# SCPI and VISA a valuable combination

***By John M. Pieper***

**(C) ACEA, Wierden The Netherlands**

***Recently the VXI Plug and Play Alliance introduced an open system architecture for virtual instrumentation, called VISA. A hardware and system level software environment is created on different computers and operating systems in order to allow end users of instrumentation systems to apply multi-vendor products in an interoperable way.***
***Key technologies for the remote operating of instruments are the SCPI (Standard Commands for Programmable Instruments) and its underlying IEEE 488.2 standard.***
***This article explains the importance of these standards to both end users and instrument vendors when applying instrument drivers on a VISA platform and describes the valuable combination of SCPI/IEEE 488.2 and VISA.***

---

The VISA concept has been driven by the vendors of VXI hard and software products. VXI instruments cannot be controlled, except by software; virtual instruments, who display a front panel image on a computer screen, is the keyword. But the vendor's hard and software products being used to create VXI applications, were mutual incompatible. Application programs couldn't be ported among the different vendor specific platforms. It is the merit of the alliance that this interoperability is much improved, if not yet guaranteed.

Apart of the interoperability issue, the alliance did also require a number of additional provisions which are deemed to be part of a professional development environment for virtual instrument control.

## Relation of VISA with the other ATE standards

The VISA protocol is a next logical step in a series of standards ATE systems. As VISA's main objective is to guarantee interoperability, main part of the requirements apply to the interfacing of the application layer functionality with its environment. In this context, '*environment'* is a rough indication of all the components which interacts with the functionality of the application, like the computer hardware and the operating system, the environment in which the application is developed, the interface platform via which the application communicates with the instruments and the corresponding I/O drivers and interface cards which are used for that purpose.

Rather than standardizing the instrument control, the processing functionality or the users interface, VISA provides a number of requirements and guidelines for the interaction with their environment. Suppliers of VISA compatible products must comply to these requirements in order to offer their customers the possibility to port their applications to different '*environments'*, which can be linked with all needed functions and be linked to other commercial available software packages, which, for example are needed for documentation and archiving the test results and procedures.

VISA is complementary to the other ATE industry standards, and does certainly not leave these standards behind. On the contrary, it builds on the standardization results, achieved for the underlying functional layers, which are needed to meet the demands of modern instrumentation systems. To clarify their mutual relations, figure 1 gives a survey of the applicable ATE standards. Notice that the grey shaded parts reflect the areas which are subject of standardization.

On the lowest level there is the definition of the interface platform itself. This can be the well known IEEE 488.1/IEC 625-1 interface, commonly called GPIB, but other type of interface media of later date, like VXI or RS 232 with IEEE 1174, can be associated also with this layer. The functionality found in this layer, defines the electrical and mechanical properties of the interface medium and contains protocols for establishing the data path between the controller and the instruments.

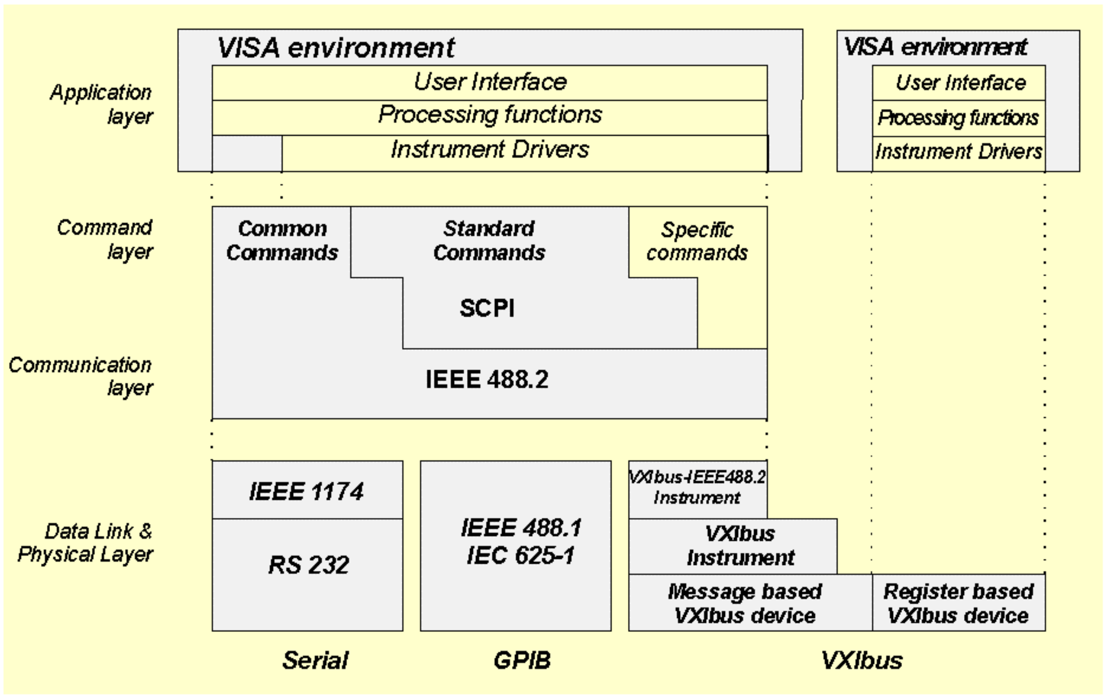

**Figure 1.** Survey of the ATE standards**.**

These lower layer standards do not deal with the contents of the data itself. For example, one may have a phone connection of excellent technical quality with Japan, but if one does not speak Japanese, communication may become troublesome.

## The IEEE 488.2 concept

Therefor, by defining the message syntax (spelling), the IEEE 488.2 provides a concept that meets the demands for instrumentation systems. The syntax is a perfect balance between the communication needs of a modern instrumentation system and the requirement to unambiguously parse (interpret) the messages in an efficient way.

But IEEE 488.2 is more than a syntax definition only. It was clearly shown by the experiences of the instrument manufacturers who took the initiative to this standard, that a protocol was needed to guarantee a reliable communication, which would not hang up under any condition. Therefor, the so called Message Exchange Protocol - MEP - was established. This protocol is based on the principle that 'an instrument may not send data until it is asked for'. For that purpose, IEEE 488.2 distinguishes between commands and queries. This concept is anchored in the language construction, since commands and queries follow different syntax's.

The requirements of the Message Exchange Protocol are simply to be met by an application program (controllers), but its implementation requires more effort from the side of the instrument designer. This is because an instrument has to detect any communication error from the side of the controller and be able to recover from any possible 'hang-up' or 'lock' condition.

Furthermore, the IEEE 488.2 standard provides protocols for:

- status and error reporting and retrieving.
- synchronization between the controller program and instrument actions and events, and between different instruments mutually.
- dealing with coupled functionality and resolving coupling conflicts; an issue which is commonly left unmentioned, but certainly exists in major part of modern instruments.

The definition of the semantics (meaning) of program messages is not part of the IEEE 488.2 standard. This was left to higher level standards like SCPI, or to the instrument designer. However, the so called 'common commands' is the exception to this rule. Most of these common commands deal with general 'house keeping' functionality from which today's modern instruments cannot desist.

It is therefor not a coincidence that the VISA required set of instrument drivers is based upon these IEEE 488.2 common commands. To implement these drivers for a IEEE 488.2 compliant device, little or no modification is needed. Non-IEEE 488.2 compliant instruments have to develop equivalent functionality for the VISA drivers or to return warnings.

## The VXI platform

Originally, IEEE 488.2 was defined as a layer above GPIB (IEEE 488.1/IEC 625-1). When the VXIbus standard was created, much attention is paid to keep the value of IEEE 488.2 available for VXI devices and to maintain compatibility with the other ATE standards. The VXI Word Serial protocol therefor defines a number of commands which emulate GPIB functionality. VXIbus instruments are defined as a class of message based VXIbus devices which use these emulation commands. This means that the Data Link and Physical layer functions of GPIB and VXI instruments are transparent for the higher level protocols, like IEEE 488.2. This implies also that IEEE 488.2 functionality does not need to distinguish between both platforms and can be ported to VXI as well as to GPIB interfaces. This enables VXIbus instruments to fully comply to IEEE 488.2; such VXIbus devices are called 'VXIbus-IEEE 488.2 instruments'.

VXIbus instruments belong to the category of 'message based' VXIbus devices. The program messages being used to control this device category are ASCII based commands. As such, they can be compared with the way the traditional GPIB instruments are controlled. Also, all ATE standards can be applied to message based VXIbus devices.

Another category of VXIbus devices are the so called 'register based' VXIbus devices. They communicate with arbitrary 8 or 16 bit binary codes, which are 'peeked' and 'poked' into registers. Although this low level way of communication may be very fast, no standards applies to this type of data transfer. The control of register based VXIbus devices is entirely determined by the manufacturer and is very device specific. Programming this type of instruments, for example by creating VISA drivers, may easily become a time consuming and tedious job.

## IEEE 1174

To maintain the compatibility among instrumentation interfaces, the new IEEE 1174 standard has a similar philosophy as the VXIbus. This serial instrument standard, whose publication is to be expected about the end of this year, defines a GPIB function emulation on a serial RS 232 port. Additionally, it describes the implementation of the IEEE 488.2 protocols on a serial port.

## SCPI

The primary goal of the Standard Commands for Programmable Instruments SCPI is to reduce the time needed to create an application program. Although instrument drivers existed before SCPI was established, major part of programmers did write their applications using I/O drivers. This type of drivers do send only the specified command to the instrument and retrieve their response directly and do not depend upon the instrument functionality. The VISA/VTL I/O drivers are a good example of such a driver set.

The wide variety of different remote programming concepts was one of the biggest problems which a programmer had to face. Different commands were used to control identical functions in instruments from different suppliers. But also the reverse occurred; identical commands had different meaning in instruments from different vendors. As will be explained in this article, these problems are not solved by simply using VISA instrument drivers.

SCPI solved this problem in several steps. As a first step, a generic model of a programmable instrument is created. This so-called '*instrument model'*, maps the instrument functionality and belonging data and control flows into a structured chart. Based upon this 'instrument model', the hierarchical command tree is developed as a second step. The programming commands, which use the IEEE 488.2 syntax, are directly linked to the structured functionality of the 'instrument model'; they are easy to learn and self-explanatory to both novice and expert users. The structured approach of the instrument model did contribute a lot to achieve the primary goal: reducing multiple ways to control the same functionality.

## The VISA concept

The main benefit of the VISA concept is that it establishes a solid and rigid hard and software environment for virtual instrumentation and allows end users to port their applications among different tools. Within these tools, different types of functionality, dealing with instrument control is distinguished, as is shown by figure 2.

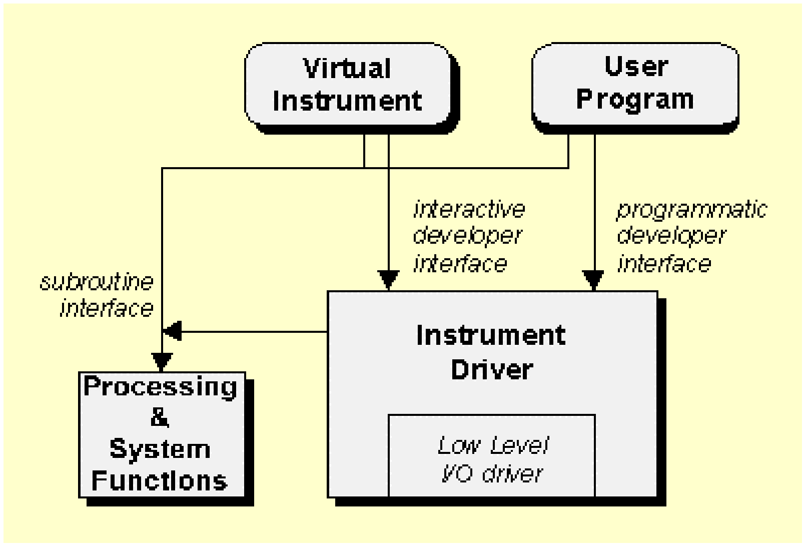
Virtual Instrument
User Program
interactive developer interface
programmatic developer interface
subroutine interface
Instrument Driver
Low Level I/O driver
Processing & System Functions

**Figure 2.** Application layer functionality

## The user interface

The way the user interacts with the application is called the 'user interface'. Such an interface can be accomplished by:

- a *virtual instrument*, being created with one of the supplied tools. All kinds of objects are available for visualization in a way that best suits the nature of the quantity to be presented. To show measuring results and status information, meters, bars, plotter functions, numeric displays, etc. are available. Also the outcome of processed results, like trends, filtered waveforms, hystograms, etc. may be displayed by such objects. Objects, like knobs, slides, switches, etc. provide input to the application and often interacts

directly with the instrument drivers. In this way a mechanism is created for intuitive visualization of the application.
- a traditional application or *user program* which as a higher level software program calls instrument driver functions to perform remote operations on the instruments.

## Process and System functions

An application might need to process the outcome of the measurements, as for example, to monitor the results, to alarm when specified limits are exceeded. Calculations, as domain transformations by FFT algorithms, statistical analysis or calculating border limits are functions which are commonly needed in major part of applications; they can be accessed via what is called by VISA a '*subroutine interface*".

## Instrument drivers

An instrument driver basically is a piece of code, which is called from inside an application program (programmatic developer interface), or by a virtual instrument (interactive developer interface); they perform one or more specific remote controls to an instrument. The functionality of the instrument driver is entirely determined by its designer and is deemed to do exactly what the user needs for a particular application. This may vary from sending a simple command to the instrument up to a complex high level test which operates on multiple instruments and uses support libraries, for data analysis or domain transformation.

As a software support for their instruments, vendors may supply *instrument drivers*. Major part of this type of drivers perform a simple control of a single function; they just send the command needed to set the desired function.

For example, to set the voltage attenuation of a hypothetical instrument, the driver *acea2111_set_volt_att(vi,a)* might be provided. In this example, the prefix 'acea2111' is entirely determined by the instrument manufacturer, except for the first two characters. The two leading characters need to be approved by the VPP consortium who is responsible for the VISA standard. This is to prevent that two different vendors use identical driver names for their instruments. The prefix unambiguously identifies the manufacturer and model of the instrument. In the example given, 'acea' would be the company name and '2111' the model number. The parameter vi is the instrument handler, which is assigned to the instrument during initialization by the *acea2111_init()* procedure. The instrument handler 'vi' uniquely identifies the instrument and is used by the drivers to refer to the particular instrument. Finally, the parameter 'a' specifies the desired attenuation value.

In order to achieve the desired attenuation, all the driver has to do is sending the appropriate command to the instrument, which in this case would be the SCPI command SENS:VOLT:ATT <value>.

## Low level I/O drivers

Instrument drivers acces the instruments by sending commands and retrieving results, using so called '*low level I/O drivers'*. Such drivers take care about the data transfer over a particular interface platform, like GPIB, VXI, RS 232, etc. They just send or retrieve data from an address at a particular interface platform and do not depend upon the type of instrument being connected at the address. Hence they are instrument independent. Up to now, their naming and functionality depended upon the vendor or the manufacturer of the interface card used.

It is the merit of the VPP consortium, that they have established a VISA standardized set of low level I/O drivers which are independent upon the manufacturers of I/O driver and interface boards. Furhtermore, the VISA I/O drivers calls do not distinguish between the interface platforms being used. This is done by the instrument handle vi, which within the I/O driver uniquely identifies the instruments on the particular interface platform. There are I/O procedures, which are common to all platforms, like viWrite() and viRead(). Calls to this category of I/O drivers do not distinguish between the interface platforms. Other procedures support functionality which is provided by the interface platform only. The usage of this group of I/O drivers is therefor limited to the particular platform only, like for example, viPeek() and viPoke() for VXI register based instruments.

The VISA I/O drivers access the interface boards regardless their manufacturers. Therefor, instrument drivers performing calls to these drivers are independent upon the vendor supplied products; applications using VISA I/O drivers, can be ported among different environments.

## The VISA Transition Library VTL

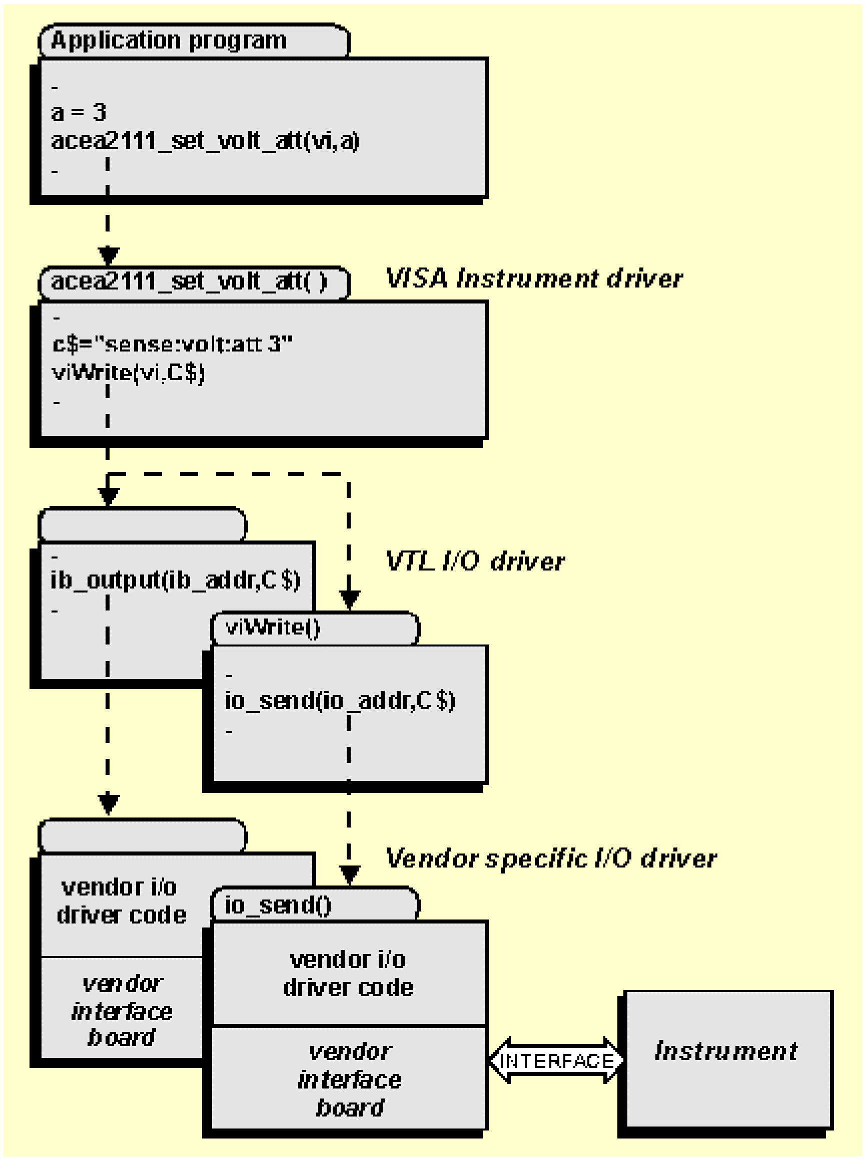

**Figure 3.** Survey of the structure of an instrument driver

Up to now, applications had to use vendor specific I/O drivers. These drivers mutually deferred by name, parameters and functionality. In order to allow VISA I/O drivers to be used on a vendor specific platform, a VISA Transition Library VTL to that platform is supplied. The VTL transfers the VISA I/O driver calls to the vendor specific I/O driver or interface boards direclty.

## Properties of instrument drivers

A property of instrument drivers is that they return information about the operational status. This information can be warnings, error reports or just an "Ok" flag which signals that the driver has properly executed. This property is certainly valuable for a number of applications, which have to be setup quickly and where flexibility is needed. Such features cause a certain programming overhead within the driver design itself, and therefor need more execution time. In order to tune the system to the performance needs of an application, the end user has to modify the instrument driver according to his own desires. If the driver itself uses a properly organized and hierarchical command set and provides a well structured status reporting and error handling mechanism, like IEEE 488.2 and SCPI compatible instruments do, this may not be a difficult job and easily to be done. However, it will become a tedious and time consuming task if a vendor uses the supplied instrument driver to hide the imperfections of the command set or the status and error reporting structure. Therefor, end users have to take care about instrument's remote operating facilities at the command level.

As said before, most instrument vendors will supply drivers for the control of single instrument functions. More complex drivers need to be created by the end users themselves. For that purpose the vendor supplied instrument drivers or utilize the low level VISA I/O drivers could be used.

It is said that the benefit of using instrument drivers is that the application programmer hasn't to worry about the details of the instrument control and does not need to be aware about the instrument command strings. Instead, he has to figure out which instrument driver is to be used to perform the desired control. As can be easily seen from the example given before, instrument driver names differ by manufacturer, model number and function.

For example, an instrument vendor called TMA, may name the driver to set the attenuation for the P4058 model '*tmap4058_config_atten()*', which performs exactly the same functionality as the '*acea2111_set_volt_att()*' driver. Another example is when the company ACEA would introduce a new or another model 2211, and the user wants to replace the existing 2211 model by the new model. In that case, the calls to the '*acea2111*' drivers in his application program are to be replaced by calls to the '*acea2211*' driver, provided they offer identical functionality.

It has to be noticed that the VISA standard does not require drivers to perform identical functionality. This type of standardization is entirely left to the instrument vendors. Therefor it may occur also that drivers having similar names, perform slightly different functionality.

For example, the *acea2211_set_volt_att()* driver may set the DC attenuation, whereas the *acea2111_set_volt_att()* driver relates to the AC attenuation, just because model 2111 is an AC meter.

This comes to the conclusion that, although instrument drivers can have identical names, they may perform a different functions. This is an observation, which may not be considered as a criticism to the VPP consortium. This is merely a consortium of manufacturers of computer and interface platform related products. Therefor, the scope and objective of the VISA standard is limited to provide compatibility among the computer hard and software platforms.

Achieving compatibility among instrument functionality is a responsibility of instrument manufacturers. For that purpose they established a consortium and created the SCPI standard. Only using both the SCPI and VISA standard will result in the highest level of compatibility. This is not only valid to end users, creating application programs, but also instrument manufacturers may considerably benefit from the concepts.

# Combining VISA and SCPI

Within application programs, instrument drivers may want to go beyond the complexity level being established by the vendor provided functionality.

A first reason to do so is to gain more precise control over what the provided driver does. He might want to customize the driver and modify to exactly do what he wants the driver to do, diminishing overhead, or extend the standard functionality. For that purpose, the programmer may use the existing vendor provided driver as a template and example of how to program the instrument.

He might also go beyond the level of complexity of the standard provided instrument driver. This is not only valid to create complex test setups for multiple instruments, but may also be needed for more simple purposes, as for example, to resolve possible coupling conflicts as exist in a lot of today's instruments. The next example shows how such a detail, which may look unimportant may have annoying consequences for the design of a properly operating application program

A good example of a device coupling is a data acquisition with a total acquisition time Ts, consisting of Ns samples with an interval time Ts. These parameters are mutually coupled by the equation Ts = Ns x Ts. When either parameter is modified by a remote program message, another parameter has to change accordingly to satisfy the coupling equation. Which parameter changes is entirely device dependent.

Assume a device whose initial settings being set to Ns = 500, Ts = 5s and Ts = 2,5 ms are to be altered to Ts = 1 ms and Ts = 1 s. This devie is such designed that it changes the sample interval time Ts when the total acquisition time Ts is programmed, meanwhile leaving the number of sample Ns unmodified. Similarly, when Ns is programmed, Ts is changed (Ts unmodified) and when Ts is programmed, Ts is changed (Ns unmodified).

If first the total interval time Ts is programmed to 1 ms and next the sample interval time Ts to 1 s, the final result would be Ns = 500, Ts = 1 s and Ts = 0,5 ms. If the programming would occur in the reverse order, the result would be Ns = 500, Ts = 2 s and Ts = 1 ms.

This clearly shows that the desired result cannot be simply achieved by instrument drivers performing a single control command. An unambiguous solution is neither possible with another design of the coupling relations. A known solution to the coupling problem is to send both parameters of the coupled functionality within a single terminated program messages, making use of the hierarchical syntax structure of the IEEE 488.2 standard. For that purpose, the programmer has to create an instrument driver which handles both parameters within one command string; furthermore, the instrument being controlled needs to be IEEE 488.2 compatibility.

A very good reason to go beyond the scope of a standard VISA driver is to provide an instrument driver set which does not only provide compatibility among the computer hard and software platforms, but also among instrument functionality.

As the examples already did show, VISA instrument drivers do achieve only half of this goal and need to distinguish by vendor specific prefixes. From the view point of an application program, this prefix is redundant. This is because the instrument handle vi, which is passed as a parameter with every instrument driver call, already unambiguously identifies the instrument (manufacturer, model, revision, interface platform, bus address, etc.) Therefor, an application program may want to rely upon a set of instrument drivers which do neither depend upon its environment nor upon the instrument manufacturer.

In order to establish compatibility among instruments, their functionality need to be defined. Identifying common instrument functionality and defining corresponding remote control is exactly what is achieved by the SCPI standard. Therefor, this standard provides an excellent approach to create an instrument driver set which answers the intended purpose. Standard instrument driver functionality can be identified by making use of the precisely defined, remote control functionality of the SCPI commands, whereas the drivers can be named according to the corresponding SCPI command headers.

The following SCPI command illustrates this.
The SENSe:VOLTage:ATTenuation <value> is used to program the voltage attenuation of the sensor part of an data acquisition device. The corresponding generic instrument driver call would be *sens_volt_att(vi,a)*' where the parameter '*vi*' is the instrument handle and '*a*' is the attenuation value. Because SCPI commands are self explanatory, the intended purpose and meaning of the corresponding instrument driver will be easy understandable.

Within the generic SCPI instrument driver, the instrument handle 'vi' is used to identify the instrument to be controlled. Two approaches, as are shown in figure 4, are possible to actually program the instrument:

- the vendor supplied instrument driver may be called. For example:

  *acea2211_sens_volt_att(vi,a)*
  *tmax4058_config_atten(vi,a)*

- the low level VISA I/O driver may be used, ignoring the vendor supplied driver. For example:

  *viWrite(vi,C$)* where C$="SENS:VOLT:ATT 3" for acea model 2211
  *viWrite(vi,C$)* where C$="CONF:ATTEN3" for tma model x4058

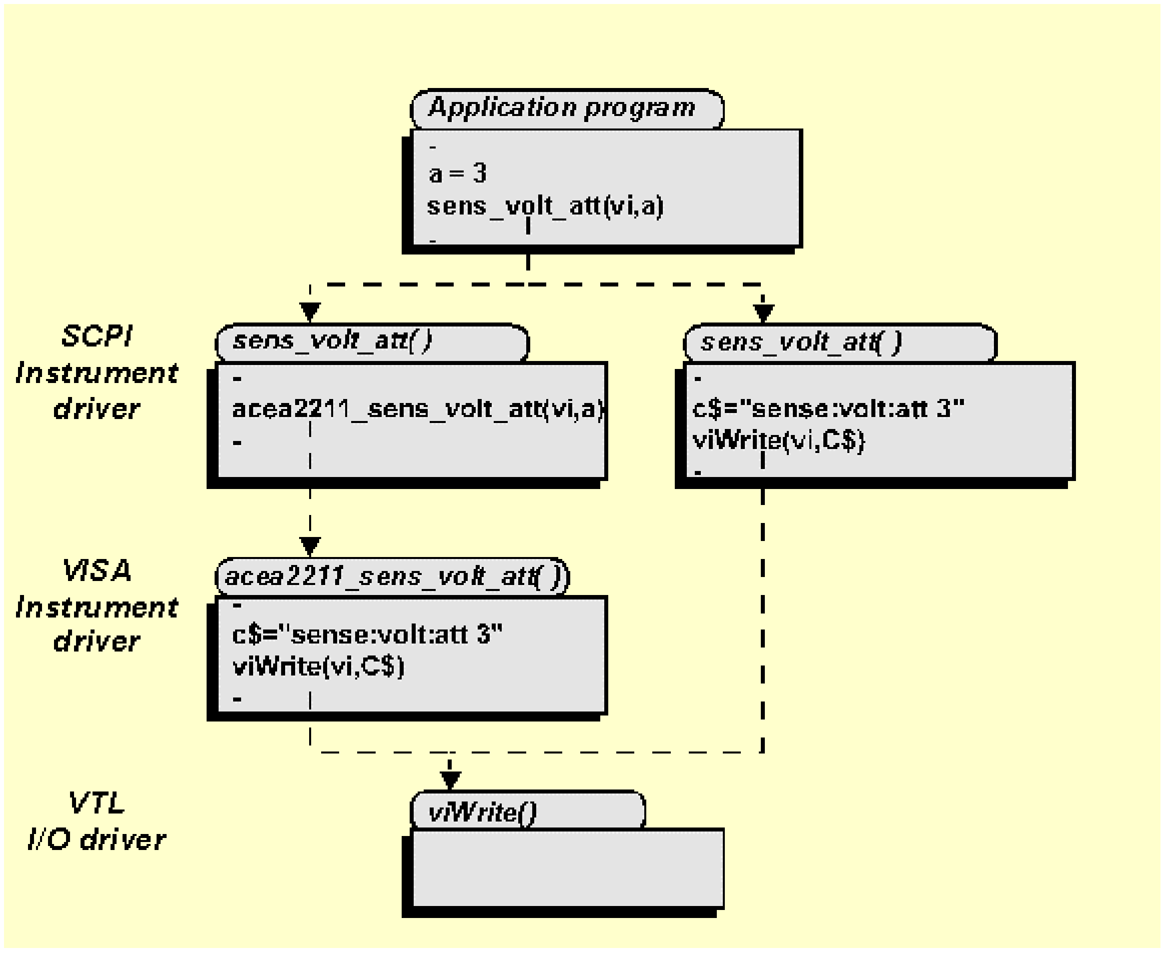

Figure 4. Vendor independent Instrument driver structure

If vendor supplied instrument drivers are used, a status reporting concept is invoked, which monitors for a proper execution and returns error messages and warnings. If this would cause to much overhead, or and optimal performance is desired, the instruments are to be directly programmed by using the VISA I/O drivers. Instruments which are SCPI (and therefor IEEE 488.2) compatible are simply to program and provide a consistent status reporting system which is easy to handle.

For vendors of SCPI compatible instruments, the definition and creation of instrument drivers for their SCPI compatible instruments can easily be achieved. No additional driver definition process is needed; just the manufacturer prefix is to precede the global SCPI instrument driver name.

For example, to program the voltage attenuation of the data acquisition sensor of the acea model 2211:

| | |
|---|---|
| the command string is: | *sens:volt:att <value>* |
| the vendor supplied instrument driver is: | *acea2211_sens_volt_att(vi,a)* |

the vendor supplied instrument driver is: *acea2211_sens_volt_att(vi,a)*
the SCPI instrument driver is: *sens_volt_att(vi,a)*

## Conclusions

1. VISA instrument drivers guarantee a portability among computer hardware/software environments and interface platforms, but do not provide compatibility among instruments; a 'plug and play' of instrument functionality is not provided
2. Application programmers will certainly benefit from the VISA concept, but:
    - they still have to deal with a wide variety of the VISA instrument driver names, where apart of the prefix, vendors supplied instrument drivers may still use different names to control identical functionality. Instead of worrying about the proper command strings, programmers now have to figure out which instrument driver is to be used to perform the desired function.
    - If a functionality is needed which goes beyond the often limited control of a vendor supplied instrument driver, he still needs to be familiar with all details of the instrument programming commands.
3. SCPI (and IEEE 488.2) compatible instruments accommodate the creation and customizing of instrument drivers and provide convenience to both instrument manufacturer and the users of their products in several aspects:
    - The consistent SCPI programming environment simplifies and reduces the efforts to create instrument drivers.
    - Because the SCPI concept:
        - provides an easy understandable command set
        - guarantees a well defined instrument behavior under all conditions, preventing unexpected instrument behavior.
        - maintains compatibility when new functionality is added,

      it facilitates the creation and maintenance of a higher level of instrument drivers and simplifies customizing vendor supplied instrument drivers to optimally suit the needs of the application

4. The SCPI concept allows for the definition of a instrument drivers set, which:
    - can be ported among several environments and does not depend upon the instrument manufacturer or the vendor supplied instrument driver.
    - reduces multiple ways to control identical functionality
    - is self explanatory to both novice and expert users
5. In addition, instrument manufacturers of SCPI compatible instruments:
    - will benefit because:
        - their instrument command set is simply to map to their (vendor supplied) instrument drivers.
        - the functionality of the VISA required instrument drivers is based

upon the IEEE 488.2 common commands.

- will not only face an increased efficiency in the definition and creation of their instrument drivers set, but similar benefits will be gained in the design and maintenance of the instrument software itself. This is caused by the fact that SCPI:
    - considerably simplifies the definition process of the remote control part of the instruments.
    - allows to be expanded with new commands, meanwhile maintaining backwards compatibility with previously defined commands
    - allows for the development of standard and re-usable software, which can be ported among different types of instruments

---

***About the author***

*Mr. John Pieper was awarded a Master's Degree in Electrical Engineering from the University of Delft, The Netherlands. He has many years of outstanding industry experience and has been involved with the development of many test and measurement instruments and software products for ATE systems.*

*Mr. Pieper participates in several international ATE standardization activities. He was a member of Working Group 3 of IEC/TC65/SC65C (responsible for GPIB and related standards) and is closely involved with the IEEE Instrumentation and Measurement Society, TC8, responsible for IEEE 488.1, IEEE 488.2, and the new RS 232 based serial instrument interface IEEE 1174. He actively cooperates with the sub committee UK951.1 of the German DKE - Deutsche Elektrotechnische Kommission - on interface technologies.*

*Until recently he was the Vice President of the Board of Directors of the SCPI consortium and a member of their Large Technical Committee. He now is the European SCPI contact address.*

*Mr. Pieper is the Managing Director of ACEA, a company residing in Wierden, The Netherlands, providing products, consulting services, educational training and applications for Automatic Test and Measurement systems.*