DATA TRANSLATION®

The Evolution of Virtual Instrumentation

What is Virtual Instrumentation?

A virtual instrumentation system is computer software that a user would employ to develop a computerized test and measurement system, for controlling from a computer desktop an external measurement hardware device, and for displaying test or measurement data collected by the external device on instrument-like panels on a computer screen. Virtual instrumentation extends also to computerized systems for controlling processes based on data collected and processed by a computerized instrumentation system.

An instrument is a device designed to collect data from an environment, or from a unit under test, and to display information to a user based on the collected data. Such an instrument may employ a transducer to sense changes in a physical parameter, such as temperature or pressure, and to convert the sensed information into electrical signals, such as voltage or frequency variations.

The term instrument may also cover, and for purposes of this description it will be taken to cover, a physical or software device that performs an analysis on data acquired from another instrument and then outputs the processed data to display or recording means. This second category of instruments would, for example, include oscilloscopes, spectrum analyzers and digital multimeters. The types of source data collected and analyzed by instruments may thus vary widely, including both physical parameters such as temperature, pressure, distance, and light and sound frequencies and amplitudes, and also electrical parameters including voltage, current, and frequency.

History of Instrumentation Systems

Historically, instrumentation systems originated in the distant past, with measuring rods, thermometers, and scales. In modern times, instrumentation systems have generally consisted of individual instruments, for example, an electro-mechanical pressure gauge comprising a sensing transducer wired to signal conditioning circuitry, outputting a processed signal to a display panel and perhaps also to a line recorder, in which a trace of changing conditions is inked onto a rotating drum by a mechanical arm, creating a time record of pressure changes.

Even complex systems such as chemical process control applications typically employed, until the 1980s, sets of individual physical instruments wired to a central control panel that comprised an array of physical data display devices such as dials and counters, together with sets of switches, knobs and buttons for controlling the instruments.

The introduction of computers into the field of instrumentation began as a way to couple an individual instrument, such as a pressure sensor, to a computer, and enable the display of measurement data on a virtual instrument panel, displayed in software on the computer monitor and containing buttons or other means for controlling the operation of the sensor. Thus, such instrumentation software enabled the creation of a simulated physical instrument, having the capability to control physical sensing components.

Creation of Virtual Instrumentation

A large variety of data collection instruments designed specifically for computerized control and operation were developed and made available on the commercial market, creating the field now called “virtual instrumentation.” Virtual instrumentation thus refers to the use of general purpose computers and workstations, in combination with data collection hardware devices, and virtual instrumentation software, to construct an integrated instrumentation system; in such a system the data collection hardware devices, which incorporate sensing elements for detecting changes in the conditions of test subjects, are intimately coupled to the computer, whereby the operations of the sensors are controlled by the computer software, and the output of the data collection devices is displayed on the computer screen, in a manner designed in software to be particularly useful to the user, for example by the use of displays simulating in appearance the physical dials, meters and other data visualization devices of traditional instruments.

Virtual instrumentation systems typically also comprise pure software “instruments”, such as oscilloscopes and spectrum analyzers, for processing the collected sensor data and “massaging” it in ways useful to the users of the data. For example, software means may be employed to analyze collected data as needed to present the user with isolated information on “max” or “min” readings, averages, standard deviations, or combinations of results from related sensing points, etc.

Early Challenges

The early development of virtual instrumentation systems faced challenging technical difficulties. Major obstacles included the many types of electronic interfaces by which external data collection devices can be coupled to a computer, and great variety in the “command sets” used by different hardware device vendors to control their respective products.

Also, data collecting hardware devices differ in their internal structures and functions, requiring virtual instrumentation systems to take these differences into account. Thus, some data acquisition devices are so-called “register-based” instruments, controlled by streams of 1s and 0s sent directly to control components within the instrument; others are “message-based” instruments, and are controlled by “strings” of ASCII characters, effectively constituting written instructions that must be decoded within the instrument. In turn, different instruments use different protocols to output data, some as electrical frequencies and others as variations in a base voltage, for example. Any virtual instrumentation system intended for connection to a typical variety of commercially available data collection hardware devices must accordingly comprise software tools capable of communicating effectively with the disparate types of hardware devices.

Programming Requirements

Until the 1990’s, the programming of virtual instrumentation systems was a task strictly for professional programmers, who wrote the required software programs using “textual” programming languages such as BASIC, C ++, or PASCAL. In addition to the factors previously mentioned, the great variety in possible applications also called for professional expertise, in that one customer's measurement application was rarely suitable for another customer. For example, one customer may have needed only to collect a single value, outside temperature, once an hour, and to have all collected values stored in a file. The next customer possibly required that several related process temperatures, in a rubber-curing process, be monitored continuously, and that a shut-off valve be activated in the event that the relative temperature between two process steps should vary by more than seven degrees for a time period of three seconds or more, in any 15 minute period, and to store only data concerning such episodes.

The development of instrumentation systems software by professionals using textual programming languages such as C++ is very time consuming and tedious, and it typically results in the production of a program consisting of many pages of source code, written in a computer language that is virtually unreadable by non-programmers, and which thus cannot be modified by the typical users of such programs.

Drawbacks of Recent Approaches

In the last ten years, there have appeared several commercial software products for the development of virtual instrumentation systems using purely graphical programming methods. Each of these products provides users, typically including users who are not skilled software programmers, with a "graphical development environment" within which to design a custom virtual instrumentation system. Typically, the user is presented with a "design desktop" environment, generally having the look-and-feel familiar to users of Windows®-based graphical applications, in which a variety of software options and "tools" are accessible from toolbars and dialog boxes featuring drop-down menus, and may be accessed by manipulating an on-screen cursor using a computer mouse.

However, these older software packages for developing virtual instrumentation systems, using graphical programming means, provide the user with tools for designing so-called "data flow" diagrams. The user of these software packages is thus required both to place icons representing desired sys tem components onto a design desktop and then to effect “wiring” connections between components. In order to design a data flow diagram that corresponds to a workable measurement system application, the user is required to have comparably deep knowledge and understanding of the specific data paths and element combinations that will be required to attain the user’s objective, which is a “solution” to the user’s measurement requirements. User designed systems developed using this type of software are also prone to errors, because they generally allow the user to unwittingly wire together components that are functionally incompatible.

The Latest Developments

The range of applications that may be made the subject of an instrumentation system spans the range of human activity. Therefore, a software development system that aims to provide a large cross-section of potential users with the tools to design their own customized instrumentation system must provide the user with a large range of development tools and options, including tools and options that may be or are mutually incompatible in a given application. Ideally, such a system should also organize the software tools provided to the user in a way that enables the user to select easily the particular tools best suited for the pertinent application, and guide the user’s selection of configuration options. Older software packages lack built-in safeguards against the construction of unworkable combinations of components, and provide inadequate intuitive guidance to the user seeking to develop a measurement application.

An important objective in the development of Data Translation’s DT Measure Foundry was to present the user with development tools in a manner that guides the user intuitively to choose system elements that are appropriate to the user’s project, and that automatically precludes the user’s selection of system elements that are mutually incompatible.