Simplifying Test System Development with IVI.NET

Kirk Fertitta
Pacific MindWorks
Motivations for IVI.NET

- Present an API more suited to .NET developers
- Simplify source code
  - Allow end users to understand instrument behavior by examining driver source
  - Allow end users to fix bugs on their own
  - Allow end users to add features to drivers on their own
- Richer, more expressive APIs
  - More flexibility with API data types
  - Clean handling of asynchronous notifications (aka “events”)
- Side-by-side deployment of drivers
  - Only one version of an IVI-COM or IVI-C driver can be installed at a time
  - IVI.NET allows multiple versions of a driver to be installed
IVI-COM and IVI-C Driver Source

- IVI-COM and IVI-C drivers are both quite complicated internally
- Supporting IVI driver features requires a lot of code
  - Multi-thread safety
  - Simulation
  - Range-checking
  - State-caching
- “Basic” COM plumbing is complex and not well understood by many
- Multi-model driver support can be complicated
- Driver development tools are required, but can only do so much
  - Nimbus Driver Studio and LabWindows both work hard to factor as much code “out of the way”
  - Tooling around C/C++ is just plain hard
- Users trying to debug through an IVI-COM driver would find themselves traversing numerous confusing source code files
IVI.NET Driver Source

Very clean and simple method implementations

- Often can be done with a single-line of code
- No “code-beside” files => simple in-line implementation of each method

Plumbing “goo” for many features factored into simple attributes

- State caching, range-checking, coercion, locking, parameter validation, and more...
- This makes it very easy for end users to customize driver behavior without writing any procedural code

Simplified I/O by use of standard I/O

- All of the advantages of IVI.NET discussed will be available at the I/O level as well
- VISA.NET API nearing completion by IVI Foundation
- Pre-release available as part of Nimbus Driver Studio distribution

Any .NET programmer will easily be able to understand and modify an IVI driver
Advanced Tooling for IVI.NET

- Many IVI-COM and IVI-C complaints tied to complex source code
- Tools have even more difficulty dealing with C/C++ source than humans
  - Includes files and macros are particularly problematic
  - Few really good C++ refactoring exist in any domain
- A prime motivator for .NET itself is the improved ability to create tooling
- Simpler source possible because .NET code can more easily be roundtripped
- Static analysis tools highlight issues at compile time that previously could only be detected at runtime
- Browsers can easily interrogate an IVI.NET driver and understand its features
- Declarative attributes can be used where procedural code was previously required
  - Achieved via “extending” the compiler (aka “code-weaving”)
- Result is that tool-generated code will look just like hand-written code
Static Analysis Example 1

```java
[DriverMethod]
public void Configure(double bandwidth, double frequency)
{
    // ...
    ioPRINTF("CONFIG %g,%g,%s", bandwidth, frequency);
}
```

![Error List]

<table>
<thead>
<tr>
<th>Description</th>
<th>File</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN1001 : VisaNet : Method 'Acme5403 Configure(double, double)' has a call to Printf where the number of arguments required by the format specifiers (3) does not match the number of arguments provided (2).</td>
<td>Acme5403.cs</td>
<td>453</td>
</tr>
</tbody>
</table>
Static Analysis Example 2

```java
[DriverMethod]
public void Configure(double bandwidth, double frequency) {
    // ...
    ioPRINTF("CONFIG %g,%s", bandwidth, frequency);
}
```
Richer Type System

Both IVI-COM and IVI-C drivers suffer from a limited set of data types

- Integers, floats, Booleans, strings
- Arrays of the above, but extra parameters are required in IVI-C

IVI-C cannot expose an array of strings

IVI-C cannot expose structs

- Can be done in IVI-COM, but it’s tedious to implement

```c
IviScope_FetchWaveform(ViSession vi,
    ViConstString channel,
    ViInt32 waveformSize,    // # of elements on input
    ViReal64 waveform[],     // actual data buffer
    ViInt32 *actualPoints,   // # of elements on output
    ViReal64 *initialX,
    ViReal64 *xIncrement);
```
Simplifying APIs with .NET Types

IVI-C signature

```c
IViDigitizer_FetchWaveformReal64(ViSession Vi,
    ViConstString ChannelName,
    ViInt64 WaveformArraySize,
    ViReal64 WaveformArray[],
    ViInt64* ActualPoints,
    ViInt64* FirstValidPoint,
    ViReal64* InitialXOffset,
    ViReal64* InitialXTimeSeconds,
    ViReal64* InitialXTimeFraction,
    ViReal64* XIncrement);
```

IVI.NET signature

```c
Channels[].Measurement.FetchWaveform(IWaveform<Double> waveform)
```
How to deal with Events?

• IVI-COM and IVI-C drivers almost never expose events
  • Exposing something as commonly needed as an SRQ is tortuous
  • Requires special knowledge/programming by the driver developer
  • Requires special knowledge/programming by the client programmer

• .NET supplies a standard mechanism for exposing events
  • No special programming required by the driver developer or client programmer
  • Trivial code to subscribe/unsubscribe
  • Trivial code for driver developers to customize subscribe/unsubscribe semantics

• Warnings can now be dealt with properly in IVI drivers by the use of events
Shared IVI.NET Data Types

IVI Foundation felt it would be useful to offer commonly used data types as part of the IVI.NET Shared Components

- Increase consistency amongst IVI.NET drivers
- Facilitate data interchange between drivers

Standardized IWaveform and ISpectrum interfaces

- Digitizers and scopes and RF spectrum analyzers all read waveforms
- Function generators and RF signal generators source waveforms
- Without a common definition of a “waveform”, client applications would need to write the tedious code to translate between each class’s notion of a waveform

Time-based parameters can use PrecisionDateTime and PrecisionTimeSpan

- No confusion about ms vs sec, absolute vs relative time, UTC time, etc
- Precision adequate for IEEE 1588 devices

Common trigger source data type

- Useful in “stitching” together devices in triggered source-measure operations
Error Handling in IVI.NET

- **IVI-C drivers rely solely on return codes**
  - Errors can easily be ignored by the client application
  - After getting the error code, a second function call is required to get the message
  - Special handling of warning codes required

- **IVI-COM error code handling depends upon the client environment**
  - Return codes in raw C++
  - Special exception classes in C++
  - COMException class in .NET interop scenarios
  - .NET clients can’t see warnings at all from IVI-COM drivers

- **IVI.NET drivers always use exceptions**
  - User can always see the full context of the error
  - Error content less dependent upon specific driver implementation
  - Natural mechanism
Simplified Usage Syntax

- Simplified access to very commonly used features
  - Enums
  - Repeated capabilities (e.g. “channels”)

C# client using IVI-COM driver through interop

```csharp
digitizer.Arm.Sources.get_Item("LAN3").Detection = IviLxiSyncArmSourceDetectionEnum.IviLxiSyncArmSourceDetectionHigh;
```

C# client using IVI.NET driver

```csharp
digitizer.Arm.Sources["LAN3"].Detection = ArmSourceDetection.High;
```
Performance of IVI.NET

- Fewer memory leaks
- Reference counting has a cost
  - Reference count field per-object
  - Increment and decrement called much more frequently than one might think
  - Reference count field must be thread-safe
    - Even more per-object overhead
    - Frequently lock/unlock operations
- Conventional memory-managed systems (such as C-runtime library) produce highly fragmented memory
  - Allocation of objects can be expensive
  - Objects spread out in memory => poor locality of reference
- .NET memory allocation produces very good locality of reference
  - Object allocation extremely fast
  - Objects allocated close together in time live close together in memory
  - Fewer cache misses and better virtual paging performance
Dynamic Memory Allocation in .NET