XVI IMEKO World Congress
Measurement - Supports Science - Improves Technology - Protects Environment ... and Provides Employment - Now and in the Future
Vienna, AUSTRIA, 2000, September 25-28

INSTRUMENT-INDEPENDENT SOFTWARE: A CASE STUDY

M. Bertocco and C. Narduzzi
Dipartimento di Elettronica e Informatica
Università degli Studi di Padova
via G. Gradenigo, 6/a, I-35131 Padova, Italy

Abstract: Recently, a standardised software interface for measuring instruments has been proposed by the IVI (Interchangeable Virtual Instruments) Foundation. The aim of the proposal is to provide independence of measuring system software from the specific instrumentation employed in a particular implementation of a test system, allowing for total interchangeability of instruments within a given class. Replacement or upgrades of hardware within an automatic test system would then be possible without affecting measurement software. The authors have implemented IVI-like drivers for some instrument classes, and have analysed their impact on the performances of a simple measurement system. The paper presents the main results of this analysis, providing indications about the implementation of IVI drivers in the most common visual environments for instrumentation.

Keywords: automatic test system, interchangeable virtual instrument, software.

1 INTRODUCTION

A considerable proportion of the automatic measuring and test systems employed by the industry is represented by modular systems, or by so-called “rack and stack” systems. These are often combinations of general-purpose instruments with adapters, switches and interface circuits, enabling specialised measurement procedures to be carried out on a given device under test. Compared to test equipment specifically designed for a single purpose such systems may have some performance shortcomings, but they have the advantage that their building blocks can be employed for different applications or, possibly, re-used when a given test set is no longer required.

In an automatic system, the knowledge related to a particular measurement procedure is encoded in the software that controls and directs the operation of the system itself. It is not surprising, therefore, that test software represents a large part of the investment involved in system development. Software must translate the measurement procedure into commands that are sent to the programmable instruments, or instrument modules, within the system. Drivers are usually available to deal with low-level programming issues, thus giving test engineers more freedom to concentrate on the high-level design of the measurement procedure. Over the years, this has given rise to a layered structure, not unlike the OSI reference model. Unfortunately, some aspects of instrument programming can still be very much device-dependent, so that if some component of a measuring system is changed, modifications are likely to be required to its software as well.

2 THE IVI STANDARD

The IVI (Interchangeable Virtual Instruments) Foundation has proposed a standard where functional classes, in addition to specific instruments, are defined. The first versions of IVI specifications were released in mid-1998 [1] and covered five different instrument classes: function generators, oscilloscopes, digital multimeters, power supplies and switches. Further class specifications are under development at the time of this writing, including spectrum analysers and power meters. These specifications define class drivers associated to each instrument class, representing a higher-level software layer than instrument-specific drivers. The aim of these class drivers is to allow a test program to be written with the highest possible degree of device independence. Ideally, substituting an instrument in an automatic test system for another in the same class should require little more than notifying the change by updating an instrument list (or system configuration file).

For each class a number of device-independent functions and attributes are specified, which represent the instrument capabilities. They are divided into fundamental capabilities and some extension groups, the former group including all the capabilities that are common to any instrument within a class. Each function specification in a class driver defines the number, structure and format of

input and output parameters, as well as the function name; thus, the test system developer is presented with a standardised Applications Programming Interface (API). Furthermore, inherent IVI capabilities are defined and shared by all types of IVI class drivers.

Interchangeability can have different meanings and limitations. Intuitively, the most appealing definition is the broadest, i.e., the possibility to obtain the same measurement results, regardless of the instrumentation employed. This aim is also by far the most ambitious and difficult to achieve. Class driver specifications implement syntactical interchangeability, which ensures that test program code will not need any modification after an instrument change in the test system. Further features in the IVI drivers account for the fact that hardware and performance differences among similar instruments do exist, and allow the introduction of suitable checks. A feature of particular importance is range checking: every time a value is sent to an instrument, the driver checks that the value falls within the acceptable range for the setting concerned and, if necessary, forces it to a safe value. Another feature, interchangeability checking, evidences any instrument-specific attribute setting that a user may have introduced in the drivers of his test program.

3 THE APPLICATION ENVIRONMENT

The work reported in this paper is part of a project that involves the measurement laboratory of the Dipartimento di Elettronica e Informatica of the Università di Padova. Teaching needs have led to the formulation of a set or requirements that is remarkably similar to those of interchangeable measurement systems.

A significant part of the contents of basic courses in instrumentation and measurement is devoted to instrument programming and automatic measurement. Laboratory exercises are required, but, because of obvious limitations in time and effort, it would be hard for students to enter into the intricacies of low-level instrument programming before producing a complete, correctly working test program. This problem is further evidenced when differences in the behaviour and programming syntax of functionally similar instruments are taken into consideration. Two approaches had been considered to address the problem: the use of specific drivers and the adoption of SCPI (Standard Commands for Programmable Instrumentation) as a generic syntax.

Instrument-specific drivers are commonly available in several application development environments. National Instruments LabVIEW and Hewlett-Packard (now Agilent) VEE are employed in the laboratory, both providing driver support. The use of these drivers simplifies instrument programming by making the low-level issues entirely invisible to the test developer, which is not always satisfactory for educational purposes. In some cases simplified drivers, rather than full-featured ones, would be preferable to allow students to concentrate on key issues. For this reason, suitable driver libraries were developed in-house.

The alternative approach of standardising on SCPI commands required the development of a suitable software interface, that translates SCPI commands into non-SCPI ones for those instruments that do not comply with this standard. This was necessary since a number of instruments in the laboratory are programmed by a different command set, even SCPI-like commands presenting subtle differences. On the other hand, since a parser had to be implemented within the command translator to identify different fields in a string and extract parameter values, a logical extension of the program was the implementation of range checking. This is seen mainly as a safety feature in the environment of an educational laboratory. In fact, range checking has been implemented so that it can refer not only to the appropriate instrument range, but also to safe limits for the measurement experiment that students are required to implement.

The approach based on SCPI commands complements the use of drivers and allows students to practise message-based instrument programming. In both cases, however, a degree of dependence on the specific instruments employed remains. This significantly affects laboratory organisation, since a variety of instruments are intentionally made available to show different solutions, and it is important that the same measurements can be obtained by different sets of instruments.

The approach to interchangeability proposed by the IVI Foundation promised to improve the situation, so it was decided to evaluate the IVI concept by the development of IVI-like class drivers, to be tested in the laboratory. IVI Foundation considers implementation in the form of Dynamic Link Libraries (DLL's) [2], although further developments based on object-oriented software technology are being discussed. In this case it was preferred to implement an "IVI layer" using the available visual environments for instrumentation mentioned above, i.e., LabVIEW by National Instruments and VEE by Hewlett-Packard. As will be explained in the next section, the resulting architecture is quite similar, and the implementation also relies on the concept of dynamic loading of specific instrument drivers by the IVI class driver.

4 IMPLEMENTATION OF AN IVI LAYER

A large number of instrument-specific drivers had already been developed in the two programming environments mentioned above, therefore it was decided that the IVI layer should be designed to make the largest possible use of existing software. This marks a departure from the IVI system architecture, where instrument-specific drivers are still expected to be IVI-compliant [3]. On the other hand, naming and interface conventions of the IVI specifications are complied with by the developed software, which is in fact a Class Driver placed on top of a non-IVI layer. Application programs are thus enabled to interface with test instruments either by IVI Class Drivers, or by already existing instrument-specific drivers. In a laboratory exercise, it is expected that the test program developer will mostly turn to class drivers, but in some cases access to some peculiar instrument functions may be needed, and is better accomplished at the lower level.

It should be stressed at this point that the purpose of this work is not the full implementation of the IVI system architecture, but the evaluation of its main concepts. As a result of the requirements outlined in the previous section, it was decided to implement a subset of the functions defined in one of the IVI Class Driver specifications, namely, the Function Generator, including the range checking capability. Specific aspects to be considered included:

1. re-use of existing instrument drivers: the class driver implementation may introduce capability extensions, but should not require modifications to already existing software;
2. existing driver functions must remain accessible;
3. naming and attributes must conform to the relevant IVI specification;
4. the resulting class driver must be contained in a library and be ready for use.

Additional software specifications refer to the particular structure of the measurement laboratory where the project is carried out. A LabVIEW or VEE application does not control instruments on the workbench directly; rather, it interfaces to a local server, which forwards the commands to the instruments. Local servers on different workbenches are connected through a local network to a laboratory server, which is also a gateway to a geographical network. This solution allows the choice of different options, since a client application can send commands either to instruments directly connected to a bus (for instance, GPIB), through the local server, or to instruments connected to a remote server, accessed through the network. The reasons for this implementation have been presented elsewhere [4]. Here it suffices to remember that the class driver developed in this project must be able to operate both with local instruments and with remotely accessed instruments.

IVI specifications do not enter into details concerning implementation; therefore a developer is given freedom to organise the project according to needs. Driver functions have been developed according to a common architecture, as shown in Fig. 1, where it can be seen that existing instrument drivers have been encapsulated in the IVI layer. When an IVI function is activated, the measuring system controller retrieves information on the hardware configuration of the system, as well as IVI-specific information, from a database, which is regularly updated. On this basis, a procedure determines the specific driver required, dynamically loads it and puts it in execution. An input adapter is also required, to account for possible differences in the number of input terminals between the defined IVI function and the corresponding non-IVI driver.

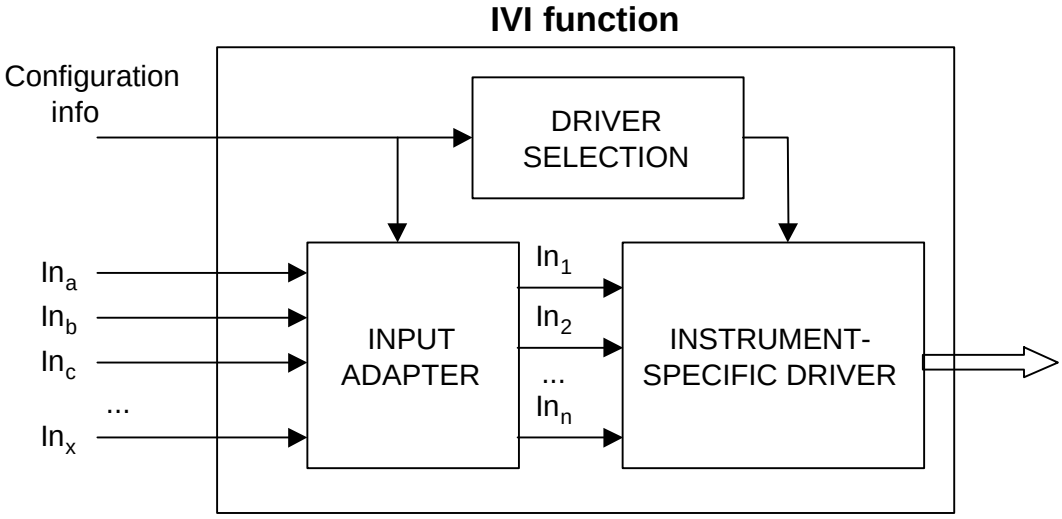

Figure 1. The building blocks of the implemented IVI driver functions

This architecture has been the result of progressive improvements, introduced so that the possibilities offered by the LabVIEW and VEE development environments could be best exploited. The key feature employed in the development of the class driver is the support for dynamic function calls, which is provided by both environments. The end result turned out to be conceptually similar to the IVI system architecture, since the driver selection function of Fig. 1 can be equated to the IVI engine and the configuration info is obviously equivalent to the IVI Configuration File. Seemingly, the availability of support for dynamic loading in any environment almost naturally leads to such architecture.

An important issue in the development of an instrument-independent class driver is that, until its activation, it is not known which instrument-specific driver function will be involved, therefore dynamic function calls have to be employed. It is also necessary to provide the driver with configuration information indicating the real instrument name that should be associated to the class call: this is obtained in two different ways, one for local instruments, the other for remotely accessed instruments. Another problem is represented by the fact that driver functions referring to different instruments may differ with regards to the number and definition of input variables (it should be remembered that instrument-specific drivers are assumed to be not compliant with IVI specifications).

The structure of Fig. 1 has been designed to account for these aspects; the discussion that follows will give a few implementation details, referred to the VEE environment.

4.1 Dynamic driver function selection

The IVI class driver must receive information about the association of the IVI class name with the instrument real name. Such information is related to the configuration of the measuring system, and should be available in a configuration file. In the laboratory environment described in Section 3 the situation is more complex, since a client application can control instruments on either a local or a remote workbench. Only the configuration of the former can be known to the local server, which maintains a file called `local_config.txt` where each instrument IVI class, real name, bus interface type, bus interface number and bus address are stored. When a remote workbench has to be accessed, information must be provided by the laboratory server, which can return data concerning the local configuration file from any local server.

A single VEE Call Function object is used to call the selected instrument-specific implementation of any class driver function; this is a dynamic call, where the instrument real name represents an input variable of the call. Unfortunately, different instruments may vary in the number and meaning of inputs associated to such functions, and it is necessary to devise a mechanism that will correctly generate the calls regardless of these differences.

The problem has been split in two parts. A function called *IVI_Input_Adapter* has been implemented, with the purpose of creating the correct associations between the IVI function input variables and the input variables of the corresponding instrument-specific function. The interface provided by this object allows assigning the appropriate variable names and forming the correct links, with the help of a file called `check.txt`. The need to implement the *IVI_Input_Adapter* is a direct consequence of the design choice to allow non-IVI specific drivers to be dealt with by the IVI Class Driver. Output from this object is a set of six variables, plus a further variable which indicates how many of the six are actually used, the remaining ones being filled with empty strings.

Activation of the requested driver function is achieved by calling the object *DriverFunction*, which receives the variable indicating the instrument-specific name of the requested function, as well as data from *IVI_Input_Adapter*. *DriverFunction* selects one object in a set of objects called *OnlyPanel, 1input, 2input,* …, *6input*, on the basis of information concerning the actual number of input variables received from *IVI_Input_Adapter*. A dynamic call to the selected object is then performed, and the specific name variable is forwarded to the called function. This approach allows dealing with functions having up to six input variables, which seems a reasonable number, without unnecessary duplications in the program structure.

4.2 Sample application

The implemented IVI drivers have been tested in some simple measurement applications, showing that the approach proposed in the paper performs satisfactorily. To obtain an assessment of the performances of the implemented IVI Class Drivers and compare them with those of instrument-specific drivers, the function generator class has been considered. A subset of the full class driver has been implemented, this comprising the three functions:

- *IVIFGEN_SETATTRIBUTE_FUNC_AMPLITUDE*
- *IVIFGEN_SETATTRIBUTE_FUNC_FREQUENCY*
- *IVIFGEN_SETATTRIBUTE_FUNC_WAVEFORM*

that allow setting the waveform amplitude, frequency and shape, respectively. These drivers have been employed to produce a simple VEE application, which generates a sequence of four different waveforms (sine, square, triangle and ramp) at a given frequency, and for each waveform progressively increases the amplitude from 0.1 V_{pp} to 0.7 V_{pp} in 0.1 V steps. Generated waveforms are acquired by an oscilloscope and displayed in a plot. The test has been performed using instruments connected to the local server on a laboratory workbench: in this way no remote access mechanisms are involved an no network overhead is incurred in to establish the association of the instrument real name with the class name. The application generates a significant amount of instrument commands, so it is considered representative of a real-life situation.

The same application has been implemented using the specific drivers of a Hewlett-Packard 33120A function generator; in this case the whole process of input adaptation and driver selection is dispensed with, since the `hp33120a` driver is called directly. The rest of the procedure remains exactly the same; therefore variations in execution time are exclusively due to the differences between drivers.

Within each application, a timer object records the time needed to execute the whole procedure. The test yielded an execution time of 20.1 s using instrument-specific driver functions and 24.2 s using IVI class driver functions. The total execution time is not representative, since several operations carried out during the procedure are not involved in the comparison. As mentioned, such operations are performed in the same way in both cases, therefore the difference between the two execution times gives a meaningful idea of the computational overhead represented by the approach discussed in this paper. The IVI procedure comprises approximately 30 IVI function calls, therefore the 4.1 s difference corresponds to 140 ms per call. If the time difference is considered in relation to the total execution time of the procedure, the increase is approximately 20%, which is considered to be a generally realistic indication, as far as the order of magnitude is concerned.

Interchangeability was checked by using three different generator types, and no difficulties were found. As already noted in Section 2, the check involves syntactical interchangeability of the instruments, while performance differences cannot be accounted for.

5 FINAL REMARKS

The approach to instrument interchangeability presented in this paper has been extensively tested with fully satisfactory results. IVI-like drivers developed within this project are currently in use in an educational laboratory, being available within an Instrument-Independent Library (IIL) on workbench local servers. Their use has reduced applications development time, making the development of a simple automated measurement procedure manageable within the time constraints of a laboratory exercise.

The increase in execution time caused by the addition of the IVI layer before the applications software has been found to be rather limited, and acceptable in an educational environment, when the advantage of instrument interchangeability is taken into account. On the other hand, automatic test systems for industry applications may have some time-critical areas where such overhead is not acceptable. It is likely, therefore, that test software will be developed using a mix of high-level IVI drivers, instrument-specific drivers and direct I/O interface functions, depending on the performances the test system should provide.

REFERENCES

[1] IVI Foundation, *IVI-4: IviScope Class Specification; IVI-5: IviDmm Class Specification; IVI-6: IviFgen Class Specification; IVI-7: IviPower Class Specification; IVI-8: IviSwitch Class Specification*, August 1998 Edition, Revision 1.0

[2] M. Rowe, B. Kerridge, Software Drivers: What Are They Exactly?, *Test & Measurement Europe* **7** (3) (1999) 16-20.

[3] National Instruments Corporation, *IVI Driver Library User Manual*, 1999.

[4] L. Benetazzo, M. Bertocco, F. Ferraris, A. Ferrero, C. Offelli, M. Parvis, V. Piuri, A Web-Based Distributed Virtual Educational Laboratory, in: V. Piuri and M. Savino (eds.), *Proceedings 16th IEEE Instrumentation and Measurement Technology Conference* (Venice, Italy, 24-26 May 1999), IEEE, USA, 1999, p. 1851-1856. To appear in: *IEEE Transactions on Instrumentation and measurement* **49** (2) (2000).

AUTHORS: Prof. Matteo Bertocco and Prof. Claudio Narduzzi, Dipartimento di Elettronica e Informatica, Università degli Studi di Padova, via G. Gradenigo, 6/a, I-35131 Padova, Italy. Phone: +39 049 827 7500, Fax: +39 049 827 7699, E-mail: mat@dei.unipd.it, narduzzi@dei.unipd.it