A .NET ASSEMBLY FOR EPICS SIMPLE CHANNEL ACCESS*

C. Timossi, H. Nishimura, LBNL, Berkeley, CA, U.S.A.

Abstract

The Advanced Light Source (ALS) at Lawrence Berkeley National Laboratory is starting a demonstration project to re-write the control room operator interface software using the .NET platform and the C# programming language [1]. Simple Channel Access (SCA) [2], developed at LBNL to simplify client access to EPICS[3], will be replaced with a new .NET assembly, ScaNET, that enables other .NET assemblies to access accelerator data.

INTRODUCTION

We have reported earlier on our effort using the .NET framework for accelerator applications [4], interoperability with EPICS Channel Access (CA) and on the feasibility of cross platform use [5]. Here we wish to report on our efforts to scale-up ScaNET for use in a complete re-write of the operator interface applications for the ALS injector.

SCANET BACKGROUND

ScaNET is a .NET class-library *assembly* written in C# meant for use by .NET applications needing CA client functionality.. An assembly has the same file extensions as a windows executable, either exe or dll, but has a different file format that includes meta-data allowing it to be self-describing.

Goals

The goals for ScaNET are somewhat conflicting:

- Provide a thin layer around the CA library which has proven over many years to be a robust and high-performance network layer for accelerator data transport.
- Hide some of the details of the CA interface in a class that is more intuitive to a .NET application builder than the CA API.

Design

The design utilizes 2 classes. Als.Epics.ChannelAccess is a static class that contains all the direct .NET to Ca.dll mappings using the Platform Invoke (P/Invoke) interface provided by .NET System.Runtime.InteropServices. This class is used by Als.Epics.SimpleChannelAccces which exposes a class more suitable for .NET applications.

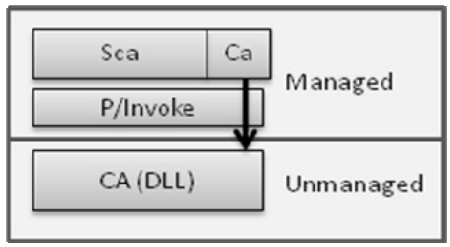

Figure 1: Als.Epics.SimpleChannelAccess (Sca) utilizes the Als.Epics.ChannelAccess to call into unmanaged code

TOOLS

The programming tools for .NET are worth special mention. They have been an absolutely key element to our progress.

Visual Studio 2008

One of the main motivators for adopting the .NET framework is the power of the Visual Studio integrated development environment. Especially notable are *IntelliSense*, a technology which recognizes types during an editing session allowing auto-completion and providing documentation for the class being entered, the symbolic debugger, and integrated unit testing.

In addition to providing symbolic information for the project being debugged, the debugger can be coupled with a symbol server that allows names of windows system calls to be displayed in the stack trace. Other symbolic information, such as the CA system calls can also be referenced in the debugger through the symbol server.

ScaNET makes heavy use of the integrated unit testing feature of VS. Since ScaNET is still under development, running the unit tests periodically help assure that code changes in one area don't break code in other areas.

Measuring Performance

MeasureIt [5] is an application, available for download, that compares the CPU performance of various .NET operations scaled to the CPU time for an empty static function call. The source code can be modified to include user code.

The author of MeasureIt has a good discussion of the interpretation of performance measurements in general with special comments on .NET.

Some of the output of MeasureIt is shown in Table1.

* Work supported by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Table 1: Output of MeasureIt. Scaled where EmptyStaticFunction = 1.0 (2.9 nsec = 1.0 units)

Name	Median	Mean	StdDev	Min	Max	Samples
NOTHING [count=1000]	2.351	2.225	0.630	0.386	2.842	10
MethodCalls: EmptyStaticFunction() [count=1000 scale=10.0]	1.000	1.011	0.026	1.000	1.088	10
Loop 1K times [count=1000]	452.175	461.586	19.742	452.070	512.281	10
PInvoke: 10 FullTrustCall() (10 call average) [count=1000 scale=10.0]	5.702	5.783	0.133	5.656	5.972	10
PInvoke: PartialTrustCall() (10 call average) [count=1000 scale=10.0]	20.953	21.181	0.894	20.674	23.821	10
GroupGetDouble: Sca.GetDouble(grouped) [count=1000 scale=10.0]	6564.613	6573.730	55.253	6500.874	6686.501	10
GetDouble: Sca.GetDouble(ungrouped) [count=1000 scale=10.0]	42658.520	42792.610	259.964	42506.390	43238.630	10

Subversion Revision Control

Although CVS is still the main version control system at use in the ALS, for the upgrade project, we are using Subversion. The primary SVN client tool for Windows is TortoiseSVN [6] which is implemented as a file explorer extension.

DEVELOPMENT ISSUES

In the latest development cycle of ScaNET, there are a few issues that stand out.

Error Handling and Error Logging

.NET encourages the use of exception handling in methods for handling errors rather than to return error codes. We found a heavy use of exceptions to be useful during application development. At this stage the stack trace dialog box that the system displays for unhandled exceptions is useful. For production we decided that ScaNET should not throw exceptions in as many cases but should write to the Windows Event Log instead. ScaNET of course has routines to return the status of previous operations but it's up to the application to check that status.

A CA timeout is an example of an error which is logged rather than handled with an exception. These timeouts occur routinely but rarely. Usually a timeout can be handled by simply asking for the value again. On the other hand it's very useful to record the fact that the timeout occurs.

Interfacing Managed and Unmanaged Code

A .NET assembly runs under the management of the .NET framework, in particular, under the management of the Common Language Runtime. CLR garbage collection is constantly shuffling objects in a memory area known as the managed heap as object go out of scope. Code in managed memory (managed code) has access to code in unmanaged memory, such as the windows system dlls, through InteropServices as illustrated in Figure 1. This separation of managed and unmanaged areas has implications in several areas.

On 64 bit Windows (Vista x64 or XP Pro x64) the debugger cannot debug a combination of managed and unmanaged code. So most development work has to be done on 32 bit Windows.

By default, the CLR 'walks the stack' before a method is called to check that all its callers have at least the same privilege as it does (the callers can have greater privilege). This behavior has known performance penalties. Note in Table 1 that a fully trusted method call through P/Invoke is almost 4 times faster than a partially trusted method call. P/Invoke methods can be marked with the attribute: *SuppressUnmanage-CodeSecurityAttribute* to eliminate the stack walk. Setting this attribute could be an important performance gain though the gain is modest in the case of most calls to CA.

The P/Invoke interface is primarily used for managed code to call into unmanaged code. However, a .NET delegate can be used by unmanaged code to call into managed code.

.NET Delegates and CA Events

A CA client can ask to be notified of changes in value or state of a process variable through a callback mechanism. On Windows, the callback function uses the C calling convention (arguments passed from right to left) with an event structure as an argument (passed by value).

.NET defines a type-safe function pointer know as a *delegate* which can be used to implement callbacks. ScaNET has delegates for CA event callbacks and connection callbacks but we had some trouble getting delegates to work reliably until VS 2008. There are restrictions, naturally, on unmanaged code calling into managed code. For example, an exception will be thrown if unmanaged code tries to write to managed memory.

DEPLOYMENT

The installer file ScaNET.msi is used to install the assembly in the 'Program Files' directory. This installer must be executed with Admin privileges since it creates a Windows Event Log.

A .NET assembly that is meant to be shared, like ScaNET, could instead be installed in the file area known as the Global Assembly Cache (GAC). There are some advantages to installing it as a *shared assembly* but during the development phase, we found this method to be too cumbersome. A special tool, *gacutil*, or an installer file

has to be used to write to this area and these two methods have some compatibility issues. Also, referencing an assembly from the GAC in VS is not convenient.

FURTHER WORK

Error handling can use improvement. The Windows event log is very convenient but breaks the compatibility with the .NET platform on Linux (MONO). It would be better to go back to standard .NET exception handling and redirect the exceptions to a log file as a configuration option.

The way CA events are now handled can be improved. Currently the user creates a delegate to handle subscriptions. It would be more natural for ScaNET to handle the callback and generate a .NET event to subscribers.

Unlike the original SCA, ScaNET does not try to moderate the number of requests that the client can make to an IOC. This is a real problem for many of the IOCs at the ALS that run older hardware and older versions of EPICS; they simply can't handle the demand.

ACKNOWLEGEMENTS

We would like to thank Greg Portmann for his leadership on the control room upgrade effort and Craig Ikami for all his help setting up the control room systems.

REFERENCES

[1] ECMA-334 and ISO/IEC 23270
[2] http://www-controls.als.lbl.gov/epics_collaboration/sca/
[3] Dalesio, et al. "The Experimental Physics and Industrial Control System Architecture," submitted to ICALEPCS, Berlin, Germany, Oct. 18-22, 1993.
[4] H. Nishimura and C. A. Timossi, "Control Room Application Development Using .NET", PCAPAC 2005
[5] H. Nishimura and C. Timossi, "Mono for Cross-Platform Control System Environment", PCAPAC 2006
[5] V. Morrison, "Measure Early and Often for Performance", MSDN Magazine, April 2008, Vol. 23, No. 5, msdn2.microsoft.com/magazine/cc135911
[6] http://tortoisesvn.tigris.org/