ATLAS2K AND THE IVI SIGNAL INTERFACE - THE FRAMEWORK FOR AN OPEN, MODULAR AND DISTRIBUTED ATS ARCHITECTURE

Dr. Ion A. Neag, TYX Corporation, (703) 264-1080, ion@tyx.com
Narayanan Ramachandran, TYX Corporation, (703) 264-1080, nr@tyx.com

ABSTRACT

This paper investigates the combined use of ATLAS2K and the IVI Signal Interface in ATS architectures. The analysis of these initiatives identifies a set of common features and demonstrates that they address complementary areas of ATS architectures. This situation makes them an excellent fit for establishing the foundation of a modular, open, distributed and coherent COM-based ATS architecture.

Keywords: automatic test system, ATLAS, ATLAS2K, Interchangeable Virtual Instruments, IVI, signal, signal interface

1 INTRODUCTION

The analysis of software technologies used in PC-based Automatic Test Systems (ATSs) demonstrates a gradual *evolution* from the *procedural programming paradigm*, relying on C-type languages and Windows Dynamic Link Library (DLL) technology, to the *object-oriented paradigm* that builds on object-oriented languages and the Component Object Model (COM) technology. This trend is visible, for example, in the work of the IVI Foundation [1], where C interfaces for instrument drivers, targeted towards the current user base, are complemented by COM interfaces, supporting the development of new projects in object-oriented environments [2].

An essential feature of the object-oriented paradigm is its ability to support the implementation of computer-based solutions for very complex problems. Consequently, *object and component libraries* have emerged as modern alternatives in application fields where domain-specific languages were previously the preferred approach. Such libraries may be used in conjunction with general-purpose programming languages, which are supported by high-quality development tools and offer better availability of programming expertise than specialized languages. IVI Drivers [3] provide an excellent example for the above trend, since their functionality and benefits (i.e., perform instrument control while supporting instrument interchangeability) are very similar to those of the SCPI language.

The ATLAS2K standard [4] defines a *COM library* oriented towards *signal-based testing*. Thus, it is a perfect illustration of both trends described above.

Recognizing that IVI drivers alone do not provide complete interchangeability for instruments, the IVI Foundation develops the *IVI Measurement and Stimulus Subsystems (IVI-MSS) standard*

[5], which defines an enhanced architecture where additional layers of COM components compensate for differences in instrument behavior and support the aggregation of multiple instruments [6] [7].

A recent initiative in the IVI Foundation, the IVI Signal Interface [8], aims to support the signal-oriented control of instruments while providing a high degree of instrument interchangeability [9] [10]. This goal may be achieved by standardizing a set of interfaces for IVI-MSS components, interfaces that support signal-oriented operations such as Reset, Setup and Fetch.

2 ATLAS2K

2.1 Overview

ATLAS2K is a new standard under development by the Test Description (TD) Sub-Committee of the IEEE Standards Coordinating Committee 20 on Test and Diagnosis for Electronic Systems (SCC20) [11].

This standard represents a significant departure from previous ATLAS specifications, which define a high-order test requirement-oriented language. Instead, ATLAS2K defines a set of *COM components that model signals*, as well as a way to *interconnect these components in order to specify test requirements*. The above components may be used from "carrier languages" that satisfy a set of minimal requirements defined in the standard [11], with C++, Visual Basic and Java being among candidates.

The definition of ATLAS2K components includes the specification of *syntax*, through an Interface Definition Language (IDL), and of *semantics,* using the Signal and Method Modeling Language (SMML) [12].

Extensibility, a weak point of previous ATLAS specifications, is addressed in the new standard by defining a set of *primitive signals* ("ATLAS2K Basic Components") and supporting the definition of new signals through *composition* operations [11]. *Compatibility with previous ATLAS standards* is supported in ATLAS2K by defining a component library implementing signals that correspond to ATLAS NOUNs defined by previous standards [13].

2.2 ATLAS2K Signal Components

2.2.1 Base Classes

ATLAS2K defines a set of base classes that model:

- signal roles:
    - Source
    - Sensor
    - Signal Conditioner (implementing signal transformation and combination)
    - Event Conditioners: time-based event, event-based event, signal-based event

- Unit Under Test (UUT) pins
- values of physical quantities

The fundamental base class, named **SignalFunction**, models a generic signal role. A symbolic representation of this class is shown in **Figure 1**.

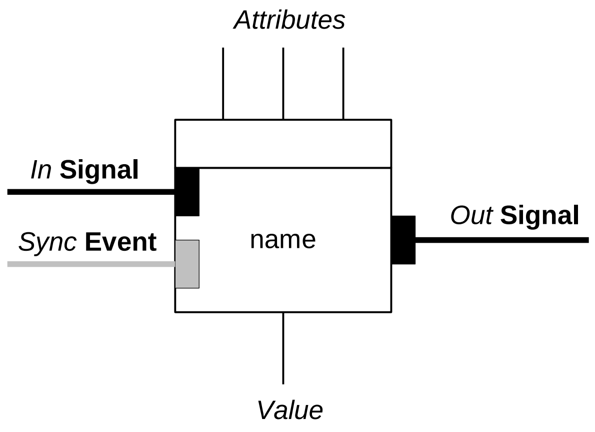

Figure 1. SignalFunction Class

The default interface of the **SignalFunction** class contains the following properties:

- *In*: reference to a **Signal** interface belonging to another object
- *Sync*: reference to an **Event** interface belonging to another object
- *Out*: reference to a **Signal** interface belonging to the current object; a reference to the *Out* interface may be assigned to an *In* or *Sync* property of another object (see **Figure 2** below for an example)
- *Attributes*: model the controllable parameters of the signal, such as the amplitude and frequency of a sinusoidal voltage
- *Value*: models the value of the measurement; exists only for the "sensor" role

Signal and **Event** are specializations of a base interface named **SignalFlow**, which contains the following elements:

- properties:
  - *State*: returns the current state of the object; the following states are defined: "Stopped"; "Paused" and "Running"; the current state is changed by the methods specified below and possibly by events the object receives
- methods:
  - *Stop*
  - *Run*
  - *Change*

The **SignalFunction** class provides **SignalEvent** callback interfaces, which support the transmission of events indicating state changes. Such events may be received from objects connected at *In* and *Sync* inputs and may be sent to objects connected to the *Out* output. For instance, a *Ready* event indicates that signal state has changed from “Stopped” to “Paused”, while an *Active* event signals a state change from “Paused” to “Running”.

The semantics of **SignalFlow** methods are signal role-specific; for example, the following actions are performed by the *Run* method of a source signal object:

- call *Run* on all input signals
- acquire resources
- prepare signal for generation
- generate a *Ready* event
- enter “Paused” state
- if the signal is not externally synchronized, also perform the following tasks:
    - activate the physical signal
    - generate an *Active* event
    - enter “Running” state

Actions are also triggered by events received from objects connected to *In* and *Sync* inputs. For example, an *Active* event received from the *Sync* input is used for external synchronization. For a sensor signal in “Paused” state, it triggers a measurement operation.

Composite signals may be modeled by interconnecting multiple **SignalFunction** objects through their interfaces. For example, the generation of an AM signal by combining two sinusoidal signals is modeled by the object structure represented in **Figure 2**.

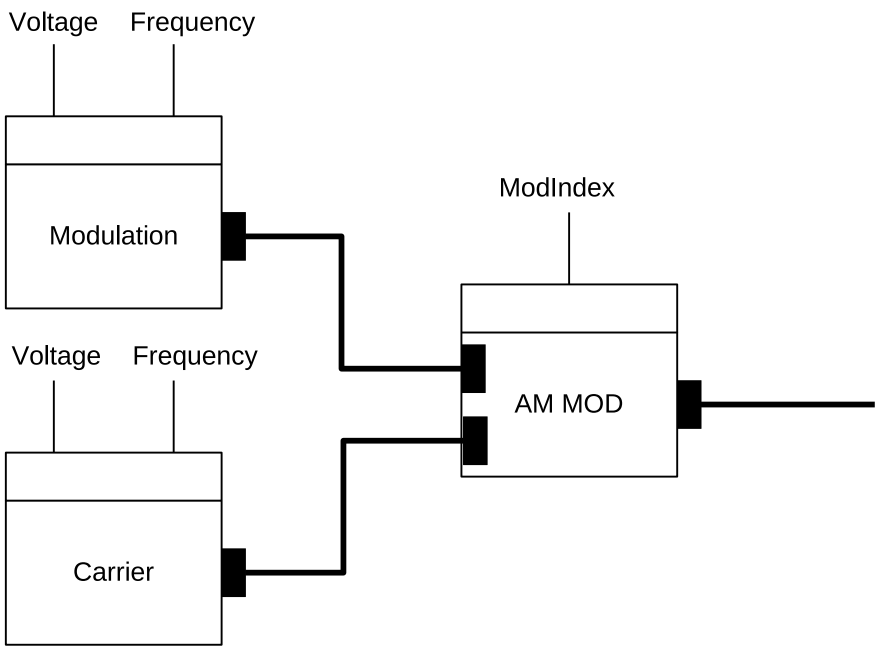

Figure 2. Modeling a Composite Signal

The operation of multiple objects is synchronized through automatic method calls and events. For example, when a client application calls the *Run* method of the **AmMod** object, this object calls the same method on **Modulation** and **Carrier**. When the state of the **Carrier** object changes (for example due to hardware triggering), it informs the **AmMod** object by sending an event.

Specializations of the **SignalFunction** class are defined for particular signal roles such as source, sensor, different types of event conditioners, etc. These derived classes have slightly different interfaces (for example, the **Source** class does not have an *In* property).

2.2.2 Basic Components

The ATLAS2K Basic Components are further specializations of the role-specific classes described above. For example, the **SinusoidVoltage** component, which models a sinusoidal voltage, is a specialization of **Source**. In addition to the properties of **Source**, the default interface of **SinusoidVoltage** contains the following properties that model signal parameters:

- *Amp*, of type <Voltage>
- *Freq*, of type <Frequency>
- *Phase_angle*, of type <PlaneAngle>

ATLAS2K defines such basic components for the following types of signals:

- source signals:
    - non-periodic: constant, single ramp, step, stepped pulse train, trapezoid, trapezoidal pulse train, exponential, damped sinusoid, random, noise
    - periodic: square wave, periodic pulse train, ramp, triangular, trapezoidal, periodic trapezoidal train, sinusoid, arbitrary waveform
- sensor signals: instantaneous, peak-to-peak, RMS, average, sampling, time interval
- signal conditioners: delay, sum, product, inverse, exponential, filters, Fourier transform, etc.
- parallel digital signals
- serial data buses: RS, ARINC 429 and 629, MIL-STD-1553B, CSDB

2.2.3 TTF Components

Complex signals may be described using basic signals and the signal composition methodology previously described. Such complex signals are grouped in libraries named “Test Technology Frameworks” (TTFs). TTF components may be defined using ATLAS2K Basic Components, as well as components from other TTFs.

Domain-specific TTFs are expected to be developed for applications such as RF testing, video testing, etc.

In addition, the ATLAS2K standard will include a TTF that describes signals defined in IEEE Std. 716 (C/ATLAS) in terms of ATLAS2K models. This approach is intended to provide backwards compatibility for existing ATLAS Test Program Sets (TPSs).

2.3 Examples

The following Visual Basic code exemplifies typical sequences of signal operations [1]. The equivalent ATLAS statements are provided as comments.

2.3.1 Signal Generation

The following code applies a sinusoidal voltage with an amplitude of 0.5 V and a frequency of 1 MHz to UUT pins “PL1-1” and “PL1-2”.

```
‘ SETUP SinusoidalVoltage Amp 0.5V, Freq 10MHz AS mySig
Dim mySig As Source
Set mySig = A2k.Require(”SinusoidalVoltage”)
mySig.Amp.Units = V
mySig.Amp = 0.5
mySig.Freq = ”1MHz”      ’alternative string format

‘ CONNECT mySig CNX HI ”PL1-1” LO ”PL1-2”
Set cnx = A2k.Require(”TwoWire”)
cnx.HI = ”PL1-1”
cnx.LO = ”PL1-2”
Set cnx.in = MySig.out
Set cnx = Nothing
mySig.out.Run       ’generate physical signal

‘ DISCONNECT mySig
mySig.out.Stop      ’stop signal generation
mySig.in = Nothing
mySig = Nothing
```

2.3.2 Signal Measurement

The following code measures the RMS amplitude of an AC voltage with a nominal amplitude of 0.5 V at UUT pins “PL1-1” and “PL1-2”.

```
‘ SETUP RMSVoltage Nominal 0.5V AS mySig
Dim mySig As MeasurementFunction
Set mySig = A2k.Require(”RMSVoltage”)
mySig.Nominal.Units = V
mySig.Nominal = 0.5

‘ CONNECT (mySig) CNX HI ”PL1-1” LO ”PL1-2”
Set cnx = A2k.Require(”TwoWire”)
cnx.HI = ”PL1-1”
cnx.LO = ”PL1-2”
Set mySig.in = cnx.out
Set cnx = Nothing
```

```
' ARM (mySig)
mySig.out.Run       'perform measurement

' FETCH (mySig) INTO rmsValue
rmsValue = mySig.Value

' DISCONNECT (mySig)
Set disconnect = mySig.in
disconnect.out.Stop
Set disconnect = Nothing
Set mySig.in = Nothing
```

2.4 Implementation Issues

ATLAS2K components with identical interfaces may be required to implement *different functionality* on different systems. The draft of the ATLAS2K standard mentions components operating differently in design, simulation and run-time modes. Besides these situations, ATLAS2K components should be able to use different instruments or different combinations of instruments, in order to perform identical signal generation and measurement tasks on different Automatic Test Equipment (ATE) setups. The above requirements are not directly addressed by the standard, being considered implementation issues [3].

In order to preserve TPS portability, changing instruments or the operational mode of the ATLAS2K components should not involve changes in the TPS code. Consequently, the ATS architecture must support the *mapping* of signal objects used in the TPS to different objects implementing instrument control (or simulation).

The above requirement has a very important consequence: to support instrument interchangeability, ATLAS2K-based ATS architectures should contain *two component layers*. The first layer includes the *components defined by ATLAS2K*, directly interconnected in the TPS code. Their implementation *delegates functionality* to components on the second layer, which *perform the actual control of instruments* (or implement simulation). Because the components on the second layer are instrument-specific, their mapping to components from the first layer is actually a *resource allocation* function.

Note: A very similar situation occurs for IVI drivers. To support instrument interchangeability, test programs must use generic "class drivers", which delegate functionality to "specific drivers". These components are instrument-specific and perform the actual control of instruments. The above mapping is manually specified by the user and recorded in the "IVI Configuration Store" [4].

Note: Another critical implementation issue for ATLAS2K is switching, i.e., the automatic control of the switching system in order to close the appropriate signal paths from instrument ports to UUT pins. Because switching is not directly related to the IVI Signal Interface, it will not be discussed in the present paper.

3 THE IVI SIGNAL INTERFACE

3.1 Overview

The IVI Signal Interface proposal [9] [10] envisions the standardization of interfaces for a set of COM components performing signal-oriented instrument control.

The use of the signal abstraction provides a high degree of instrument interchangeability, while the standardization of interfaces would support the portability of components between test environments from different vendors.

3.2 Architecture

The architecture proposed for the IVI Signal Interface standard is shown in **Figure 3**.

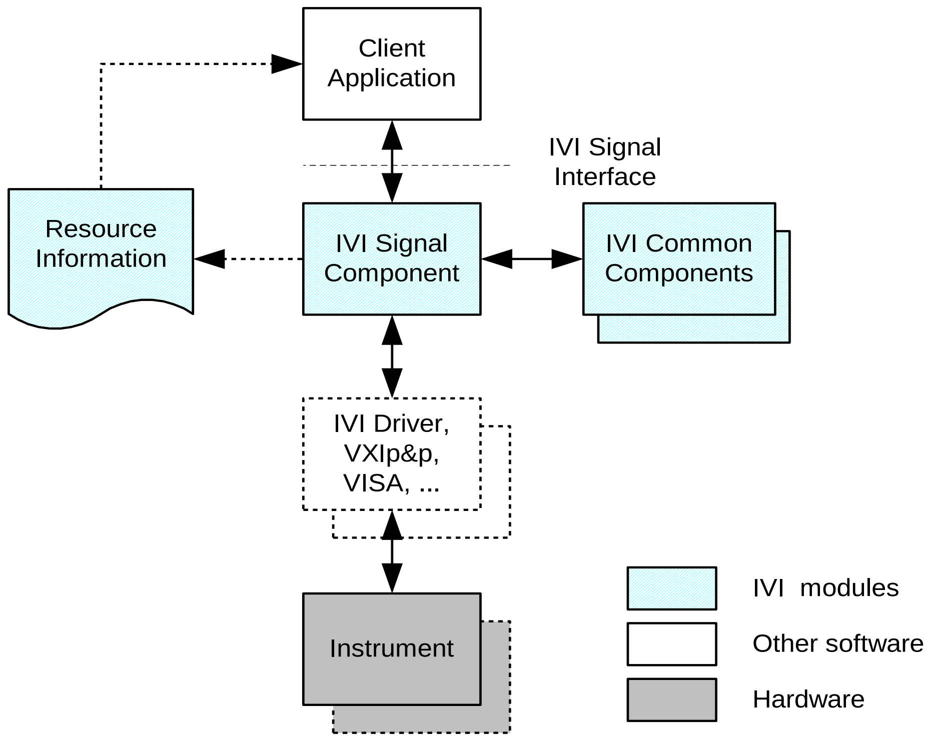

Figure 3. IVI Signal Interface Architecture

The **IVI Signal Component** is an IVI-MSS role component with a standardized application interface (the **IVI Signal Interface**). This interface contains methods that implement a set of basic signal operations such as Reset, Setup, Change, Fetch, etc. It allows the client application to control the generation and/or measurement of a physical signal that may belong to one or more signal types (e.g., DC current, AC voltage, etc.).

An **IVI Signal Component** controls one or more instruments that perform the generation or measurement of the physical signal. Instrument control may be implemented through IVI Drivers, *VXIplug&play* drivers, SCPI commands or some other mechanism. **IVI Signal Components** use services provided by **IVI Common Components** such as IVI Factory, IVI Configuration Store and IVI Event Server. This allows them to operate consistently and interact with other IVI components such as IVI Drivers and IVI-MSS Servers.

IVI Signal Components provide programmatic access to **Resource Information**, a data module that describes the signal generation and measurement capabilities of the component, including the following information:

- supported signal types
- for each signal type, the supported signal parameters
- for each signal parameter, the range, resolution and precision that may be achieved in generation or measurement
- connectivity of signal ports to instrument ports

The above information supports the automatic allocation of resources in signal-based ATSs.

IVI Signal Components may be used in different types of ATS architectures, as follows:
In signal-based ATSs, along with a general-purpose language and a component library (such as those defined in ATLAS2K). This architecture is described in detail in Section 4.

In signal-based ATSs, along with the ATLAS language. In this approach, the object code generated by an ATLAS compiler may contain calls to IVI Signal Components, which implement the actual control of instruments.

In instrument-based ATSs, along with a general-purpose language. Although such a solution does not implement automatic resource allocation or automatic switching, it can still benefit from the higher level of abstraction provided by the IVI Signal Interface. For example, test programs that use IVI Signal Components would feature a higher degree of instrument interchangeability compared to VXI*plug&play* drivers or IVI drivers, supporting the replacement of an obsolete instrument with an instrument from a different class or even with multiple instruments that operate together to generate or measure a given signal.

3.3 Standardization of Signal Type Information

In the approach proposed in [10], the IVI Signal Interface allows the *transmission of signal information* (such as signal type and signal parameters) in a *generic, signal-type independent format* (see Section 3.4 for examples). Thus, an IVI Signal Interface standard would specify the <u>semantics</u> of the information <u>channel</u> that links IVI Signal Components to their clients. For example, the standard may define a Setup method that takes as arguments:

- the role of the signal (e.g., source, sensor or load)
- the type of the signal (e.g., AC Signal)
- a collection of objects of type Parameter, where each Parameter has the properties Name and Value (e.g., Amplitude, 1.0 V).

Specifying the semantics of the information transferred through the channel implies the *standardization of signal type information*, including the following elements:

- a set of signal types and their correspondence to physical signals
- for each signal type, the set of signal parameters and their correspondence to the parameters of the physical signal

The standardization of signal type information is critical for instrument interchangeability, since it guarantees the consistent use of such information by IVI Signal Components from different vendors, as well as their client applications. It is the authors' opinion that the IVI Signal Interface standard should not specify its own signal types, but rather reference such information in an existing standard. The following arguments support this approach:

- signal types are already defined in existing ATLAS specifications; their adaptation for an interface standard is for the most part a mechanical process
- new signal types are being defined in the ATLAS2K initiative; compared to previous ATLAS specifications, the new definitions will have a precise formal definition
- being focused on defining interfaces for instrument control, the IVI Foundation is not the proper place to trigger another effort for defining signal types

3.4 Examples

The following Visual Basic code exemplifies the use of IVI Signal Components to implement typical sequences of signal operations.

Note: The sample code below would appear as shown only in instrument-based TPSs. In signal-based systems, the operations would be implemented in a component library such as ATLAS2K or in the object code generated by an ATLAS compiler (see Section 3.2).

3.4.1 Signal Generation

The following code generates a sinusoidal voltage with an amplitude of 0.5 V and a frequency of 1 MHz with 1.0 Hz resolution, using the instrument with GPIB address 1.

```
' initalize
Dim mySigSource as IviSignalSource
mySigSource.Init("GPIB:1:INSTR")

' set signal parameters and start generation
Dim control As ParamValSet
control.Add("Amp", 0.5)       'name and value
control.Add("Freq", 1.0E6, 1.0)    'name, value and resolution
mySigSource.Setup(SENSOR, "AcSignal", control)
'signal role, signal type and parameters
```

3.4.2 Signal Measurement

The following code measures the RMS amplitude of an AC voltage with a nominal amplitude of 0.5 V, using the instrument with GPIB address 2.

```
' initalize
Dim mySigSensor as IviSignalSensor
mySigSensor.Init(”GPIB:2:INSTR”)

' set signal parameters
Dim capability As ParamValSet
capability.Add(”Amp”, 0.5)
mySigSensor.Setup(SENSOR, ”AcSignal”, capability)

' start measurement
Dim measList As ParamSet
measList.Add(”RmsAmp”)
mySigSensor.Arm(measList)

' retrieve value
Dim measurements As ParamValSet
measurements = mySigSensor.Fetch()
Dim rmsValue as Double
rmsValue = measurements.GetValue(”RmsAmp”);
```

4 COMBINING ATLAS2K AND THE IVI SIGNAL INTERFACE IN AN ATS ARCHITECTURE

4.1 Principle

As shown before, ATLAS2K and the proposed IVI Signal Interface standard address *complementary areas of ATS architectures*, as follows:

- ATLAS2K components:
    - are designed for direct use in signal-based TPS code
    - as shown in Section 2.4, should not implement instrument control directly; instead, they should delegate functionality to another layer of components
    - the ATLAS2K standard defines signal types
- IVI Signal Components
    - not intended for direct use in signal-based TPS code
    - implement instrument control
    - in authors' oppinion, an IVI Signal Interface standard should not define signal type information but rather reference such information in other standards

Consequently, the *IVI Signal Interface may provide an implementation mechanism for ATLAS2K*, while *ATLAS2K defines the signal type information* required by an IVI Signal Interface standard.

On the other hand, because *both standards specify interfaces for COM components*, they may *define the framework of a coherent COM-based ATS architecture*, which may include other COM standards under development, namely *IVI-COM Drivers* [4] and *COM VISA* [14].

4.2 Example

A signal-based ATS architecture that uses ATLAS2K and IVI Signal Components is shown in Figure 4.

The **Device Information** module provides a description of capabilities for all the instruments available on the system. For example, this module may be a file generated by the ATLAS2K development environment. When a new instrument is installed on the system, the Resource Information provided by the IVI Signal Component (see Figure 3) is included in the **Device Information** file. The **Switch Information** module describes the connectivity of the **Switching Subsystem**, while the **Fixture Information** module describes the connectivity of the **Fixture**.

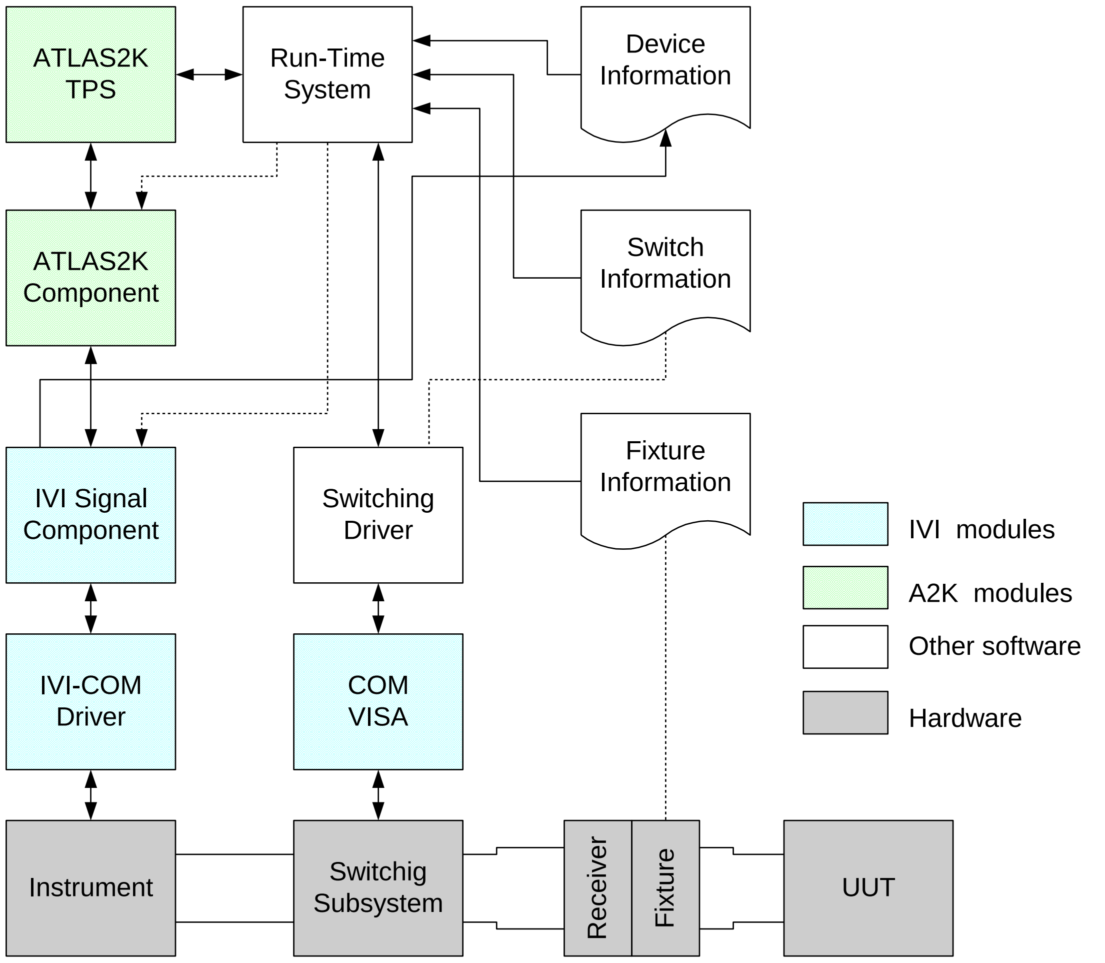

Figure 4. ATS Architecture Using ATLAS2K and IVI Components

The **ATLAS2K TPS** code requests each signal object from the **Run-Time System**. This component compares the requirements specified for the signal with the capabilities of the instruments available on the system, in terms of supported signal type and parameters, as well as range, resolution and precision of these parameters. If an instrument with sufficient capabilities is found, the **Run-Time System** uses data from the **Switch Information** and **Fixture Information** modules to verify if signal paths can be closed from **Instrument** ports to the appropriate **UUT** pins. If this condition is also fulfilled, the **Run-Time System** instantiates the **IVI Signal Component** corresponding to the allocated instrument, instantiates the **ATLAS2K Component** for the required signal, initializes the **ATLAS2K Component** by passing it a reference to the **IVI Signal Component**, then returns to the **ATLAS2K TPS** code a reference to the **ATLAS2K Component**.

The **ATLAS2K TPS** code sets properties and invokes methods of the **ATLAS2K Component**. This component calls the appropriate methods of the **IVI Signal Component**, which performs the actual control of the instrument, possibly using an **IVI-COM Driver**. During TPS operation the **Run-Time System** performs automatic switching by closing and opening the appropriate signal paths from **Instrument** ports to **UUT** pins. The actual control of the **Switching Subsystem** may be implemented by a specialized **Switching Driver**, which in turn may use SCPI commands sent over **COM VISA** to control the switches.

4.3 Development Process

Figure 5 exemplifies a software development process for the previously described ATS architecture.

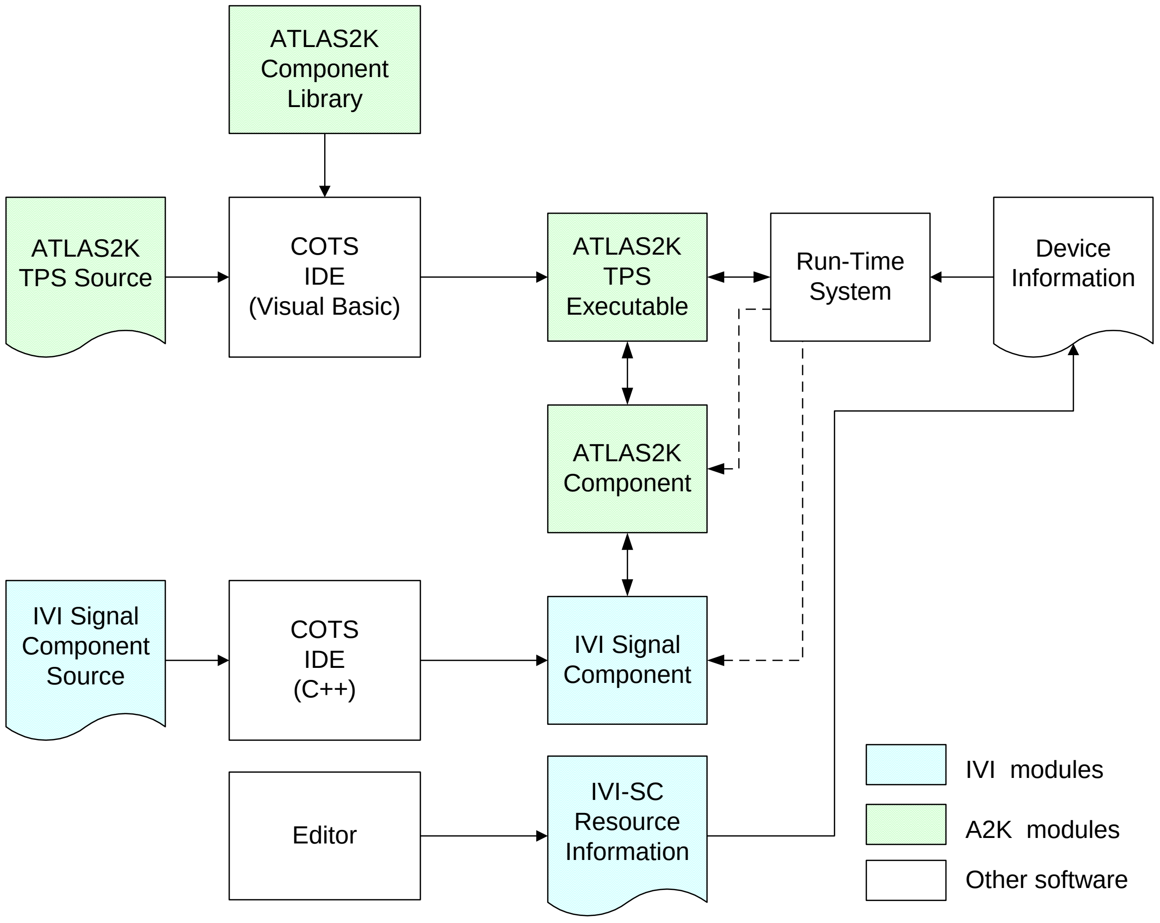

Figure 5. Software Development Process for the ATS Architecture

Commercial Off-The-Shelf (COTS) Integrated Development Environments (IDEs) such as Visual Basic or Visual C++ may be used for building TPSs and IVI Signal Components. A COTS editor may be used to edit the Resource Information file distributed along with the IVI Signal Component.

In business model envisioned to evolve from the technical specification of the IVI Signal Interface [10], the IVI Signal Components are developed by *solution providers* and delivered to system integrators, which integrate them with TPSs and the ATE hardware. The standardization of the IVI Signal Interface supports the *accurate specification and validation* of IVI Signal Components by system integrators.

5 CONCLUSIONS

The ATLAS2K standard currently under development defines a set of COM components that model signals, as well as a way to interconnect these components in order to specify test requirements. The above components may be used to write signal-oriented TPSs in general-purpose languages such as C++ or Visual Basic.

The IVI Signal Interface proposal envisions the standardization of COM interfaces for IVI-MSS components, providing a high degree of instrument interchangeability, along with the portability of components between test environments from different vendors.

ATLAS2K and the proposed IVI Signal Interface *address complementary areas of ATS architectures*. The IVI Signal Interface may provide an implementation mechanism for ATLAS2K, while the ATLAS2K standard may provide the signal type information required for the IVI Signal Interface standard.

The two standards, along with IVI-COM Drivers and COM VISA may *define the foundation of a modular, open, distributed and coherent COM-based ATS architecture*. This approach offers the following benefits:

- based on *up-to-date software technologies* such as COM and general-purpose object-oriented languages
- *wide coverage*, addressing four software interfaces defined for the ATS subdomain by the Department of Defense (DoD) Joint Technical Architecture (JTA) [14]
- decouples test and diagnosis operations from instrument control, promoting *code reuse*
- favors *TPS portability* among test execution environments from different vendors
- provides a *high degree of instrument interchangeability*

6 REFERENCES

[1] *IVI Foundation web site*, http://www.ivifoundation.org

[2] Fertitta, K.G., Harvey, J.M., *The Role of ActiveX and COM in ATE*, Proc. AUTOTESTCON 1999, San Antonio, TX, pp. 35 - 51

[3] ***, *ATLAS2K specifications*, Draft E1, April 2001

[4] ***, *IVI-3.1: IVI Driver Architecture Specification*, IVI Foundation, Rev. 0.91, March 2001

[5] ***, *IVI-3.10: Measurement and Stimulus Subsystems (IVI-MSS) Specification*, IVI Foundation, Draft Rev. 0.4, February 2001

[6] Oblad, R., "Applying New Software Technologies to Solve Key System Integration Issues", *Proc. AUTOTESTCON*, Piscataway, NJ, 1997, pp. 181-189

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

[8] ***, *IVI Signal Interface, Functional Specification*, Rev. 0.3, IVI Foundation, October 2000

[9] ***, *The Role of the IVI Signal Interface Standard in Supporting Instrument Interchangeability*, White Paper, IVI Foundation, April 2000

[10] 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 2000, Anaheim, CA

[11] *IEEE Test Description Sub-Committee web page*,
http://grouper.ieee.org/groups/scc20/td/index.html

[12] ***, *Signal and Method Modeling Language*, Issue C (Draft), August 2000

[13] ***, *Standard Test Language for All Systems - Common/Abbreviated Test Language for All Systems (C/ATLAS)*, IEEE, 1995

[14] ***, *VPP-4.3.4: VISA Implementation Specification for COM*, VXIplug&play Systems Alliance, Revision 1.0, April 2000

[15] ***, *Department of Defense Joint Technical Architecture*, Version 4.0, April 2001