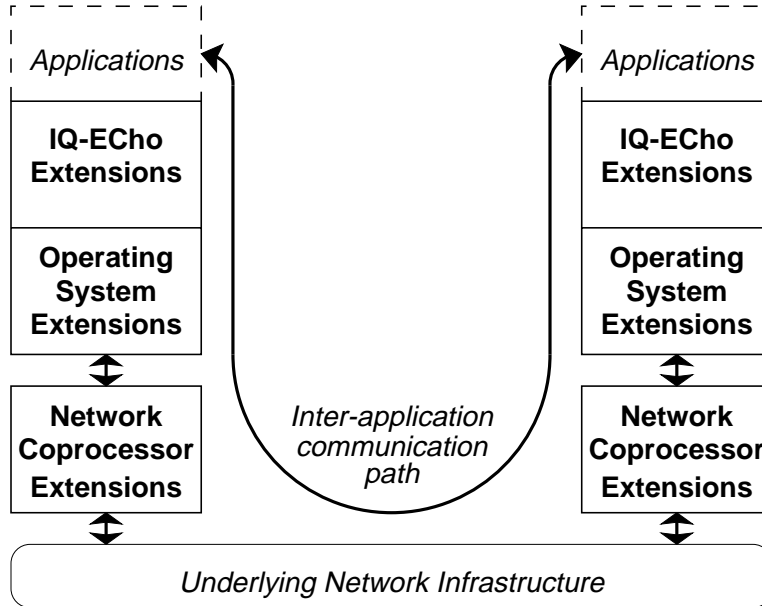


Figure 1: Possible layers of specialization/customization in an application communication path.

degree of data downsampling that should be used, or they select which data should be sent vs. discarded, as per current client needs. Conversely, the network or operating system levels provide to such functions the information they need to operate properly, such as the rate of transport of application-level events. To explain this functionality in more detail, we present the following example.

Consider the real-time visualization of scientific data on heterogeneous display machines and across heterogeneous networks. In remote collaborations involving such visualizations, the network connectivities experienced by end users vary, including dynamic changes in available network bandwidth and communication latency. A specific instance of such collaboration is a remote collaborator connected via the Internet or using a home-based PC who works with a scientist who is using a high end immersive visualization engine located in a lab. Since these clients' data display capabilities differ and since their connectivities vary over time, in order to collaborate effectively, they must be able to dynamically vary not only how much data they receive and display, but also what this data contains. For example, it is not useful for the remote collaborator to receive 'half' of the simulation data the lab-resident scientist wishes him to inspect, but it is useful to receive an explicitly downsampled data set that still retains those data elements of current importance to their collaboration. Such data reduction is meaningfully performed only with an application-supplied filter function, not by some network-resident generic loss function. Similarly, the data transport layer cannot know what value the application places on receiving some of the data in time vs. all of it with some delay. Again, such interpretations can be meaningfully done only with application-specific rules provided by programmers or end users.

For the example explained above, end users might place the following functions into the underlying platform: (1) filter functions placed by clients into the address spaces of data providers, so that only the data currently desired by the clients is actually sent, (2) performance interpretation functions placed into the operating system kernels of the sending and/or receiving machines that collect information about the delays experienced for the receipt of certain amounts of application-level data, from which it is then possible to compute changes in the compression levels or degrees of detail used for the data being transmitted, (3) action functions placed into application-level code or into operating system kernels that enact such changes, preferably in the machines that send the data, so that the network as well as the receiving machines are offloaded, and (4) selected elements of (1), (2), and (3) placed into the network co-processors themselves, such as the lossy packet scheduling functions described in [KSR00] that can both reduce network traffic as well as offload host machines' I/O busses and memory, thus further improving application performance.

Clearly, this list is not complete. For instance, when data is streamed from a server's disks to remote clients, once a streaming action has been initiated by server code, it would be useful to place the filter function not into the address space of such code but instead, into the operating system kernel of the server machine, so that data can be filtered directly on its path from the server's disk to the machine's network interface. Or, the filter function may be placed into the network co-processor itself, especially when data is not resident on a disk but is instead, received by the network co-processor from remote sensors like satellites or radars. In experiments performed with multimedia applications and described in [KSR00], for example, we have found that such host offloading is particularly important when a host's performance should not be perturbed, as when it runs synchronized high performance applications, where even moderate levels of perturbation experienced by any one host may cause substantial slowdowns in the entire application's performance.

'Open platform' programming – specific goals and implementation approach. Our specific goal is to permit the recipients of data to 'move into' the address spaces of remote processes producing data and/or into the underlying platforms used by these processes (1) the data filtering and/or interpretation functions needed in order to send exactly the data that is currently needed by recipients, and (2) the functions that apply rules that determine 'data quality' in ways meaningful to applications, such as 'downsample the image' rather than 'send it late', for instance. Developing the APIs needed to support such functionality is one of the goals of our research. We next briefly outline our approach to developing these interfaces.

The IQ-ECho prototype will offer the previously explained *derived event channel* abstraction, using which data recipients can customize the data they receive. We will also instrument the protocol stacks in the Linux kernel (using existing kernel-level functions like 'netfilter' and by porting portions of TCPDump to the kernel from its current user-level implementation), in order to gather the performance information needed by applications to make reasonable decisions about how to transmit data and which data to transmit. Finally, we will utilize research being conducted as part of ongoing DARPA-, NSF-, and DOE-funded projects (the Infosphere project funded by DARPA, and the ASAN project funded by NSF and DOE) in order to not only instrument but also dynamically customize the open platforms on which IQ-ECho applications run.

Programming the network. We will evaluate the benefits of 'programming' underlying platforms on Linux-based Beowulf clusters that use high performance interconnects based on Gigabit network interfaces. In this context, platform programming that goes beyond extending the address spaces of data senders involves (1) making changes to the operating system's transport layer, including the device drivers used for network interconnects, and (2) making changes to the firmware of network co-processors. The ASAN project's goals are to customize network co-processors, both with respect to their firmware and their hardware, the latter using FPGA technologies. ASAN's basic technology developments are funded by NSF and DOE. The funding received from this project will use those basic technologies to experiment with specific firmware enhancements for the iXP 1200 co-processor boards the ASAN project is now using. Our intent is to understand the degree to which remote visualizations of large data can be assisted by directly programming network interconnects. Our expectation is that especially tasks like data filtering are better done on network co-processors than on host CPUs, because unnecessary loads are removed from host memory and I/O busses and because host CPUs and their deep cache hierarchies are not well-utilized by the computationally light but data-intensive nature of data filtering operations. Preliminary evidence of the suitability of our approach appears in publications at a recent HPDC conference and in last year's ICPP conference [RSF97, KSR00].

Programming the operating system. The second platform component end users should be able to 'program' is the underlying operating system used on machines that provide and/or receive data. Technically, this requires the runtime extension of such operating system kernels with application-level functionality. To attain this goal, IQ-ECho must offer not only its event-based API for sending and receiving data, but it must also 'open up' to the application level the underlying data transports it utilizes. These transports include the TCP/IP or UDP communication protocols layered on top of Media Access protocols (the MAC layer) implemented by device drivers and firmware. 'Programming' these communication layers, then, implies that IQ-ECho must permit an application (1) to receive and interpret with its own 'functions' the performance information that is available from the network and from the OS, and (2) provide its own 'functions' for responding to changes in network performance within the firmware and the operating system kernels. We propose to address these needs by generalizing the notion of derived event channel offered by ECho to one that permits the functions specified as part of derivation to be placed into the operating system and/or communication firmware. The resulting *open channels* offered by IQ-ECho will offer multiple 'anchor points' for functions that extend channel functionality, at minimum including anchors in the sending and/or receiving operating

system kernels and when available, also including anchors in the sending and/or receiving communication firmware. Channel derivation, then, specifies both the functions to be applied to all channel traffic and the anchors at which these functions should be attached in the underlying open platform.

In addition, we will develop globally named and interpreted special formats, termed *attributes*, using which performance information can be transported both across system layers and machines. Using attributes, anchored functions in open channels can transport performance information and/or instructions as to what actions to take in response to new information across the different layers of IQ-ECho's implementation, including the application level, the operating system kernel, and network co-processor firmware.

Summary of proposed work. The outcome of the proposed research on platform programming will be twofold. First, the IQ-ECho middleware to be created as part of this research will offer rich interfaces for end user encodings of desired quality of service and for managing changes in service quality (via application-defined filter functions) for single data streams. Filter functions may be executed by data recipients as well as placed into the address spaces of data producers. The notion of *derived event channel* will be key to the implementation of this functionality, using runtime binary code generation when deploying functions on remote machines. Second, IQ-ECho will facilitate resource management across multiple data streams by permitting quality management functionality to be placed into the underlying OS kernels and communication co-processors, as well, using the notion of *open channels* and *anchor points* for filter functions defined for them. Third, quality and performance information will be represented by special formats maintained by IQ-ECho, termed *attributes*.

B.4 Research Demonstration

The proposed research will be experimental, its principal deliverables being the IQ-ECho event-based middleware that can operate across multiple operating system and hardware platforms, thereby addressing the various machines our end users will employ when accessing their applications interactively and collaboratively. Middleware development will commence in two steps. In the first step, we will utilize representative visualization tools employed by end users, such as Cave5D, Viz5D, and VizAD. We will enhance these tools by developing quality of service formulations and solutions that address their collaborative use across heterogeneous target machines and networks. Given these enhancements, we will then develop techniques for managing middleware and communications so that end users experience the quality of service they desire for ongoing collaborations, despite dynamic variations in the underlying network and computational infrastructure. For instance, for WAN environments, we will develop techniques that react to changes in network availability by adapting the data streams used in applications, focusing on the data streams used in remote sensing and visualization. These techniques will draw on previous and ongoing research in adaptive (real-time) systems, but in contrast to such work, they will explicitly address the large-scale data files and streams prevalent in DOE's high end scientific and engineering applications.

The IQ-ECho middleware and its quality management techniques will be run on hardware infrastructures and applied to applications relevant to DOE's mission, (1) by working with end users at Georgia Tech who are developing high performance applications (often in collaboration with DOE researchers – e.g., research on combustion modeling), and (2) by studying and interfacing with large-scale systems being deployed by DOE, such as the 'Visualization Corridor'. In addition, we will integrate the functionality explored with IQ-ECho into software infrastructures relevant to DOE's mission, such as the CCA, portal architectures, and the grid and access grid software (we are members of the AG interest group and have attended Grid Forum meetings, as well). Our integration approach will be explained in more detail in the full proposal.

B.5 Deliverables and Milestones

The proposed research aims to understand how collaborative applications may use and be supported by adaptive middleware that in turn uses and is supported by the underlying network and operating systems. Toward this end, we will construct sample collaborative applications, develop and implement scalable methods for meeting the quality of service demands of such distributed applications, and map some of these methods into the underlying communication/computation platforms.

The following...

B.6 Linkages and Technology Transfer

The long term impact of this research is to improve the collaborations undertaken by distributed end users, where our efforts focus on real-time collaborations. Specifically, we wish to make distributed scientific applications as accessible to end users as their Excel spreadsheets or text editors, where these accesses are supported across a wide spectrum of platforms, with users conducting a broad set of intellectually and computationally demanding tasks. In a sense, while the World Wide Web is now being used for information browsing and exchange, and to an increasing extent for electronic commerce, we envision applications of the next generation of large-scale computational/communication infrastructures in which difficult tasks are easily and collaboratively carried out across physical distances, with task components including high performance computations, real-time data streams, and human end users. In other words, we view the future world as one in which it is straightforward to construct virtual entities in which interactions and collaborations are conducted. The entities we will construct are sample collaborations that use high performance computations and large data sets.

Our specific plans concerning technology transfer as well as linkages to ongoing DOE efforts are as follows. First, we have been interacting with NCSA researchers like Reed and Stevens over the last few years, in our capacity as one of the NCSA partners and to transfer some of our program steering technologies to ongoing work. As a result, should this effort be funded, we will endeavor to deepen such collaboration to become involved in the ongoing Visualization Corridor project. Of particular interest to us in that context is the issue of 'impedance matching' raised by project researchers, where collaborators may work at different time scales and data exists at multiple levels of spatial resolution, and where data must be represented to permit its transformation from bits to movies or meaningful images. Such transformations are precisely some of the computationally expensive tasks to be carried out by processes connected via IQ-Echo data channels. Similarly, in our recent interactions with groups like Johnson's at Utah, it has become clear that Problem Solving Environments like their UINTAH system can benefit in their distribution from the proposed IQ-Echo system.

While we plan to transfer technology to other DOE or U.S. sites, we will also evaluate and test our ideas by working with other researchers at Georgia Tech who are currently collaborating with us as part of our joint IHPC project. For instance, the combustion modeling applications considered by Prof. Menon routinely generate large data sets that must be analyzed and visualized. We will test our ideas with applications like Menon's, and we have already begun to evaluate the applicability of our work to particle simulations like those created and used by Physics researchers here at Georgia Tech (Matt Wolf, one of the participants in this research, is a research scientist in the IHPC project who also works in GT's particle physics group). Finally, from our past work with atmospheric scientists at Georgia Tech, we have generated sample data sets, filters, and transformations representative of the high performance domain.

Bibliography of literature

- [AGG⁺99] Rob Armstrong, Dennis Gannon, Al Geist, Katarzyna Keahey, Scott Kohn, Lois McInnes, Steve Parker, and Brent Smolinski. Toward a common component architecture for high performance scientific computing. In *Proceedings of the 8th High Performance Distributed Computing (HPDC8)*, 1999. <http://www.acl.lanl.gov/cca>.
- [BESW00] Fabián E. Bustamante, Greg Eisenhauer, Karsten Schwan, and Patrick Widener. Efficient wire formats for high performance computing. In *Proc. of Supercomputing 2000 (SC 2000)*, November 2000.
- [CSR] Yuan Chen, Karsten Schwan, and David Rosen. Java mirrors: Building blocks for interacting with high performance applications. Submitted to ACM 2001 Java Grande Conference.
- [EBS00] Greg Eisenhauer, Fabian Bustamante, and Karsten Schwan. Event services for high performance computing. In *Proceedings of High Performance Distributed Computing (HPDC-2000)*, 2000.
- [ED00] Greg Eisenhauer and Lynn K. Daley. Fast heterogenous binary data interchange. In *Proceedings of the Heterogeneous Computing Workshop (HCW2000)*, May 3-5 2000. <http://www.cc.gatech.edu/systems/papers/Eisenhauer00FHB.pdf>.
- [Eisa] Greg Eisenhauer. ECho web site. <http://www.cc.gatech.edu/projects/ECho>.
- [Eisb] Greg Eisenhauer. Mirror object steering system web site. <http://www.cc.gatech.edu/projects/MOSS>.
- [Eisc] Greg Eisenhauer. PBIO web site. <http://www.cc.gatech.edu/projects/PBIO>.
- [Eis94] Greg Eisenhauer. Portable self-describing binary data streams. Technical Report GIT-CC-94-45, College of Computing, Georgia Institute of Technology, 1994. (*anon. ftp from ftp.cc.gatech.edu*).
- [ES98] Greg Eisenhauer and Karsten Schwan. An object-based infrastructure for program monitoring and steering. In *Proceedings of the 2nd SIGMETRICS Symposium on Parallel and Distributed Tools (SPDT'98)*, August 1998.
- [GBG⁺00] Dennis Gannon, Randall Bramley, Madhusudhan Govindaraju, Nirmal Mukhi, Madhuri Yechuri, and Benjamin Temko. A componentized services architecture for building distributed grid applications. In *Proceedings of the 9th High Performance Distributed Computing (HPDC9)*, 2000.
- [GESV98] Weiming Gu, Greg Eisenhauer, Karsten Schwan, and Jeffrey Vetter. Falcon: On-line monitoring for steering parallel programs. *Concurrency: Practice and Experience*, 10(9):699–736, Aug. 1998.
- [IS00] Carsten Isert and Karsten Schwan. ACDS: Adapting computational data streams for high performance. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, May 2000.
- [KCS98] Robin Kravets, Ken Calvert, and Karsten Schwan. Payoff adaptation of communication for distributed interactive applications. *Journal on High Speed Networking: Special Issue on Multimedia Communications*, 1998.
- [KSR00] Rajamar Krishnamurthy, Karsten Schwan, and Marcel Rosu. A network co-processor-based approach to scalable media streaming in servers. In *Proceedings of the International Conference on Parallel Processing (ICPP00)*, August 2000.
- [KSS⁺96] Thomas Kindler, Karsten Schwan, Dilma Silva, Mary Trauner, and Fred Alyea. A parallel spectral model for atmospheric transport processes. *Concurrency: Practice and Experience*, 8(9):639–666, November 1996.
- [PES⁺98] Beth Plale, Greg Eisenhauer, Karsten Schwan, Jeremy Heiner, Vernard Martin, and Jeffrey Vetter. From interactive applications to distributed laboratories. *IEEE Concurrency*, 6(3), 1998.

- [PS00] Beth Plale and Karsten Schwan. dQUOB: Efficient queries for reducing end-to-end latency in large data streams. In *Proceedings of High Performance Distributed Computing (HPDC2000)*, 2000.
- [RSF97] Marcel-Catalin Rosu, Karsten Schwan, and Richard Fujimoto. Supporting parallel applications on clusters of workstations: The intelligent network interface approach. In *Sixth IEEE International Symposium on High Performance Distributed Computing (HPDC-6)*. IEEE, Aug. 1997.
- [Sch] Karsten Schwan. Description of the Interactive High Performance Computing Laboratory (IHPCL). [http://www.cc.gatech.edu/\[0\]projects/ihpcl](http://www.cc.gatech.edu/[0]projects/ihpcl).
- [VS95] J. Vetter and K. Schwan. Progress: a toolkit for interactive program steering. In *Proc. 1995 Int'l Conf. on Parallel Processing*, pages II//39–42, 1995.
- [VS97] J. Vetter and K. Schwan. High performance computational steering of physical simulations. In *International Parallel Processing Symposium (IPPS)*, Geneva, April 1997. IEEE.
- [WP00] Richard West and Christian Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. In *Proceedings of the Real-Time Systems Symposium*, 2000.
- [ZBS97] J.A. Zinky, D. E. Bakken, and R. Schantz. Architecture Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, January 1997. Also see <http://www.dist-systems.bbn.com/tech/QuO>.
- [ZSEC01] Dong Zhou, Karsten Schwan, Greg Eisenhauer, and Yuan Chen. Supporting distributed high performance application with java event channels. In *Proceedings of the 2001 International Parallel and Distributed Proceeding Symposium (IPDPS)*, April 2001.