INSTRUMENTATION

TESSELLA INC

Issue V1.R1.M1
April 2003

1. WHAT IS INSTRUMENTATION CONTROL?

Introduction

Instrumentation control is a large topic that covers many areas. Broadly speaking it is the control and automation of devices through software, usually via an external standalone computer. Traditionally, instruments have focused on doing a limited number of tasks, such as measuring pre-defined parameters such as temperature, voltages etc., or as motor controller platforms as used by robots and sample preparation equipment.

Another common characteristic of instruments is that they have a command language, and the remote computer communicates through a set of messages or text commands. Typical examples of laboratory instruments are those employed in the field of analytic investigation such as X-ray, infrared and mass spectroscopy and chromatography.

Instrument control systems are found across a wide range of industry sectors, from automotive (robotic control and non-destructive testing), and industrial analysis (X-ray fluorescence and Infra-red), through to medical (monitoring and drug delivery), the petrochemical industries (chemical analysis) and pharmaceuticals (high throughput screening).

In the last few years the pharmaceutical and bioscience industry has seen a huge explosion in the need for instrument control systems as the automation of gene sequencing, DNA synthesis and drug discovery has taken off. The recent completion of such high profile projects as the mapping of the human genome have relied heavily on instrument control and automation.

Background

Historically instruments were built using their own proprietary command protocols and bespoke hardware such as micro-controllers and firmware. These instruments could not easily fit into the integrated information systems demanded by modern industries. After the introduction of commodity PC hardware in the 80's and the progressive development of a fast i/o bus architecture, scientists and engineers could connect their instruments directly to their computer and take advantage of the benefits that transferring data directly gave them.

- ❑ Fast and accurate data transfer.
- ❑ Ability to share data throughout the organisation.
- ❑ Archive data and off line processing.
- ❑ Data mining and analytic processing of historical data.

It was not until the development of modern software tools, such as integrated development environments, high-speed communication protocols, high-level languages and specialist toolkits, that the scientists and engineers could really take advantage of the PC architecture for instrument control. These improvements in software tools provided the flexible and low cost instrument development platform that we have today.

2. EXAMPLES OF SCIENTIFIC INSTRUMENTS

Below are a few example Industrial Analysis applications for scientific instruments.

X-ray fluorescence

Energy dispersive X-ray fluorescence (ED-XRF) is a technique of chemical analysis. The interaction of X-rays with an object causes secondary (fluorescent) X-rays to be generated. Each element present in the object produces X-rays with different energies. These X-rays can be detected and displayed as a spectrum of intensity against energy: the positions of the peaks identify which elements are present and the peak heights identify how much of each element is present.

Scanning Electron microscopes

Electron Microscopes are scientific instruments that use a beam of highly energetic electrons to examine objects on a very fine scale. This examination can yield information on the topography, morphology, composition and crystal structure of the sample.

Infra-red and near infra-red spectroscopy

Infra-red spectroscopy is used to examine the chemical structure of organic molecules. The absorption and transmittance of the infra-red radiation through the substance of interest can provide information on the chemical structure and bonds of the molecules present.

3. KEY ISSUES

It is important to examine some of the key issues that will influence the design of the software for a scientific instrument. This list is not intended to be exhaustive, but will serve to give an example of some of the aspects that need further investigation.

- **Computer literacy of the users.** Are the users likely to be sophisticated, for example, laboratory technicians with extensive experience using instrument software, or are they operators in an industrial environment who may have little or no experience of laboratory systems and computer software? The makeup of the target user group will have a big influence on the level of sophistication of the software user interface.
- **Environment.** Will the instrument be located in a clean and well-maintained laboratory or in a harsh industrial environment? This will have an impact on any peripherals that will be attached to the instrument. If the environment is susceptible to electromagnetic interference then this will have a bearing on the PC-instrument interface options.
- **Computer Architecture.** Will the instrument contain an integrated SBC (Single Board Computer) or will it be connected to a standalone Intel compatible PC? Does the instrument need a separate embedded processor for real-time signal processing etc? If so, which type of embedded processor will be used?
- **Operating systems.** Can a desktop operating system for the software development such as Win2000 or XP be used? Does the instrument need a more stable multi-user OS such as Linux? What are the licensing issues we need to address? Is a real-time OS needed for any time critical tasks? Does the operating system support emulators for the available hardware?
- **Data Acquisition**. Is there a need to develop any specialist data capture interface boards and their corresponding device drivers, or can off the shelf PCI acquisition boards and drivers be purchased?
- **Micro-controllers.** What development tools are available and how much do they cost? What sort of documentation is available (reference manuals, books)? Which manufacturer to choose from; Motorola, AMD, Intel etc? What are the memory options; EEPROM, Flash etc? What are the voltage requirements? Does the instrument need Peripheral Interface Controllers (PICs) for motor control etc? How will the micro-controllers communicate, for example, should they use the Serial Peripheral Interface (SPI) protocol?
- **Data Input.** Which data input methods need to be considered in the design of the software? For example does the instrument need to be keyboard driven only, or is the use of a mouse and keyboard acceptable? Is the working environment too inhospitable for a keyboard and mouse and therefore should the instrument have a touch screen display instead? Does the instrument need a bar-code scanner to read sample labels? Does the instrument need to be controlled remotely over a web interface?
- **Software Interface.** How will the main user interface software be developed? Low level system programming languages such C or C++? Which libraries should be used for the graphical user interface, for example, MFC under windows or Qt

under Linux? Should the user software be split between a low level COM layer and a high level GUI developed in a RAD language such as Visual Basic? Can the software be developed using high-level instrument control toolkits such as LabVIEW etc?

- ❑ **Data Storage**. Will the instrument store data locally on internal disks or will data be transferred to a central data repository? If data is stored locally how will data backup procedures be implemented? In what format will data be transferred from the instrument, is the format an open standard or proprietary?
- ❑ **Connectivity.** Which types of network connections does the instrument need, Ethernet to connect to a corporate network or over a web interface? Is traditional serial (RS-232) acceptable or should the instrument support newer protocols such as USB 2.0, Bluetooth or FireWire for external devices etc? Does the instrument need an internal modem to allow remote diagnostics and fault detection by the manufacturer?
- ❑ **Integration.** Does the instrument interface to external systems such as LIMS etc?
- ❑ **External Devices.** Does the instrument need to connect to external devices such as CDROM/CD-RW/Plotters/Printer/Modem etc?
- ❑ **Regulations.** Does the instrument need to conform to any external regulatory requirements for storage of electronic records?
- ❑ **Workflow.** Does the software need to provide a workflow interface to lead the user through a controlled sequence of tasks?

4. POSSIBLE IMPLEMENTATIONS

Selecting your instrument-computer architecture

Essentially there are two approaches for controlling instruments with computers. One is to use an externally connected IBM compatible PC. The other is to have the computer built into the instrument chassis. The integrated PC will still need to talk to the instrument via an i/o bus, but external cables will not be required.

External Computer

The advantage of using an externally connected computer to control the instrument is that no special hardware is required and software development is simplified by allowing the software development machine to be connected directly to the instrument.

If the instrument is designed to be portable then one option is to use a hand held computer such as an iPAQ or similar device attached to the instrument. These devices are lightweight and portable and have the advantage that they support roughly the same software development tools as desktop PCs. For example, there are embedded versions of Microsoft Visual Basic and Visual C++. The disadvantage is that they have limited processing power and storage capacity- battery lifetime may also be an issue that needs to be addressed.

Integrated Computer

Computers designed for internal use within instruments are known as Single Board Computers (SBCs). They are IBM compatible PCs built on specially designed motherboards which contain all the built in devices, such as processor, memory, graphics cards, Ethernet and hard disk drives. SBCs are not usually fitted with the very latest processors, but they are usually designed with component longevity and low power consumption in mind. SBCs are also less susceptible to component obsolescence, which can be an issue if the instrument and software is to be supported over a number of years. Another advantage of using SBCs is that the software developer has full control over the hardware and software running on the computer.

Software Development Models

The following examples of development models are provided for the Intel x86 compatible computers, which is the most common platform for instrument control. The software options have been split into two categories - programming languages and toolkits.

Programming languages

Low-level system languages

If efficiency and speed are going to be primary considerations for the system software, or if time critical control is required, then system languages such as C or C++ will probably be the first choice. These languages provide the programmer very fine control of all aspects of the software, for example, memory allocation and threading support. Libraries are available for all the major communication protocols such as Ethernet and RS-232. The disadvantage of these languages is that they are prone to common programming errors such as memory leaks and buffer overruns.

For development under Microsoft Windows the preferred choice is Visual Studio. Microsoft also provides a number of libraries to provide graphical interfaces such as MFC and ATL. C++ COM components are particularly common when developing software for instruments because they can be used as a hardware abstraction layer. This separates the developer from any peculiaritiesof the hardware and makes the software easy to port to new instruments.

Another approach is to develop the software under the Linux operating system. The choices of low-level languages here are also C/C++ but the graphics toolkits are different. For C++ the toolkit available is Qt and for C development Gtk.

Interpreted and high level languages

Interpreted languages offer some substantial benefits over system programming languages, in terms of ease of development. The common languages available on Windows are Java, Visual Basic and C#. These languages come with extensive libraries, which improve development productivity.

Interpreted languages under Linux include Perl, Python, Tcl/Tk and Java. The graphics toolkit Gtk contains bindings for both Perl and Python.

Instrument Toolkits

A popular method of instrument control is through software toolkits. These are graphical programming languages that let the user build the application from components. They allow complex control and automation applications to be developed quickly and with little programming knowledge. The mostly commonly used instrument toolkit is National Instrument's LabVIEW software.

LabVIEW

LabVIEW, short for *Laboratory Virtual Instrument Engineering Workbench*, is a programming environment in which you create programs with graphics; in this regard it differs from traditional programming languages like C, C++ or Java, in which you program commands in a text editor. LabVIEW uses the graphical programming language, 'G', to create programs in a pictorial form called a block diagram. LabVIEW programs are called *Virtual Instruments* (*VI*s) because their appearance and operation imitate actual instruments.

LabVIEW also contains libraries of code for data acquisition (DAQ), General Purpose Interface Bus (GPIB), serial instrument control, data analysis, data presentation, data storage and communication over the Internet. The libraries also contain functions for signal generation, signal processing, filters, windows, statistics, regression, linear algebra, and array arithmetic.

Ready to Run Solutions

So called ready to run solutions are software applications that provide a virtual instrument interface such as an oscilloscope or a digital multi-meter. Once the software has been setup to read from a PCI data acquisition card then the instrument software is complete. The advantage of these programmes is that they require no programming, the disadvantage is that they are inflexible and are really only useful for data logging.

Interacting with instruments

As well as the traditional methods of data input, such as keyboards and mice, it is worth examining alternative methods of interacting with instruments.

Touch Screen Displays

A touch screen display panel is a piece of glass which sits over the monitor's CRT (Cathode Ray Tube). Touch screen software processes all data coming in from the screen when it is touched. Software drivers convert the touch data (x y position) into mouse events. As far as application software is concerned, all it is seeing is a mouse clicks coming from the computer.

Touch screen displays are extremely useful when the data inputs into the instrument are relatively small, but they can become clumsy and slow when inputting large amounts of text. They are most often used when the operating environment for the instrument is too harsh to allow sustained use of a keyboard and mouse. In an industrial environment computer mice and keyboards can become damaged. If the user interface can be kept relatively simple, for example, no menus or complex interface controls etc., then a good case can be made for touch screen displays.

Bar Code Scanners

Instruments working in high throughput laboratories are often required to operate unattended. One of the issues that arises is keeping track of samples throughout the analysis process. One common method is to provide the instrument with automatic bar code readers. This frees the operator from constantly having to enter sample data into the instrument. It also means that instruments that deal with sample preparation, analysis and result processing can quickly identify samples. Examples of instruments that use bar code readers would be high throughput screening robots as used in biotech laboratories.

Communicating with instruments

The following is a brief introduction to some of the options for connecting computers to scientific instruments. It is not intended to be definitive, but to give the reader an idea of the choices available.

GPIB (IEEE 488)

Traditionally the communication interface of choice for instrument control has been the IEEE 488, General-Purpose Interface Bus (GPIB). GPIB is a digital, 8-bit parallel communications interface with data transfer rates of up to 8 Mbytes/s. The bus provides one system controller for up to 14 instruments and cabling is limited to less than 20 m. Users can overcome both of these limitations by using GPIB expanders and extenders. GPIB cables and connectors can be industrially shielded for use in noisy environments.

Serial Port - RS-232

RS-232 is a common interface primarily because all personal computers have a least one serial port connector. RS-232 has a maximum baud-rate of 20,000 bits per second. The most commonly used baud-rate values are 300, 1200, 2400, 9600 and 19,200 baud. Data is transmitted using a serial (i.e. one bit at a time) full-duplex (i.e. simultaneous send and receive) at a rate governed by the cable capacitance. The maximum cable length specified by the standard is 17m. RS-232 is slowly being replaced by USB ports on newer computers.

Ethernet - TCP/IP

Instrument control applications that make use of Ethernet as the communication protocol have definite advantages over the two previous protocols mentioned (IEEE 488 and RS-232). Instrument operators can make use of existing Ethernet networks in their companies and laboratories. Integrating the instrument directly into the corporate network allows the sharing of data and results more easily. The data transfer rate is also improved, as most common Ethernet networks today are 10BaseT or 100BaseTX, transferring data at 10Mb/s or 100Mb/s respectively.

The main advantage of Ethernet is that it allows the instrument to be controlled remotely from the users desktop. Most of the common communication protocols have cable lengths limited to a few metres. Ethernet allows instruments to be controlled from different corporate sites and even different countries. Web interfaces can be developed to instruments, which make use of Ethernet connections to allow users constant access to instruments and data. However, for sensitive data, users must take additional security measures to ensure data integrity and privacy.

5. MICRO-CONTROLLERS AND FIRMWARE

If the instrument is designed to carry out a task such as signal processing, then it will be likely that an embedded micro-controller will be employed to carry out the necessary processing, while leaving the controlling computer free to carry out other tasks. If the instrument is fitted with a new or highly customized detection system or is carrying out some highly specialized data acquisition task, then it will not be possible to source the required data acquisition hardware from a third party. If this is the case then the micro-controller code will also need to be developed in conjunction with the high-level user facing software.

This document is not intended to give a detailed account of micro-controller software development, but it will give a brief overview of some of the issues that need to be addressed if the fully custom route is to be taken.

Embedded Controllers

Embedded controllers or microprocessors come in a wide variety of specification and functionality, for example, there are 4, 8, 16 and 32bit microprocessors available. Choosing the correct architecture will depend on several factors such as the processing power required and particular feature set needed such as I/O, interrupt handling, analog to digital conversion or digital signal processing etc. An important consideration when choosing a microprocessor is to select one which comes with a full software development kit and extensive documentation.

Programming Micro-controllers

At the lowest level, micro-controllers only understand machine language programmes, assembly language is the human readable form of a micro-controllers machine language. Assembly/machine language programmes are small and fast but unfortunately they are difficult to write, error prone and the development process is slower than with other development methods.

Micro-controllers can also contain interpreters resident in memory. Interpreters are high-level language translators that convert each language instruction into machine language and execute it in turn. The two most popular interpreters for micro-controllers are BASIC and FORTH. Java is also becoming a popular interpreted language for embedded systems. Interpreters allow for high development productivity because the program can be built interactively. A downside of interpreted languages is that performance is not as good as using assembly language programs.

A good compromise between low-level assembly and high-level interpreted languages is achieved using compilers. The compiler will translate the language instructions into machine language, which can then be downloaded into the micro-controller. The language of choice for micro-controller programming is C. Although a high level language, it also gives the developer access to the underlying machine. There are C compilers available for most micro-controllers. C is widely used, available, supported and produces efficient code.

Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGA), are a new type of computer architecture. In contrast to traditional computer processors that have a fixed internal structure, FPGA contain large amounts of programmable logic gates and memory. These can be used for performing logical and arithmetical operations on data. Programming the FPGA will determine the internal structure of the processor. Anadvantage of FPGA is that they offer the flexibility of computer software with the very high speed of dedicated hardware. They also avoid the cost, time delay and risk in having custom microprocessors manufactured. The disadvantage is that they require expensive software development tools and specialist development knowledge to program.

The main use of FPGA is implementing highly CPU intensive tasks such as digital signal processing, image processing, pattern recognition and encryption.

6. DATA PROCESSING

Data Storage

The data storage requirements for instruments vary drastically - at one end of the scale we have large scientific experiments such as those at CERN that can produce terabytes of data and at the other, small instruments such as digital multi-meters, which usually have no data storage requirements at all.

Data storage effectively comes down to three options;

- ❑ **No Storage** this is the usual option for instruments such as oscilloscopes, multi-meters etc. These instruments are effectively data acquisition viewers; the data has no real value after it has been collected. For these types of measurement devices a printout containing results is usually sufficient.
- ❑ **Local Storage** all data collected by the instrument is stored locally either directly in the computer file system or in a database file. This is a straightforward system to implement and works very well if the amount of data generated is not too big or if the instrument is not connected to a network. But also has disadvantages - how will database administration be carried out? How do we increase the storage capacity? How do we backup data?

- ❑ **Central Storage** data collected by the instrument stored in a central location such as a relational database on a corporate database server. This allows data generated by multiple instruments to be combined and provides the benefits of professional database administration such as scalability, regular backups etc. Data stored centrally can easily be post-processed and the results made available to a wide selection of users.

Reprocessing
When deciding on the data storage requirements for the instrument it is quite common to store only the raw, uncalibrated data from the instrument and disregard any post-processed or calibrated results. This can result in substantial reductions in data storage requirements. Reprocessing the raw data against the calibration can then be used to generate previous results.

7. FURTHER CONSIDERATIONS

Regulatory Compliance
Companies that are involved in the development of pharmaceutical drugs for the United States must seek clearance approval from the US Food and Drug Administration. For the case of electronic records then the company must comply with 21 CFR Part 11. For clearance to 21 CFR Part 11, the regulations that software has to comply with are quite clear and rigorous in terms of data protection and record retention.

For an analytical instrument, any information that is captured and stored on a PC is considered data or metadata (metadata is just data about data, which helps put the real data into context) and is therefore considered an electronic record under the conditions of 21 CFR Part 11. For example, parameters that are captured by a HPLC system (High Performance Liquid Chromatography), such as flow rate and sample number, are considered metadata and should be stored along with the electronic records.

Laboratory Data management
With the automation of the laboratory and the “industrialisation” of laboratory techniques comes the need for efficient management of the large amounts of data that is generated. Traditionally, instruments would store the data captured on the controlling computer, but in large organizations this approach is no longer feasible because the data becomes fragmented and difficult to control. The more managed approach is to allow each instrument to export data to a central data store using a single formatting model. Processes that use this data are not then dependent on the source or type of instrument that the data came

from. Users throughout an organization can then perform quality control and offline analysis on collected data.

This area of data management is referred to as Laboratory Informatics and is responsible for turning raw data into information that will benefit the organisation. One aspect of this data management is integrating the data acquisition instruments into LIMS (Laboratory Information Management Systems). The LIMS will give its users control over issues such as sample tracking, laboratory processes and workflow, data access, data storage and regulatory compliance.

8. CASE STUDIES

The following case studies demonstrate three separate approaches to instrument control development. They cover the entire range available from a complete bespoke software solution at all levels, to a modular approach using a high-level instrument toolkit.

Custom Hardware and Software

Customer

The customer is a manufacturer of scientific and industrial analytical instrumentation and a specialist in X-ray fluorescence instrumentation. The market for the X-ray instrument is in the analysis of oil samples for the petrochemical industry.

Business Problem

The customer wishes to develop a brand new product that will make use of recent developments in its X-ray detector technology. To gain maximum performance out of the instrument the customer is willing to design a custom signal processing solution and custom desktop software suite. The instrument must also be able to transfer large quantities of data to the PC for data visualisation in real-time.

Solution

Because extremely high-speed data processing is required when dealing with X-ray events (microsecond interval events must be distinguished), the multi-channel analyser, which converts X-ray signals into a continuous spectrum, is implemented using a Field Programmable Gate Array (FPGA). The FPGA and instrument motors are controlled using an embedded Motorola processor running a real-time embedded operating system.

Communication to the instrument motor PICs (Peripheral Interface Controllers) is through the SPI (Serial Peripheral Interface) protocol.

The embedded processor is responsible for queuing command messages from the host PC and sending back data for analysis. The instrument communicates via a command language that is transmitted via the RS-232 serial interface. An ATL/COM server on the host SBC receives the results of instrument analysis and makes the information available to the high level software, which is responsible for presenting the results.

Results and Benefits

By developing an in-house solution the customer was able to control the entire data flow from detection to presentation. The quality of the signal could be monitored and controlled. By utilising the FPGA and micro-controller for the data acquisition and analogue/digital conversion, the specification for the host PC was kept modest, therefore lowering the total cost of the instrument.

Software Toolkit For Third Party Hardware

Customer

A large US Instrumentation company, which as part of its product line manufactures infrared spectrometers. One market for these instruments is in the analysis of engine lubricating oil where levels of contamination and degradation are used to determine engine condition. The company adds value to their instruments by supplying a variety of complementary software systems for different industry sectors.

Business Problem

The company wanted a new 32bit user interface to an existing IR instrument system. The software was to be built around a sophisticated workflow model of analysis methods, samples and results. The instrument was to include a third party robotic auto-sampler as part of the system to provide automated sample analysis. The software was to be used by computer literate lab technicians familiar with tools such as Microsoft Office. The system had to be completed to a fixed price and within a very demanding time frame.

Solution

The software was developed using MS Visual C++ with the MFC toolkit and made use of third party software libraries to provide a familiar look and feel, similar to that of the MS

Outlook application. The instrument was controlled using COM objects supplied by the customer designed for their own instrument hardware.

Programming the auto-sampler was carried out using device drivers and documented high level APIs provided by the manufacture. The instrumentation company also provided an emulator to allow the development to proceed even without access to the relevant hardware.

Results and Benefits

The system was delivered on time and to the cost specified by the customer. The use of third party tools was instrumental in getting the system completed on time. By using the high level tools, such as the API's and emulator, provided by the hardware manufacturers, the development time was drastically reduced and the deadline met.

Instrumentation Toolkit

Customer

A company that produces individual beamline components and complete beamline systems to harness, manipulate and utilise x-ray radiation generated by synchrotron light sources.

Business Problem

The company wished to develop a new bespoke beamline system for the National University of Singapore. The instrument would also form part of their standard product catalogue. Their Beamline system was to utilise the continuous, high energy x-rays from a synchrotron to etch templates of minute devices (1mm feature size) into x-ray sensitive materials. The system combined a traditional beamline, to constrain and modulate the x-ray beam, with a scanner end-station, to create a state of the art micro-machining tool. This software was to remotely control, monitor and fully automate the operation of the complete Beamline system.

Solution

The control software was developed using LabVIEW, interfacing with many different hardware device types, including an advanced motor control system to spatially control the position of the x-ray sensitive target along four directional axes. The LabVIEW control software was a complex system, interfacing in real-time with more than 90 individual devices including precise, powerful motors to move the complete modulate the x-ray beam, with a scanner end-station, to create a state of the art micro-machining tool. This software was to remotely control, monitor and fully automate the operation of the complete Beamline system.

Solution

The control software was developed using LabVIEW, interfacing with many different hardware device types, including an advanced motor control system to spatially control the position of the x-ray sensitive target along four directional axes. The LabVIEW control software was a complex system, interfacing in real-time with more than 90 individual devices including precise, powerful motors to move the complete scanner. A combination of hardware interface types were used including GPIB and RS-232.

Results and Benefits

The control software provided both a graphical interface, to allow the remote control and monitoring of all aspects of the beamline, and full automation of the micro-machining process. The use of an instrument toolkit such as LabVIEW provided all the hardware drivers and programming API's need to control and monitor the instrument while allowing the developers to concentrate on the functionality and graphical user interface of the system.

1. WHAT THE FUTURE HOLDS FOR INSTRUMENT CONTROL

Introduction

The following is a brief introduction to the future direction of instrument control software from the point of view of data capture/management and the communication protocols for data transfer to and from the instrument.

Data management

Instrument Markup Language

One of the outstanding problems with integrating collections of instruments into a LIMS or data management system is the proprietary format the data is generated in. Vast quantities of data are being generated in real-time and stored in databases in various incompatible formats. Modern laboratories are a mix of different types of instrument manufactures and software suppliers. Usually organisations have more people who wish to view the data than instrument workstations that are available to present the data in its proprietary format. Another problem that arises is that regulatory compliance data needs to be stored for longer periods than the useful lifetime of an instrument.

The solution to this problem is to use an open file format that structures the information generated from the instrument. The format chosen was a variant of XML, the extensible markup language. XML has the following advantages as an information exchange language:

- ❑ Public domain standard not owned by any one corporation.
- ❑ Uses ASCII as its storage medium.
- ❑ It uses a system of hierarchical data structures to store complex relationships.
- ❑ The DTD or schema can be widely distributed to software developers.
- ❑ Common data exchange format between businesses.

Communication Protocols

USB (Universal Serial Bus)

Universal Serial Bus was designed as a serial bus replacement for PC's. It was primarily used for connecting peripheral devices such as scanners, disks, cameras etc. Universal Serial Bus is a plug and play technology, the USB host automatically detects when a new device has been added, queries the device for its identification and configures the device drivers appropriately.

USB is an inexpensive and easy-to-use connection between instruments and PCs. USB improves on conventional serial port technology by providing faster performance - the USB 2.0 specification supports data throughput of 60Mbytes/sec. Up to 127 USB devices can run concurrently on one port. USB connectivity is currently implemented in Microsoft Windows 2000 and XP and USB drivers are also available for Linux. The downside of USB for instrument control is that the cables are not shielded and therefore can potentially suffer data loss in noisy environments.

FireWire (IEEE 1394)

FireWire is a high performace serial bus that was pioneered by Apple Corporation Inc. to address the needs of high-speed data. It is not currently as widespread as USB but does offer some advantages. Its main features are as follows:

- ❑ The ability for transmission to occur at **different speeds**. For example, some devices can communicate at 100 Mbps, while others communicate at 200 or 400 Mbps within the same medium.
- ❑ **Low susceptibility to noise**. FireWire uses a pair of signal lines for each channel of communication, with opposite current and voltage swings. Any external noise is coupled onto the two wires as a common mode voltage and is rejected by the receiver.
- ❑ **Both asynchronous and isochronous transactions** on the same interface. This allows both non-real-time and real-time critical applications on the same bus.
- ❑ **Flexible daisy-chain or tree configuration**. Each FireWire serial bus node has one or more ports and acts as a repeater. A single-port node discontinues the bus along a given line whereas nodes with two or more ports act as repeaters and allow continuation of the bus, eliminating the need for hubs when connecting many devices to the same network.

Up to **63 devices** may be connected to a FireWire bus segment and **1023 segments** may be connected via bridges. Devices may be up to **4.5 metres apart**.

- ❑ **Peer-to-peer transfers** are supported so that serial bus nodes can transfer data between themselves without the intervention of the host system. For example, a video camera can set up a transaction between itself and a video recorder, if both reside on the same serial bus.
- ❑ **Connecting cables carry both signals and power**. This reduces the need for extra power cables when devices are connected to the bus.
- ❑ **Plug and play**. FireWire provides automatic reconfiguration that allows devices to be connected and disconnected while the network is active.

FireWire provides benefits over most of the available data transfer standards.

- ❑ High-speed transmission that allows a wide range of performance devices to be attached on the same bus.
- ❑ Plug-and-play facility with automatic configuration.
- ❑ Tree or chain topology.
- ❑ Inexpensive and easy to connect.
- ❑ FireWire interface card can be easily installed on a PC.
- ❑ Programming APIs are available for the windows environment.