A UNIFIED INTERFACE FOR SIGNAL-ORIENTED CONTROL OF INSTRUMENTS AND SWITCHES

Stefan Gal, TYX Corporation, (703) 264-1080, stefan@tyx.com
Ion A. Neag, TYX Corporation, (703) 264-1080, ion@tyx.com

ABSTRACT

The signal-based testing paradigm, which reduces the impact of instrument obsolescence, is typically implemented via software architectures that encapsulate instrument-specific code in signal drivers. The paper describes the design of a signal driver interface and an extension of this design that enables the uniform treatment of instruments and switches. The unified interface supports the control of switches found inside instruments, and the modeling of signal transmission capabilities of switches and signal paths. Its use has the potential to lower software development costs by reducing the number of different interfaces in Automatic Test Systems.

Keywords: Automatic Test System, ATS, obsolescence, interchangeability, signal-based testing, IVI Foundation, signal interface

1 INTRODUCTION

In the *signal-based testing paradigm*, test procedures specify testing operations in terms of signals to be applied and measured at the pins of the Unit Under Test (UUT). The Automatic Test System (ATS) software that supports the execution of these test procedures is responsible for mapping signals to instruments or instrument subsystems (“resource allocation”), as well as for calculating the signal paths through the switching subsystem and controlling the appropriate switches in order to open and close the signal paths (“automatic switching”).

The signal-based testing paradigm provides significant benefits for applications where *long-term instrument obsolescence* is a serious concern, such as avionics, weapon systems, transportation and nuclear power plants. Because the test procedure code does not contain instrument commands or instrument driver calls and it does not reference instrument or switch ports directly, this code is independent on the instrumentation used to execute it. When instruments become obsolete, they can be replaced without changing the test procedure code or the fixture hardware. This avoids software re-hosting and validation costs, leading to important savings over the lifetime of the equipment under maintenance [1] [2].

Section 2 of the paper introduces two typical software architectures used for implementing the signal-based testing paradigm, showing that both architectures make use of “signal drivers”, as a means of encapsulating instrument-specific code. Section 3 focuses on the signal driver interface, presenting a design developed by the authors and proposed for standardization in the IVI

Foundation. The extension of the above design for switches and signal connections is described in Section 4.

2 SIGNAL-ORIENTED CONTROL OF AUTOMATIC TEST SYSTEMS

Signal-based ATSs may be implemented using the ATLAS language or a general-purpose programming language in conjunction with a software library [3]. Typical architectural solutions for these implementations are described in the following sub-sections.

2.1 ATLAS-Based ATS

In the ATLAS-based ATS shown in Figure 1, **Test Procedure Source Modules** are processed by an **ATLAS Compiler**, which generates a **Test Procedure Object Module**. The test procedure source code specifies requirements for the signals that it uses, including the range, resolution and precision of signal parameters, as well as the connectivity of signal ports to UUT pins. For each signal, the **Resource Allocator** to identifies an available resource (i.e., instrument or subsystem of an instrument) that satisfies the requirements, as well as the switching paths necessary for connecting the signal. The **Resource Allocator** uses the description of instrument capabilities and connectivity from the **Device Information Store**, switching subsystem connectivity from the **Switch Information Store**, and fixture connectivity from the **Fixture Information Store**, returning information about the allocated instruments and switches. This information is embedded in the **Test Procedure Object Module**.

The **Device Information Store** contains a description of capabilities and connectivity for all instruments and switches available on the system. The following information is typically present for instruments:

- for each instrument, a list of supported signals
- for each signal:
    - signal role (source, sensor, load, etc.)
    - signal type (“DC Signal”, “AC Signal”, etc.)
    - the list of supported signal parameters
    - for each parameter:
        - parameter role (controllable, measurable)
        - range, resolution and/or precision
    - connectivity of signal ports to instrument pins

The connectivity of the **Switching Subsystem** is described in the **Switch Information Store** and the connectivity of the **Fixture** is described in the **Fixture Information Store**, possibly as lists of signal paths.

The **Test Procedure Object Module** is executed under the control of a **Run-Time System**. In response to signal operations specified in the test procedure, the **Run-Time System** delegates the signal control functionality to the appropriate **Signal Driver(s)** and the switching functionality to a **Switching Driver**.

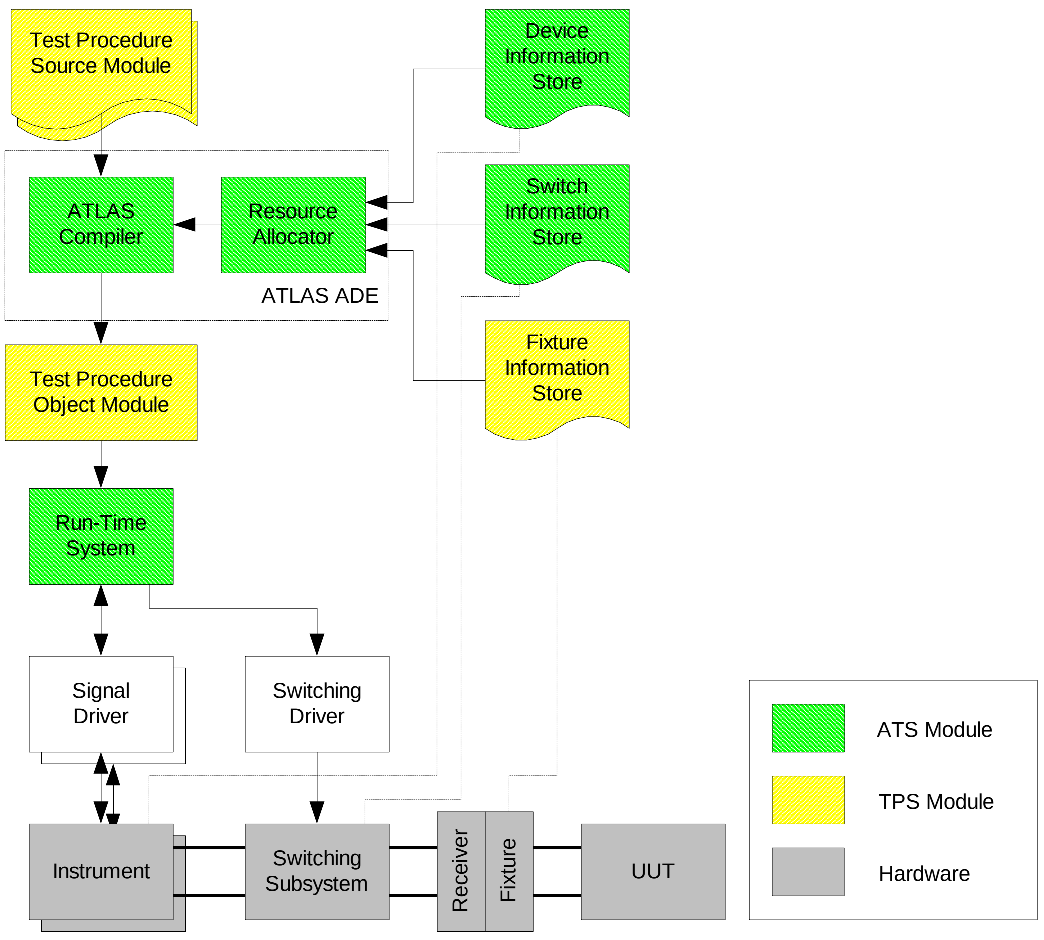

Figure 1. Signal-Based ATS Using the ATLAS Language - Example

The **Signal Drivers** are instrument-specific software modules that expose signal control functions such as Setup, Arm, Fetch, Reset, etc. The **Switching Driver** is typically a software module specific to the switching subsystem, exposing signal path control functions, typically Connect and Disconnect. **Signal Drivers** and the **Switching Driver** control the hardware via SCPI commands, instrument driver calls or other instrument control mechanisms.

2.2 Signal-Based ATS Using a General-Purpose Programming Language

The ATS architecture shown in Figure 2 uses a commercial **Application Development Environment** (ADE) to compile **Test Procedure Source Modules** developed in a general-purpose programming language. The signal control operations are specified in the test procedure code by using objects from a **Signal Library**. This library contains classes or components for various signal roles and types, such as “Source_AC_Signal”, “Sensor_DC_Signal”, etc.

The **Test Procedure Object Module** is executed under the control of a **Run-Time Engine**. When a signal object is needed, the test procedure code calls a function of the **Run-Time Engine**, passing a set of requirements for the signal. The **Run-Time Engine** requests the

Resource Allocator to identify an available resource that satisfies the above requirements, as well as the switching paths necessary for connecting the signal. The **Resource Allocator** uses the description of instrument, switch and fixture capabilities and connectivity from the **Device Information Store**, the **Switch Information Store** and the **Fixture Information Store**. Information about the allocated instruments and switches is passed back to the **Run-Time Engine**, which returns a signal object to the test procedure code.

In response to signal operations specified in the test procedure, the signal objects delegate the signal control functionality to the appropriate **Signal Driver(s)** and the switching functionality to the **Switching Driver**.

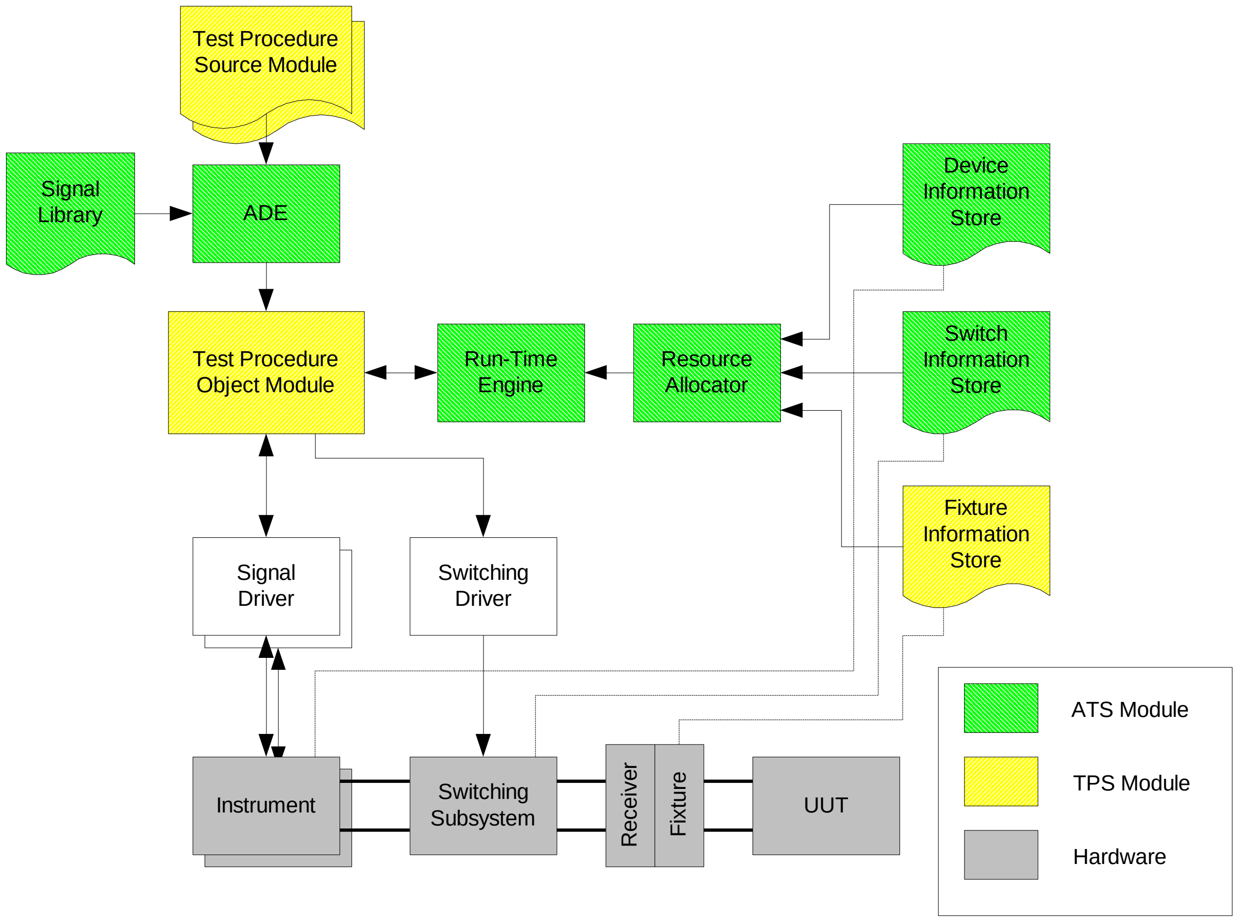

Figure 2. Signal-Based ATS Using a General-Purpose Programming Language - Example

3 THE IVI SIGNAL INTERFACE

The *standardization of a signal driver interface* enables the portability of such drivers between the signal-based test environments of different vendors, cutting development costs for ATSs that are required to support multiple test environments. Moreover, the portability of signal drivers is expected to provide multiple customers for their developers, leading to higher initial quality and better maintenance.

The IVI Foundation [4] has formed a working group that develops a standard for signal driver interfaces [5]. One task of that group is to develop an interface design that supports the usage scenarios identified in Section 2, while enabling the use of signal drivers from a multitude of test environments and programming languages. The design issues relevant for the subject of the present paper are briefly described in the following. The description does not cover specialized topics such as signal synchronization or inter-dependent instrument capabilities. While the working group currently considers both COM and C interfaces, the following description uses for illustration purposes only the COM interface.

3.1.1 Interfaces for Signal Control and Signal Capability Description

The signal-based ATS architectures described in Section 2 include a generic data storage module named Device Information Store, which contains the description of signal generation and measurement capabilities for each instrument available on the system. On the other hand, both architectures contain one signal driver for each controlled instrument.

In order to simplify the distribution, integration and maintenance of signal drivers, it is desirable to pack the capability information specific to each instrument along with the signal driver controlling that instrument. In this situation, the Device Information Store would contain, for each instrument, a simple reference to the signal driver component (in the case of IVI Signal Drivers, this information is present in the IVI Configuration Store). Resource allocators use the reference to instantiate the signal driver component and retrieve capability information via its software interface.

In principle, the capability information may be modeled using a specialized language or an object model. The second approach is currently considered for the IVI-SI design, in order to provide consistency across IVI components.

Figure 3 shows the top-level interfaces exposed by a COM IVI Signal Driver (IVI-SD).

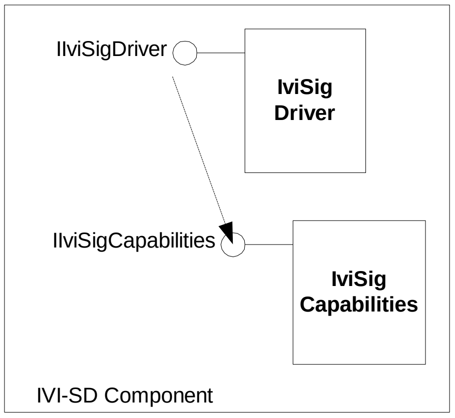

Figure 3. Top-Level Interfaces of an IVI Signal Driver

The `IIviSigDriver` interface is the default interface of the component and provides access to the signal control functionality. A property of this interface returns a pointer to the `IIviSigCapabilities` interface of an **`IviSigCapabilities`** component, which in turns provides access to a hierarchy of COM components that model the capabilities of the instrument.

3.1.2 Signal Control Interfaces

Numerous instruments are able to generate and/or measure simultaneously multiple signals. In the IVI-SI design, the control of these signals is achieved via multiple **`IviSigSignal`** components, belonging to the same IVI-SD. The interfaces of these components are accessible via properties exposed by the `IIviSigDriver` interface, as shown in Figure 4. Because different signal roles require different control functions, interfaces specialized for each role are derived from `IIviSigSignal`. For example, the `IIviSigSource` interface contains the methods `Setup()` and `Reset()`, while `IIviSigSensor` adds `Arm()` and `Fetch()`.

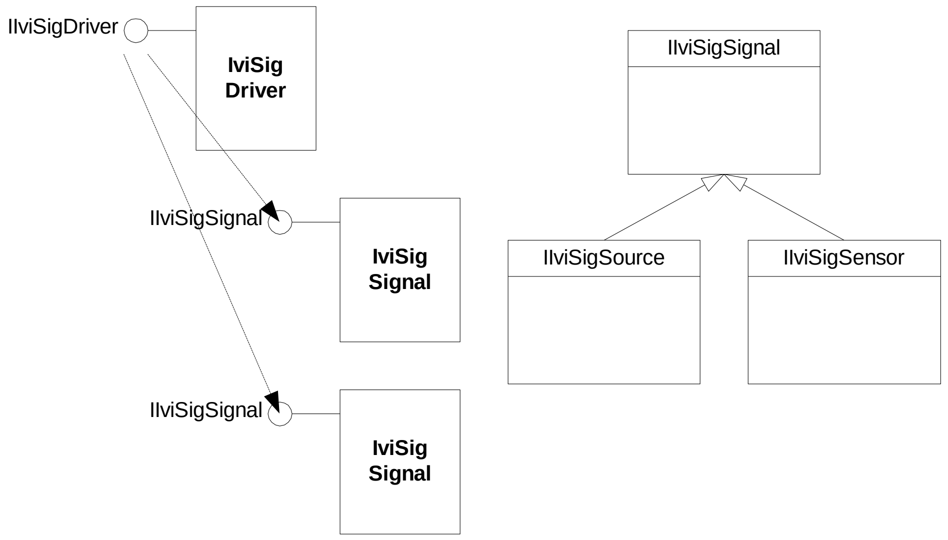

Figure 4. Signal Control Interfaces

3.1.3 Signal Capability Model

The `IIviSigCapabilities` interface introduced in Section 3.1.1 provides access to a structure of objects, implemented as COM components, which describe the signal capabilities of the instrument. A simplified representation of this structure is shown in Figure 5.

An instrument is represented by a **`Device`** object. This object contains a **`Subsystems`** collection, where each **`Subsystem`** object represents a physical module of the instrument that is exclusively used for generating or sensing a signal (i.e., once allocated to a signal, may not be allocated to a concurrent signal). The **`Device`** object also contains a **`Resources`** collection, where each **`Resource`** object represents the capability to generate, measure or monitor <u>one</u>

signal. The **Resource** object has properties that describe the signal role (source, sensor or event monitor) and the signal type. It also contains a **SubsystemRefs** collection, where each **SubsystemRef** object represents a reference to a device subsystem, indicating a physical module of the instrument that is used for generating or sensing the signal modeled by the **Resource** object.

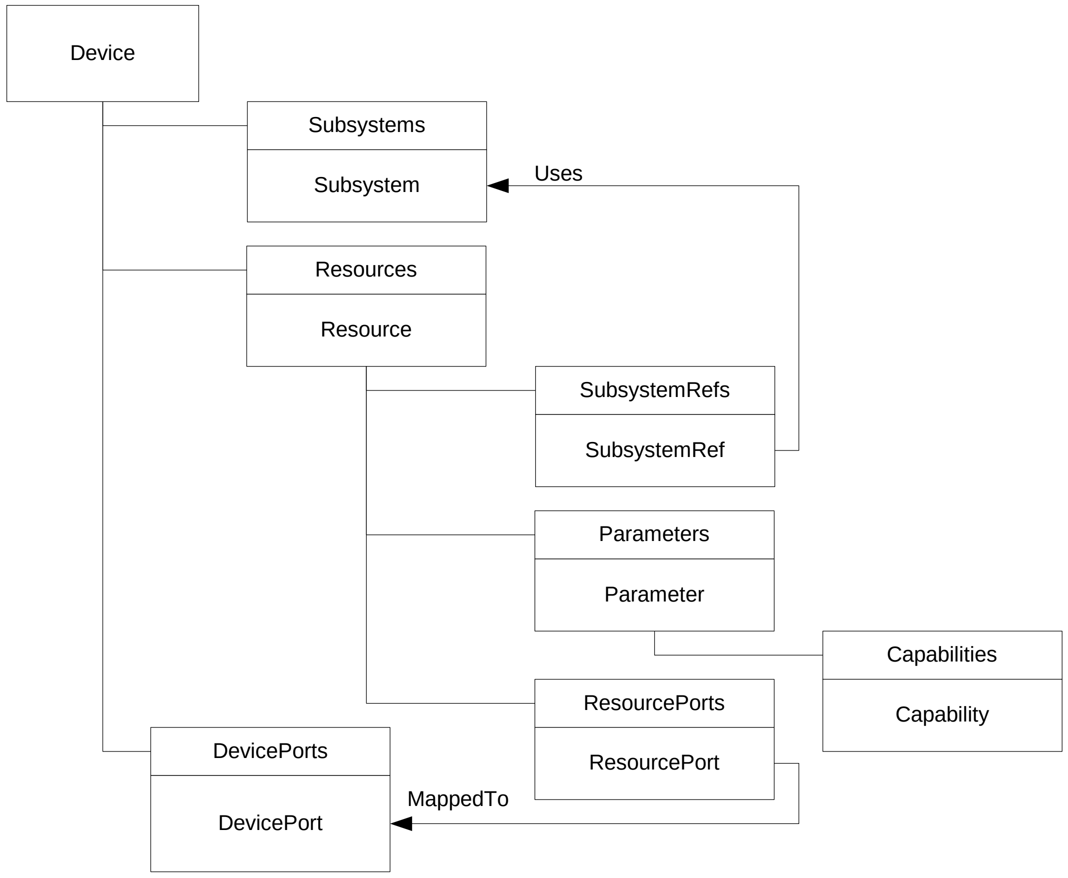

Figure 5. Signal Capability Model (simplified)

Each **Resource** object contains a **Parameters** collection, where each **Parameter** object represents the capabilities related to one parameter of the signal type specified for the resource (for example, a signal of type "AC Signal" has the parameters "Voltage", "Frequency", DC Offset", etc.). The **Parameter** object has properties that describe the parameter role (measurable, controllable) and its unit.

Each **Parameter** object contains a **Capabilities** collection, where each **Capability** object represents an alternative set of capabilities for the given parameter. The **Capability** object has properties for value, range, resolution and precision.

The **Device** object also contains a **DevicePorts** collection, where each **DevicePort** object represents a port of the particular instrument model (with the mapping of instrument model ports to the port of a particular instrument installed on the system being specified in the IVI Configuration Store [6]). Each **`Resource`** object contains a **`ResourcePorts`** collection, where each **`ResourcePort`** object represents a port of the signal type specified for the resource (for example, a signal of type "AC Signal" has the ports "HI" and "LO"). A property of the **`ResourcePort`** object, "MappedTo", references the **`DevicePort`** object representing the instrument model port where the signal port is mapped, when the signal is generated or sensed.

4 UNIFIED TREATMENT OF INSTRUMENTS AND SWITCHES IN THE IVI SIGNAL INTERFACE

The generic ATS architectures introduced in Section 2 handle the control and modeling of capabilities for the switching subsystem in separation from the control and modeling of capabilities for instruments.

During the design of the IVI Signal Interface, the authors have discovered that both the control interface and the capability model originally developed for instruments may be extended to handle switching. This allows the use of IVI Signal Drivers for controlling the switching subsystem and for providing a description of capabilities for the modules that compose this subsystem. In this scenario, the lower part of the ATS block diagram introduced in Figure 1 may be redrawn as shown in Figure 6. A similar transformation is possible for the block diagram represented in Figure 2.

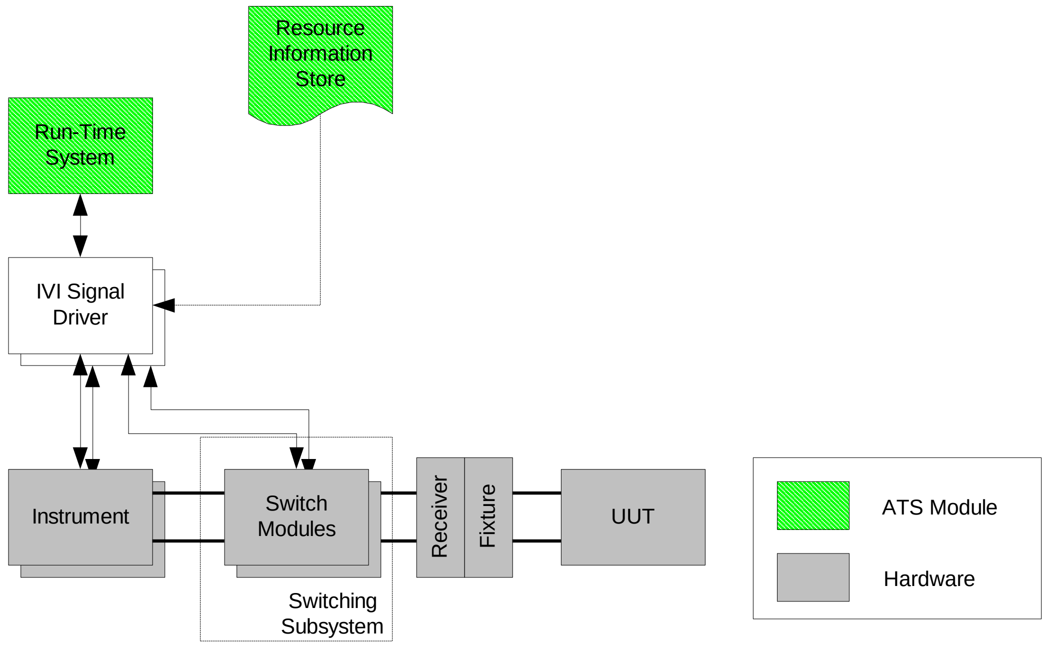

Figure 6. Signal-Based ATS Using a Unified Control Interface (Partial Block Diagram)

The **Resource Information Store** contains references to the IVI Signal Drivers that control instruments and switch modules. The drivers provide information regarding the capabilities and connectivity of instruments and switches.

In the proposed approach, the modeling of capabilities for the switching subsystem may cover, besides basic connectivity, signal transmission capabilities such as: maximum current, maximum voltage, impedance and bandwidth.

Furthermore, the proposed model supports the control and modeling of capabilities for switches that are sometimes present inside instruments. Because of this, the resource allocator can treat the instrument switches and the switches of the switching subsystem in a uniform manner. In other words, all the hardware of the test station is modeled as a mix of instruments, switches and connections, which are no longer separated into subsystems.

4.1.1 Switch Control Interface

Because the functions required for controlling signal paths are different from those used for controlling the generation and measurement of signals, a new interface, `IIviSigSwitch`, must be defined. This interface contains the methods `Connect()`, used to connect one or more signal paths, and `Disconnect()`, which disconnects one or more signal paths.

The **`IviSigSwitch`** components that expose this interface control switching module subsystems, in the same way **`IviSigSignal`** components control instrument subsystems. Consequently, the **`IviSigSwitch`** components are also accessed via the `IIviSigDriver` interface (see Figure 4). Figure 7 shows a simple example, where the IVI-SD that controls a DC power supply contains an **`IviSigSource`** component that controls the generation circuit and an **`IviSigSwitch`** component that controls the switches connecting the generation circuit to the instrument ports.

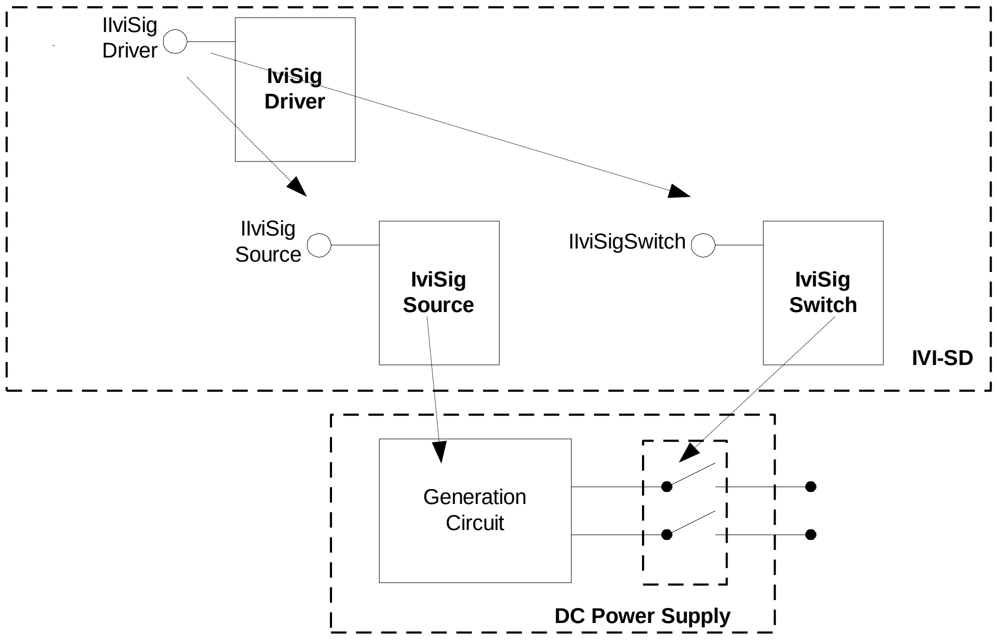

Figure 7. Switch Control Interface - Example for a DC Power Supply

4.1.2 Switch Capability Model

The following design requirements have been identified for the extension of the capability model described in Section 3.1.3 to switching:

1. support the description of *signal connectivity* and *signal transmission capabilities*
2. support the modeling of *switches that belong to the switching subsystem* and *switches inside instruments*
3. support the modeling of *hardwired signal connections* inside instruments, in the switching subsystem and in other hardware subsystems (for example in the fixture), as well as between such subsystems

4.1.2.1 Modeling Signal Transmission Capabilities of Switches and Connections

The signal transmission capabilities of switches and connections may be used by resource allocators when selecting an instrument for a particular signal. In such a usage scenario, the test procedure code would specify, besides signal requirements, a set of requirements for the signal paths used to connect the signal to the UUT pins.

To be usable by resource allocators, the capabilities of switches and connections must be expressed in signal terms, i.e. they should be specified as values or ranges for some signal parameters. The modeling approach used in the IVI-SI design defines signal parameters only in the context of a signal type. The signal types that model generation and sensing capabilities (such as “DC Signal” or “AC Signal”) are not usable for signal transmission capabilities, because a switch or connector can carry a multitude of such signal types. Consequently, a set of specialized signal types has to be defined. Several examples are shown below:

- “Electrical”, with parameters “Max Current”, “Max Voltage”, “Max Power”, “Bandwidth”, “Impedance”, “Loss”, etc.
- “Fluid”, with parameters “Fluid Types”, “Max Pressure”, “Max Mass Flow”, “Max Volume Flow”, etc.
- “Optical”
- “Microwave”, etc.

The object structure introduced in Figure 5 for instrument capabilities may be also used for modeling the signal transmission capabilities of switches, as demonstrated below. The **`Device`** object may be used models a switching module (typically, a hardware asset with a unique bus address). **`Resource`** objects may be used to to model switches inside switching modules. This requires the addition of a new signal role, “switch”. **`Parameter`** and **`Capability`** objects may be used to describe the signal transmission capabilities of the switch, using the specialized signal types introduced before.

4.1.2.2 Modeling the Connectivity of Switches

The object model developed in Section 3.1.3 will be extended for modeling the connectivity of switches (i.e., the set of possible signal paths, along with concurrency information). Because **`Resource`** objects model switches, each switch port is represented by a **`ResourcePort`** object. A property of this object returns a pointer to the **`DevicePort`** object representing the switch module port to which the switch port is connected.

In addition to the external connectivity of switches, the model must describe their internal connectivity, i.e. the set of possible signal paths through the switch. Commonly used switch topologies such a "multiplexer" and "matrix" may be identified using a specialized property of the corresponding **`Resource`** object.

Exclusive reliance on a predefined set of switch topologies should be avoided, because it compromises extensibility. Consequently, the model must be able to describe in a generic format a set of possible paths between the switch ports. This description must identify the paths that are concurrent (e.g. for ganged switches), as well as the paths that are exclusive (e.g. for multiplexers). A design that achieves these objectives is described by the partial object diagram shown in Figure 8.

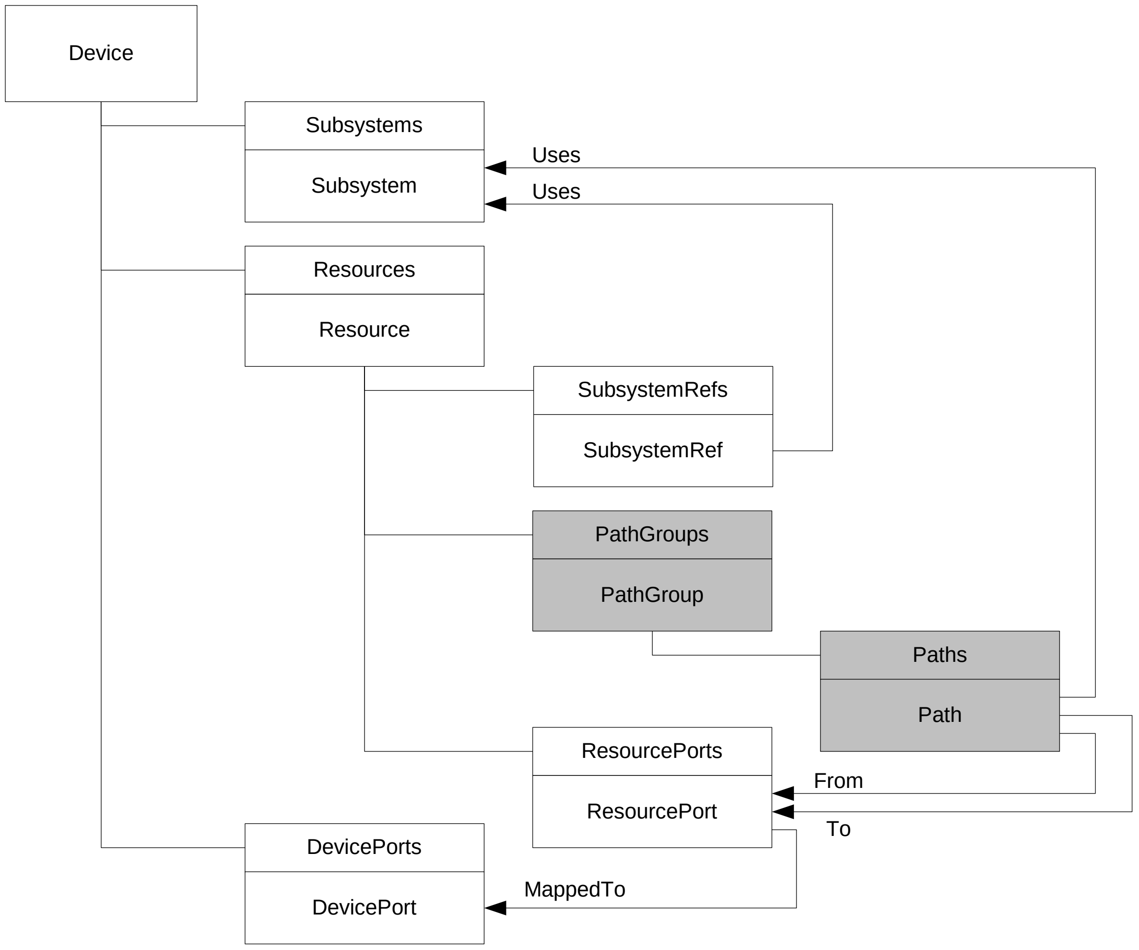

Figure 8. Signal Capability Model, Including Switch Capabilities (simplified)

Each **`Path`** object represents a path between two switch ports. These ports are indicated by two properties of the **`Path`** object, "From" and "To", which reference **`ResourcePorts`** objects.

Another property of the **`Path`** object, “Uses”, references a **`Subsystem`** object, indicating the physical switch used by the path. A physical switch may be used at any given time by a single switch path. This information is necessary for the automatic calculation of signal paths by resource allocators. For example, in the case of a multiplexer all signal paths use the same 1-to-n physical switch, while for a matrix each path uses a different row-column physical switch.

Paths within a switch that may exist only simultaneously (e.g. in the case of ganged switches) are grouped in the same **`Paths`** collection below a **`PathGroup`** object. This information is also necessary for the automatic calculation of signal paths. For example, in the case of a ganged multiplexer all signal paths that exist for a given position of the 1-to-n switches belong to a path group.

4.1.2.3 Modeling Signal Connections

The first paragraph of Section 4.1.2 identifies a set of requirements for modeling hardwired signal connections. While the model designed for the IVI-SI meets all these requirements, the description provided below will use, for illustration purposes, only *signal connections inside instruments*.

As described in Section 3.1.3, **`DevicePort`** objects were introduced for modeling external instrument ports. In the proposed extension, these object are also used for modeling the ports of instrument subsystems, which are internal to the instrument. Such a subsystem is capable of generating or sensing one or more signals, each of them represented by a **`Resource`** object. The **`ResourcePort`** objects below the **`Resource`** object reference **`DevicePort`** objects, indicating the mapping of signal ports to subsystem ports.

Additional **`Resource`** objects are used for representing connections between subsystem ports and/or between subsystem ports and external instrument port. These objects have a special signal role, “connection”. As indicated in Section 4.1.2.1, the signal transmission capabilities of connections are modeled using **`Parameter`** and **`Capability`** objects below **`Resource`**. If such a **`Resource`** object represents one connection, it has a single **`PathGroup`** object, which in turn has a single **`Path`** object, whose properties “From” and “To” reference the subsystem/instrument ports linked by the connection. One **`Resource`** object may represent multiple connections, in which case its single **`PathGroup`** object has multiple **`Path`** objects (one for each connection). This approach may be used for representing multiple connections that have identical signal transmission capabilities.

For example, an object model for the DC Power supply represented in Figure 9 contains three **`Resource`** objects representing the following hardware elements:

- Generation Circuit: role = source, signal type = “DC Signal”
- Switches: role = switch, signal type = “Electrical”
- Connections: role = connection, type = “Electrical”

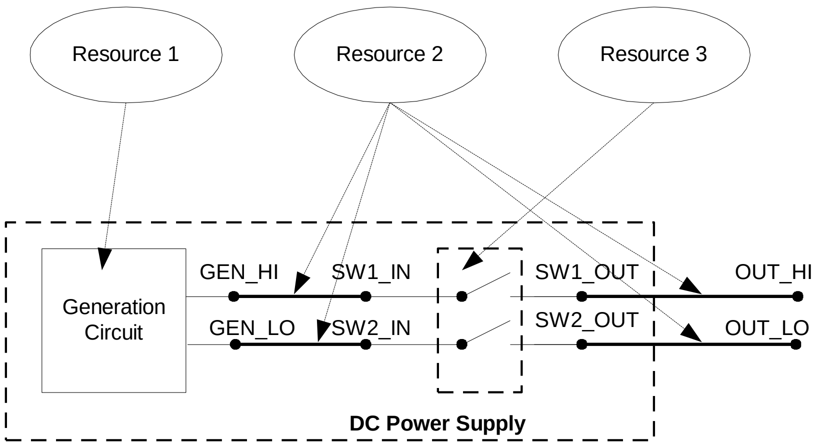

Figure 9. Switch Capability Model - Example for a DC Power Supply

5 CONCLUSION

The paper describes a signal-based driver interface design supporting the control and the modeling of capabilities for both instruments and switches. The unified design provides the following benefits:

1. enables the control and modeling of capabilities for switches inside instruments

2. enables the modeling of signal transmission capabilities such as maximum current, impedance and bandwidth, for both switches and hardwired signal connections

3. simplifies the implementation of ATS software, by using a single control interface and a single capability model for all hardware assets

4. reduces development costs for switching drivers, by avoiding the learning overhead associated with different interfaces and different capability models

5. extends the benefits of standardization to drivers that control the switching subsystem

6 REFERENCES

[1] Oblad, R., “Achieving Robust Interchangeability of Test Assets in ATE Systems”, *Proc. AUTOTESTCON*, San Antonio, TX, 1999, pp. 687-698

[2] Ramachandran, N., Oblad, R.P., Neag, I.A., Tyler, D.F., “The Role of the IVI Signal Interface Standard in Supporting Instrument Interchangeability”, *Proc. AUTOTESTCON*, Anaheim, CA, 2000

[3] Neag, Ion A., Ramachandran, N, “ATLAS2K and the IVI Signal Interface - the Framework for an Open, Modular and Distributed ATS Architecture”, *Proc. AUTOTESTCON*, Valley Forge, PA, 2001, pp. 23 - 37

[4] *** *IVI Foundation Web Site*, http://www.ivifoundation.org

[5] *** *IVI Signal Interface Working Group*, IVI Foundation, http://www.ivifoundation.org/groups/Signal-Interfaces/default.htm

[6] *** *IVI-3.5: Configuration Server Specification*, Revision 1.0, IVI Foundation, 2002