A Networked Instructional Instrumentation Facility

Stephan C. Werges, David L. Naylor
Department of Electrical Engineering and Computer Science,
University of Illinois at Chicago

The Networked Instructional Instrumentation Facility (NIIF) is a prototype system to allow multi-user access to a library of sophisticated test equipment for measurements on a library of devices, in real time via the World Wide Web (WWW). This prototype system is the first realization of our primary goal to make measurement equipment available in real time to a broad range of users via the WWW. Since the NIIF is available on the WWW for public use, there are many special design issues that must be considered. These issues include: 1) The efficient scheduling and queuing of measurement jobs that may take one or more instruments to perform; 2) Security precautions to prevent malicious use of the facility; 3) Making the NIIF fault tolerant and robust; and 4) Designing the client user interface to be intuitive and easy to use for a wide range of users. To meet these challenges we have developed an object oriented client/server architecture and a Measurement Applications Programming Interface (MeAPI) for the NIIF.

The basic architecture for the NIIF server consists of four integral parts: instrument objects; the control objects; measurement objects; and job objects. The instrument object is the lowest level object in the server, and allows the server to communicate with the instruments that are physically or logically attached to the server. These instrument objects conform to the MeAPI, which allows them to plug in to the control object. The control object is responsible for the validation, queuing and scheduling of measurement jobs, control of the matrix switch that connects test equipment to test devices, communication with the client, and control of the instruments. The control object uses a measurement object to define what instruments and parameters are needed for a given measurement. Lastly, the job object contains the client information that the control object needs to perform a measurement and returns the data to the client.

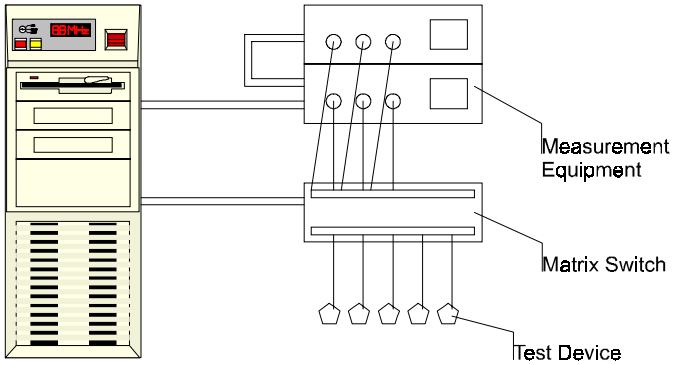
The NIIF's client architecture consists of two parts: the send object, and the receive object. The send object allows the user to choose a type of measurement, a set of specific measurement parameters, and a device to test. This object collects the measurement parameters, and sends the data to the server. The receive object listens for incoming data from the server and allows the data from the server to be displayed in a variety of different ways. This paper will describe how the object oriented client/server architecture described above and MeAPI is implemented to meet the special design requirements of the NIIF.

INTRODUCTION

The explosion of the Internet and especially the World Wide Web (WWW) has revolutionized how the public obtains and deals with information. The WWW has given the public a simple way to access vast repositories of on-line information, but the paradigm for this access has been predominantly a static client/server model. Over the past two years, this static client/server

model has changed to an active one with the advent of Java and ActiveX. Both of these tools have allowed the creation of dynamic web pages [1], which enable users to dynamically interact with the information they receive or the information source itself. This has enabled the WWW to become a medium for true client/server computing, and led among other things to the creation of the Networked Instructional Information Facility (NIIF).

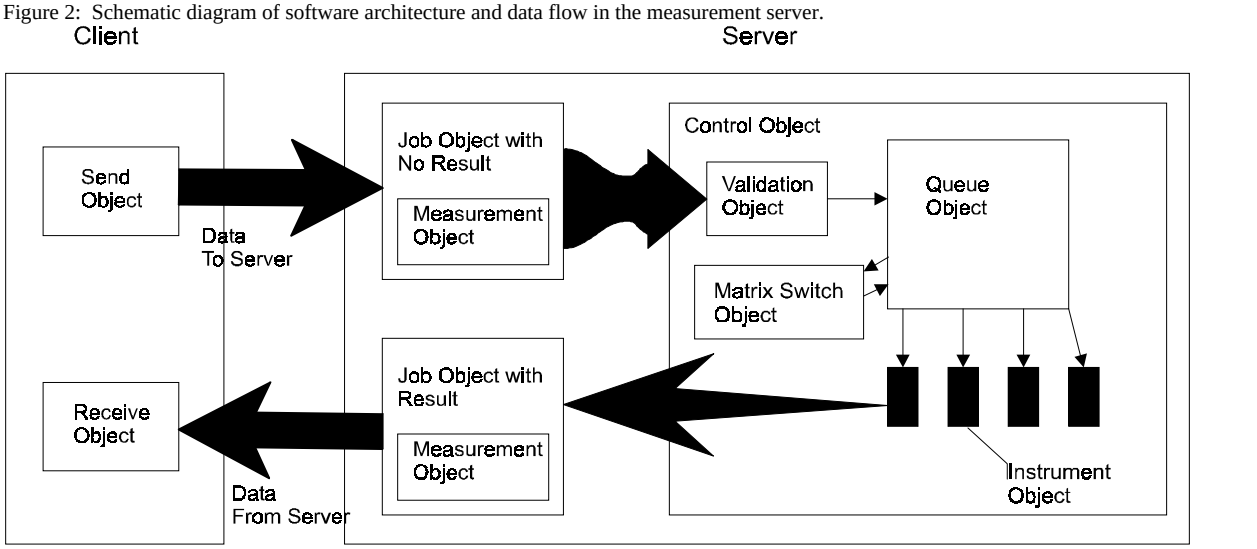
Figure 1: Schematic diagram of the NIIF's measurement server.



Typically, scientific equipment placed on the Internet has been highly specialized and available only to a limited number of users. Examples of these specialized equipment facilities are: the Remote Experimental Environment (REE) [2]; the University of California at Santa Barbara Remote Access Astronomy Project [3]; the Upper Atmosphere Research Collaboratory (UARC) [4]; and the Collaboratory for Environmental Molecular Sciences (EMSL).[5] By contrast the NIIF is a prototype system to allow multi-user access to a library of sophisticated test equipment for measurements on a library of devices, in real time via the WWW as shown in figure 1. This prototype system is the first realization of our primary goal to make measurement equipment available in real time to a broad range of users via the WWW. The NIIF was designed using an object-oriented client/server architecture and a Measurement Application Programming Interface (MeAPI) that we have developed.

The motivation to create the NIIF was to make a "collaboratory," [6] where students can perform electronics labs at their own convenience and from any location. Thus, a student need not be at the university to perform a lab, and that makes the NIIF an excellent remote learning tool. In a broader sense, the measurement server designed for the NIIF can be used for time sharing prohibitively expensive equipment between research institutions or in corporations. It allows equipment to be used around the clock with no geographic barriers.[7] The NIIF's measurement server provides one approach to making remote measurement available on the Internet.

Figure 2: Schematic diagram of software architecture and data flow in the measurement server.



This paper will discuss the general object-oriented architecture and MeAPI of the NIIF, and how the architecture deals with the special design issues of a public facility on the Internet. These special design issues include: 1) The efficient scheduling and queuing of measurement jobs that may take one or more instruments to perform; 2) Security precautions to prevent malicious use of the facility; 3) Making the NIIF fault tolerant and robust; and 4) Designing the client user interface to be intuitive and easy to use for a wide range of users.

GENERAL ARCHITECTURE

The NIIF server was built as a service for Microsoft Windows NT Server 4.0 using Microsoft Visual C++ and the Win32 API. The architecture of the NIIF server consists of four basic types of objects: an instrument object, a job object, a measurement object, and the control object. The instrument object is the root object within the server. It contains its own input/output (IO) data parsers and code for all measurements that can be performed. The instrument object then communicates with the control object via the MeAPI. The MeAPI specifies that the instrument object must provide a unique id for each of its measurement functions, provide a unique id for the instrument itself, and must provide a common interface to call each of the measurement functions. The advantage of using the MeAPI is that it abstracts an instrument object allowing the control object to handle many different types of instruments and measurements.

The control object handles the validation, queuing, scheduling, and execution of jobs, and the control of the matrix switch. The control object itself contains the validation object, queue object, matrix object, and instrument objects. The validation object checks the parameters of a measurement job against values in a Microsoft SQL 6.5 database to make sure that the measurement will not harm the device under test (DUT) and the test equipment. It also fetches the locations of the DUT and the test equipment on the matrix switch. The queue object queues jobs and commits the jobs to the instruments. The matrix object controls the matrix switch that connects the proper test equipment to the proper DUT.

The job object contains all relevant data about the client, what measurement is to be performed via a measurement object, the appropriate measurement parameters, any error messages encountered during the measurement, and the data to be sent to the client when

available. It also handles the sending of the data back to the client or any error messages. The job object also can be stored to disk for archival or error checking purposes.

The measurement object describes a particular measurement. It handles the decomposition of a job object in to pieces that must be passed to each instrument. This object interfaces with the validation object to make sure that all required parameters for a measurement are present.

The client has been built with Java using Microsoft J++. The client architecture consists of the send client and receive client. The send client takes the user input, parses it, sends it to the server, and spawns a receive client. The receive client receives any error messages sent from the server, receives the output from the server, and provides facilities to manipulate the data from the measurement.

The cycle of a measurement request is shown in figure 2. The incoming data is received by the server and a job object is spawned. The job object is then decomposed by the appropriate measurement object and passed to the validation object. Here the job is checked to make sure that all the parameters are acceptable, and the locations of the test equipment and DUT in the matrix switch are obtained. If all parameters are proper then the job is passed to the queue. If not the job is rejected and the user is informed. Once the job has reached its turn in the queue the matrix object is called to electrically connect the test equipment to the DUT and the measurement is performed. The results of the measurement or error messages are returned to the job object after the measurement has been performed. At this point, the job is serialized and the results sent back to the client. Once the data has been received, the user may manipulate it with the tools provided by the send client.

DESIGN ISSUES

Queuing and Scheduling

The NIIF has been designed to be used in a multi-user multi-instrument multi-DUT environment, and this has created a queuing and scheduling problem. The problem is as follows: Consider a measurement that requires three instruments to be available simultaneously (figure 3a). Each particular job for an instrument is placed at the back of the appropriate instrument queue and the queues are allowed to empty. If a measurement job for one of the instruments should reach the front of its queue, it will stall the queue waiting for the other instruments measurement jobs to reach the front of their queues. This is a very inefficient use of the NIIF’s resources. To solve this problem a novel priority queue has been developed.

Figure 3: a) The test job is placed in the queues of each of three required instruments. The queues then are allowed to empty using a first in first out algorithm. Notice that one of the jobs in the queues reaches the front of a queue before the others. This stalls that queue and prevents job processing on that instrument. b) The test job is placed in the instrument queues, but the instrument queues communicate to keep the job at the same level in the queues. As new jobs arrive in the queues they can jump over the test job, because they can be processed before the test job reaches the front of the queues.

figure 3a

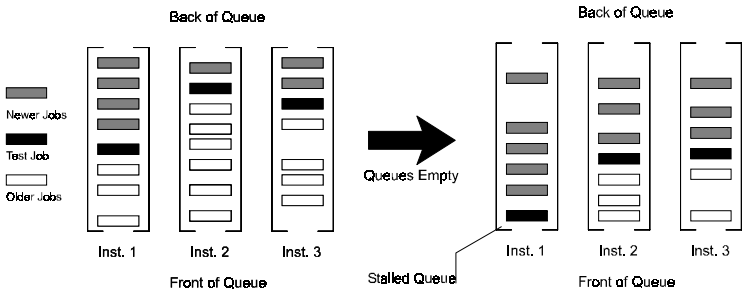
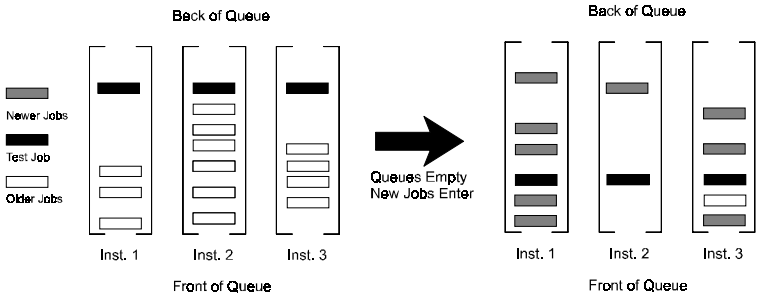


figure 3b



The NIIF's main queue object consists of intercommunicating priority queues for each instrument and DUT. These smaller priority queues communicate to keep multi-instrument jobs level between the queues and allow other jobs to be processed in the queues. This queuing algorithm differs from standard queuing algorithms, because simultaneous access to some or all resources that the queues control is required.[8] Consider the same three instrument measurement job with the new queuing algorithm (figure 3b). The jobs are inserted in to the different instrument queues such that they arrive at the front of the queue at the same time. In addition, other jobs are allowed to jump over this job if they can be completed before this job reaches the front of the queue. This simple queuing and scheduling algorithm should allow the NIIF to process jobs more efficiently.

Security

The NIIF is a public facility on the Internet and vulnerable to attacks and malicious use. Two major precautions have been put in place to safeguard the NIIF. The first precaution is that the user is never aware of which TCP port the NIIF is monitoring for measurement requests. This is done to prevent anyone from directly accessing the NIIF's measurement server without using an official client. When the user first downloads a Java client it is passed the utility port number of the NIIF. From this utility port the client downloads the TCP port number that the NIIF uses to receive measurement requests.

The second precaution lies in the validation object of the server. Here the job is checked to make sure that: 1) the measurement job is originating from a valid client through a security code and, 2) that all parameters in the measurement job do not exceed the tolerances specified for the instruments and DUT in the SQL database. With the validation object no unsafe

measurements are allowed to be performed. Thus protecting the measurement equipment and DUTs from misuse.

Fault Tolerance

The NIIF is fault tolerant in both the server and the client. The server's fault tolerance is rooted in the instrument object. If an instrument should have a problem, it communicates the problem to the control object. Depending on the situation the control object will halt the queue and wait until all instruments have finished their measurements. It will then try to reset the instrument and restart the queue. If this fails, the instrument and all measurements requiring that instrument are disabled, the jobs that required this instrument are purged from the queue, the affected users are informed of the problem, the queue is restarted, and all other measurements are completed.

The fault tolerance of the NIIF clients lies in the fact that the send and receive clients are two different objects. It was found during development that web browsers could be very unstable when running a programmer controlled multithreaded applet. By dividing the send and receive client into two different objects the web browser takes care of the threading, providing a much more stable applet. Moreover, if either send or receive client should fail it will not crash the whole applet because they are tied together by threads. This allows the NIIF to have a more robust client.

Client Design

Since the NIIF is available to anyone using a web browser on the Internet, the client interface must be intuitive and easy to use. The ease of use is achieved by writing a send client for each type of measurement. The client architecture allows any send object to be plugged in to the client. Thus a specific user interface can be designed for a given measurement type, and used with a generic or specifically designed receive client. This feature also allows the NIIF to be extended to perform highly specialized measurements.

CONCLUSION

The Networked Instructional Instrumentation Facility is to our knowledge the first public multi-user remote measurement facility that allows users to perform measurements with a library of test equipment on a library of test devices. Its novel object-oriented architecture has been designed to meet the many special issues that face on-line Internet facilities. These special issues include: 1) The efficient scheduling and queuing of measurement jobs that may take one or more instruments to perform; 2) Security precautions to prevent malicious use of the facility; 3) Making the NIIF fault tolerant and robust; and 4) Designing the client user interface to be intuitive and easy to use for a wide range of users. The NIIF is the first facility of this kind and is one possible model for remote test and measurement on the Internet.

References

[1] David Hazarika, "Developing and Deploying Interactive Applications on the Internet," Microsoft Developer Network Library-Visual Studio 97.

[2] Remote Experimental Environment, http://www.es.net/hypertext/collaborations/REE/REE.html.

[3] University of California at Santa Barbara Remote Access Astronomy Project, http://www.deepspace.ucsb.edu/rot.htm.

[4] Upper Atmosphere Research Collaboratory, http://www.si.umich.edu/UARC/.

[5] Collaboratory for Environmental Molecular Sciences, http://www.emsl.pnl.gov:2080/docs/collab/CollabHome.html.

[6] R. Kouzes, J. Myers, and W. Wulf, "Collaboratories: Doing Science on the Internet," *Computer*, Vol. 29, No. 8, August 1996, pp. 40-46.

[7] National Research Council, *National Collaboratories: Applying Information Technologies for Scientific Research*, National Academy Press, Washington, D.C., 1993.

[8] L. Kleinrock, *Queuing Systems Volume 2: Computer Applications*, Wiley, New York, 1976.

Biographies

STEPHAN C. WERGES (scwerges@uic.edu) is a designer and the lead programmer for the Networked Instructional Instrumentation Facility, as well as Network Administrator for the Microfabrication Applications Laboratory at the University of Illinois, Chicago. His research interests include distributed systems, user interface design, and virtual reality. Werges received his BS in mathematics from the University of Illinois, Chicago and is currently pursuing his MS in electrical engineering and computer science at the University of Illinois, Chicago.

DAVID L. NAYLOR (naylor@uic.edu) is an Associate Professor in the Electrical Engineering & Computer Science Department at the University of Illinois at Chicago. He is Co-Director of the Microfabrication Applications Laboratory and the MicroFluidics Center at UIC and has research interests in: Microfabricated fluidic and optical components; Microelectromechanical systems (MEMS); and Internetworked instrumentation. He received his BA in physics from Oxford University and Ph.D. in electrical engineering from the University of Southern California.