A Virtual Instrument Bus Using Network Programming

D.J. Rawnsley, D.M. Hummels, B.E. Segee
University of Maine, Orono Maine [1]

Abstract

This paper provides an overview of a virtual instrument bus created at the University of Maine Orono. Software to support automated tests has become difficult to maintain as the number of test boards and test instruments grows. A variety of test instruments such as logic analyzers, signal generators, and data caches connect and communicate to workstations using a General Purpose Interface Bus (GPIB).

This paper describes two software packages. The first is a "virtual instrument bus" that makes a large number of GPIB buses on separate networked computers appear to be on a single bus. The second is an object-oriented instrument library. The Library is designed to support a variety of instruments using a common framework in an easily maintained software package.

The virtual instrument library is developed using remote procedure calls (RPC). All workstations supporting an instrument bus run a background program called a Bus Server that handles bus communications and provides an interface to the computer network. The Bus Server can be programmed to handle any kind of bus, not just the GPIB.

Communication to the various Bus Servers is handled by the Virtual Bus Library. This interface makes the physical configuration of the instrument buses transparent to the software developer. The library supports a small set of routines modeled after the IEEE 488.2. It also provides searching functions for locating specific instruments on the computer network, and maintains a list of all machines that have instrument buses connected to them.

The virtual bus software provides easy code reuse for quick program generation used for automated testing, at the same time making all instruments appear to be located on one single bus. This software will greatly facilitate the future development of complex experiments requiring multiple bus instrument coordination.

1 Introduction

This paper presents the development and implementation of instrument control software for use in a networked computer environment. The project was motivated by ongoing research

'This work has been supported in part by the ARPA HBT/ADC program under a contract administered by the Office of Naval Research Grant N000149311007 and the DEPSCoR program through the Army Research Office Grant DAAH04-94-G-0387

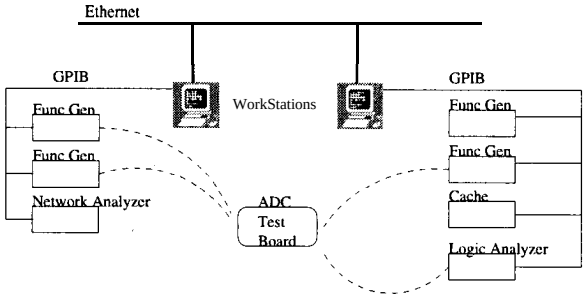

Figure 1: Ideal Lab Configuration

in the Communication Laboratory at the University of Maine. The Communications Lab, among other things, analyzes Analog to Digital (A/D) converter output to provide a means of compensation for the error introduced by the device. Software to support automated tests for data acquisition from A/D test boards has become difficult to maintain as the number of test boards and test instruments grows. A variety of test instruments such as logic analyzers, signal generators, and data caches connect and communicate to workstations using a General Purpose Interface Bus (GPIB). Software to control test instruments that are physically located on separate workstations within the lab as illustrated in Figure 1 are extremely time consuming or impossible to configure. Moving instruments from one workstation to another required reconfiguring software and recompiling an extensive software package.

The software maintenance and network support issues encountered on the Communications Lab are typical of those encountered when instruments are controlled over a computer network. While users have become accustomed to distributed network resources (shared file systems, transparent access to printers, etc) instrument control software has not supported the capabilities of most networks. For a networked computing environment, instrument control software should support the following features:

- Instruments should be portable to any machine on the local network without recompiling test software.
- Test software should not be platform dependent. Tests should operate correctly regardless of the platform that the test is run from.
- Development of test software should not be platform dependent. Once the network instrument control libraries are compiled for a particular architecture, the test software should be supported for any machine using that architecture.
- The software interface should be consistent regardless of the physical instrument bus interface.
- A common software interface should be provided for instruments with common functionality. For example, all function generators should respond to a common set of amplitude/frequency configuration commands.

2 Existing Software

Existing software used in the Lab for data acquisition and controlling instruments is written in the C programming language. The physical addresses of the test instruments and the names of the machine hosts that they are connected to are hard-coded into the software. In order to move instruments from one machine to another, or to change its address, the existing software package has to be recompiled for the changes to take effect.

To access instrument software for a specific instrument, the user has to be logged-on to the workstation to which the instrument is physically connected. Automated tests involving multiple instruments connected to different physical buses cannot be supported. In order to incorporate an instrument on a different bus than the one the test is running on, the cabling would have to be physically changed to the new bus. The address of the instrument would have to be set so that it did not conflict with any other instrument on the new bus location, and the software would have to be recompiled to reflect these changes. Setting up for such changes is time consuming and problematic.

With the expansion of our facility to include new high speed instruments for A/D testing, the current setup is not an efficient use of equipment.

3 The Approach

Two software packages are described which together address these issues. The first is a "virtual instrument bus" which makes a large number of physical buses on a computer network look like a single bus. The Virtual Instrument Library is designed to support the computer network communications making the computer networking transparent to program developers.

The second software package is an object-oriented instrument library which is specific to instruments within the communications lab. The Communications Lab Library is designed to support a variety of instruments using a common framework in an easily maintained software package. The communication between the libraries is illustrated in Figure 2.

3.1 Virtual Instrument Library

The virtual instrument library is developed using remote procedure calls (RPC). RPC is a mechanism for building a distributed system of programs that handle all communications between the physical buses, the workstations, and the network. All workstations supporting an instrument bus run a background program that handles all bus communications (like GPIB) and provides an interface to the computer network. These programs are shown in Figure 2 as Bus Servers. The Bus Server can be programmed to communicate with instruments using any kind of bus (not just a GPIB).

Communication to the various Bus Servers is handled by the Virtual Bus Library. This

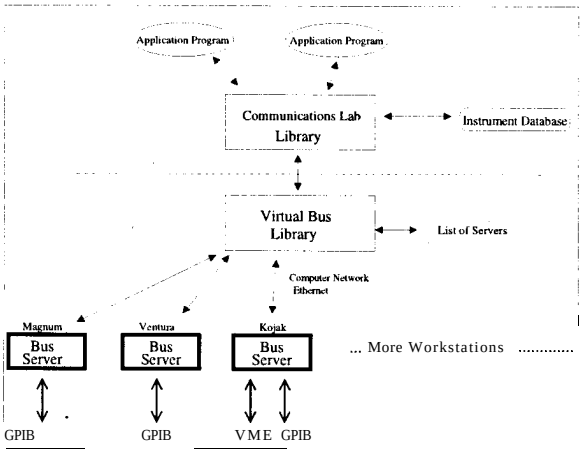

Figure 2: Virtual Bus Block Diagram.

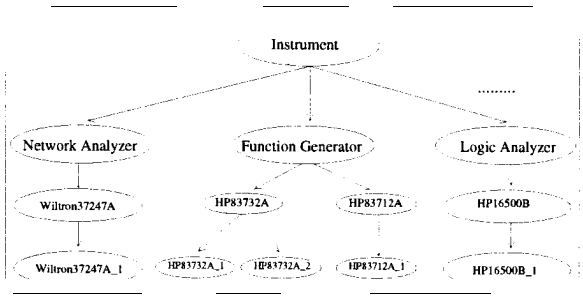

Figure 3: Communications Lab Library Software Structure.

interface makes the physical configuration of the instrument buses transparent to the data acquisition software developer. The library supports a small set of routines modeled after the IEEE 488.2 GPIB standard. It also provides searching functions for locating specific instruments on the computer network, and maintains a list of all machines that have instrument buses connected to them. This library handles ail communications to the RPC Bus Servers, and is the interface to the computer network for the Communications Lab library.

3.2 **Communications Lab Library**

The Communications Lab library is created using an object-oriented architecture design. It is designed to represent the functionality of the test instruments and provide simplified software reuse and changeability in a modularized fashion. The structure of the library is illustrated in Figure 3.

Object-oriented programming is a method of extending abstract data types to allow for type/subtype relationships among data types. In C++ this is accomplished with inheritance. Instead of re-implementing shared characteristics, an object can inherit the functionality of the class it was derived from. The C++ class mechanism allows programmers to define their

own data type.

The Communications Lab library uses C++ inheritance extensively. Each level in Figure 3 inherits the functionality of the level above it. All test equipment are bus instruments that have common Instrument functions. A piece of test equipment, such as a Function Generator, has its own common functions to complement the common Instrument functions. For example, every Function Generator supports a common software interface for controlling the frequency or amplitude of the generator. A specific generator is a Hewlett Packard 83732a which has a variety of functions that are provided which are specific to that model.

The Communications Lab library maintains a bus configuration database shown in Figure 2 which is automatically updated if a change to an instrument's address or location is detected. When one of these instruments, like an HP83732a, is used in a program, the library first checks the current location and address in the instrument database. This is done to make sure the program is talking to the correct instrument. If not, search functions of the Virtual Bus library are run to locate the test instrument and update the instrument database of the new test instrument location and address.

The virtual bus software provides easy code reuse for quick program generation used for automated testing, at the same time making all instruments appear to be located on one single bus. This software will greatly facilitate the future development of complex experiments requiring multiple bus instrument coordination.

4 Virtual Bus Software Architecture

This section gives a quick overview of the software architecture including the names and purposes of the major executables and routines. Figure 4 shows the client-server architecture used for the Virtual Bus Software.

4.1 Client Side

The Application Programs are the client side of the architecture. All Application programs use the two software libraries, the Communications Lab Library and the Virtual Bus Library, to create client executables. The Communications Lab Library is an object-oriented library that models types of instruments, and communicates with the Instrument Database Server for up to date information on instrument locations. The Virtual Bus Library is the interface to the network communications. This interface is used by the Communications Lab Library to provide reusable objects for Application Programs.

4.1.1 *Virtual Bus Interface*

The interface for the virtual bus abstracts away the ideas of network programming from the Communications Lab Library and Application Programs. All interface functions establish

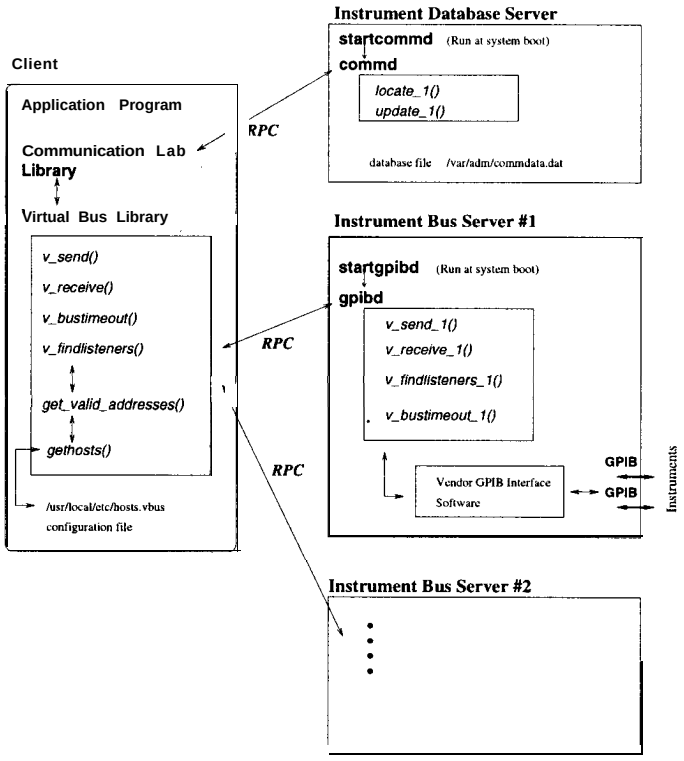

Figure 4: Software Architecture

connections with the specified servers and handle network communications. When completed, each routine disconnects from the server. Each routine provides an interface that makes it appear that the routine is running locally. When in fact, it maybe executing on a different workstation. The following is brief review of each interface routine.

1. v_send(): Send commands or data to a specified instrument.
2. v_receive(): Receive data from a specified instrument.
3. v_bustimeout(): Set the timeout value for the physical bus. The timeout value is the approximate minimum length of time that I/O functions can take before a timeout occurs.
4. v_findlisteners(): Poll the bus to find the number of listeners.

There are two helper functions that are used by v_f indlisteners 0 :

1. get_valid_addresses(): Build a list of addresses for the v_f indlisteners 0 function.
2. gethosts(): Get a list of host workstations and possible bus addresses from a configuration file.

4.2 Server Side

Two different types of servers are used for the virtual bus: the Instrument Server and the Instrument Database Server.

4.2.1 Instrument Server

The Instrument Server, also called the Bus Server, is run as a background process which is configured by the startgpibd executable. When this process is started during workstation boot-up, it is replaced with the gpibd executable. gpibd is the server that handles all client requests to communicate with the instrument bus. When a connection is made, a specific service is performed by calling one of the following routines:

1. **v-send-lo:** Send commands or data to a specified instrument physically connected to the same workstation this procedure is executed on.
2. **v_receive_l():** Receive data from a specified instrument physically connected to the same workstation this procedure is executed on.
3. **v_bustimeout_l():** Sets the timeout value for the local bus.
4. **v_findlisteners_l():** Poll the local bus to find the number of listeners.

Each one of these routines calls vender specific GPIB interface software to communicate on the bus.

4.2.2 *Instrument Database Server*

The Instrument Database Server is run as a background process which is configured by the startcommd executable. When this process is started during workstation boot-up, it is replaced with the commd executable. The commd server handles all client requests for information about the location of a specific instrument. This server provides two database services:

1. **locate_l():** Given an instrument identifier, return the last known location of that instrument.
2. **update-lo:** Update the location of an instrument in the database to the current location.

There maybe as many Instrument Servers as there are workstations that have external buses, but only one Instrument Database Server is needed to maintain instrument locations.

5 Conclusions

The Virtual Instrument Bus software has proven to be an excellent software package for data acquisition across a local network. The convenience of running and creating data acquisition software from any workstation on the network makes development easy for the user. The ease of moving instrument locations and changing instrument addresses for specific test setups without recompiling software allows for easy configuration of automated tests. Once an instrument has had its location or address changed the software will update the database so that no searching will take place the next time the software is run. The Virtual Instrument Bus software is a powerful tool for providing development of complex experiments requiring multiple bus instrument coordination.

Biographies of the Authors

Daryl Rawnsley received a B.S. in Computer Engineering at the University of Maine in 1995. He is a candidate for the Master of Science degree in Computer Engineering from the University of Maine in May 1997. His current research interests include communications and computer networks. He is a member of IEEE, Eta Kappa Nu, and his interests include firefighting and sports.

Don Hummels is an Associate Professor of Electrical and Computer Engineering at the University of Maine. He has been with the University of Maine since 1988. He obtained his B.S. degree from Kansas State University, and his M.S. and Ph.D. degrees from Purdue University, all in Electrical Engineering. His research interests include nonlinear signal processing, and characterization and compensation of nonlinear components used in communications receivers.

Dr. Bruce Segee received a PhD in Engineering from the University of New Hampshire in 1992. He has been an assistant professor of electrical and computer engineering at the University of Maine since that time. At the University of Maine he heads the Instrumentation Research Laboratory, an organization dedicated to research and teaching involving instrumentation and automation. Work in the lab includes the use of PC's, PLC's, and embedded controllers for instrumentation, automation, and networking. Work also includes the use of fuzzy logic and artificial neural networks.