A .NET INTERFACE FOR CHANNEL ACCESS

G. Cox, STFC Daresbury Laboratory, U.K.

Abstract

The control system for Accelerators and Lasers In Combined Experiments (ALICE) under construction at Daresbury Laboratory uses EPICS and vxWorks on VME64x. The client software in use during the commissioning of the accelerator is based on PC consoles running Red Hat Linux 9. Synoptic displays and engineering panels are created using the Extensible Display Manager (EDM) and other standard EPICS extension software is used for archival, alarm handling etc. A similar EPICS based control system is being used for the commissioning of the Muon Ionisation Cooling Experiment (MICE) under construction at the Rutherford Appleton Laboratory. The Synchrotron Radiation Source (SRS) control system uses a bespoke control system with client software on PC consoles running Microsoft Windows. We would like to employ a similar approach for the operational client software on ALICE and MICE with Channel Access (CA) clients running on Microsoft Windows PC consoles. This paper presents the .NET Channel Access interface developed at Daresbury and showcases .NET client applications being developed for both ALICE and MICE operations.

INTRODUCTION

CA clients for the commissioning of ALICE and MICE are currently hosted on Linux PC consoles running a Scientific Linux operating system. We will be migrating to the Microsoft Windows platform for ALICE operations and for the later phases of the MICE project. The Electron Model for Many Applications (EMMA) non-scaling Fast Field Alternating Gradient (FFAG) accelerator currently under construction at Daresbury will also be integrated into the ALICE control system and utilise CA client software on the Windows platform.

A number of different options were investigated for building CA clients on the Microsoft Windows platform. These included ActiveX CA, Java CA (JCA), CA Java (CAJ) [1], and calling native code in CA via C++.

Each of these options has its own advantages and disadvantages. ActiveX CA is simple to use, however performance is limited and Process Variable (PV) support is incomplete compared to the records of an Input/Output Controller (IOC).

JCA performance is again limited due to the implementation using Java Native Interface (JNI), this is improved by the CAJ native Java implementation. Although, with Java's rigid adherence to the notion of write once, run anywhere it can be difficult to use to the maximum the unique features and modes of working within an individual desktop environment.

For best performance the calling of native code in the CA dynamic-linked libraries (DLLs) can be used, but this is a complex approach not ideally suited to rapid application development of visual applications.

Microsoft is promoting .NET as its flagship development platform. As such it seemed a logical way forward for developing CA clients on the Microsoft Windows platform. At the time of starting this development there did not exist a full CA implementation for the .NET platform.

THE MICROSOFT .NET FRAMEWORK

The Microsoft .NET Framework is a software technology that is available with several Microsoft Windows operating systems. It includes a large library of pre-coded solutions to common programming needs and a virtual machine that manages the execution of programs written specifically for the framework. The .NET Framework is a key Microsoft offering and is intended to be used by new applications created for the Windows platform.

The pre-coded solutions that form the framework's Base Class Library cover a large range of programming needs in a number of areas, including user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. The class library is used by developers who combine it with their own code to produce applications.

Applications developed using the .NET Framework execute in a software environment that manages the application's runtime requirements. Also part of the .NET Framework, this runtime environment is known as the Common Language Runtime (CLR). The CLR provides the appearance of an application virtual machine so that programmers need not consider the capabilities of the specific CPU that will execute the program. The CLR also provides other important services such as security, memory management, and exception handling. The class library and the CLR together compose the .NET Framework.

Managed code is code that has its execution managed by the .NET Framework CLR. Unmanaged code executes outside of the control of the .NET CLR. Unmanaged code may perform unsafe operations such as pointer arithmetic and is used for accessing unmanaged memory, calling Windows APIs, interfacing to Component Object Model (COM) components, and coding performance-critical methods which avoid the overhead of the CLR.

Platform Invocation Services

Platform Invocation Services, commonly referred to as P/Invoke, is a feature of Common Language Infrastructure implementations, like Microsoft's CLR, that enables managed code to call native code in DLLs.

The native code is referenced via metadata that describes functions exported from a native DLL.

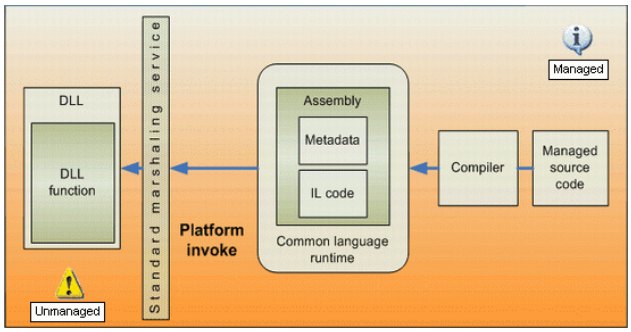

Figure 1: Illustration of Platform Invoke.

.NET CHANNEL ACCESS INTERFACE

Platform Invoke has been used to create a comprehensive CA implementation for the .NET platform. Following this approach allows the .NET CA implementation to be divorced from the internal implementation of CA within EPICS base. The .NET CA interface makes use of function calls to the EPICS 3.14 implementation of CA [2]. Provided the interface exposed by the CA implementation of future versions of EPICS base remains consistent, then the .NET CA interface can be built and run against any 3.14 or above version of EPICS.

This approach also enables the .NET CA interface to be independent of any changes to the internal implementation of CA in EPICS base. Any improvements or bug fixes applied to the CA libraries of EPICS base libraries can immediately be taken advantage of by linking the .NET CA interface against the new version of the CA libraries.

Development Tools

Initial development was carried out in C# within Visual Studio 2003 (.NET 1.1), before upgrading to Visual Studio 2005 (.NET 2.0) and then to Visual Studio 2008 (.NET 3.0 & 3.5).

Early versions of the interface utilising .NET 1.1 relied upon having special versions of the EPICS base CA libraries built using the .NET compiler. Upgrading from .NET 1.1 to .NET 2.0 caused the interface to stop working as callbacks from unmanaged code would lead to memory access violations. This was due to major changes by Microsoft to the .NET Framework between version 1.1 and 2.0, including to the implemetation of P/Invoke.

This technical problem with later versions of the .NET Framework had prevented any serious .NET development with CA getting off the ground. After lengthy investigation it was found that defining the P/Invoke callback function pointers with *CDecl* calling convention prevented callbacks from unmanaged code causing memory access violations.

The move to .NET 2.0 and the resolution of the memory violations also removed the need for using a special .NET build of the EPICS base CA libraries. The interface can now be built and run against a 'standard' build of the EPICS 3.14 CA libraries.

Implementation

The implementation of the .NET interface consists of the following areas:

DLL import declarations, callback delegate declarations, structure declarations, static methods, constant declarations and enumeration declarations.

The DLL import declarations allow managed code within .NET applications to call unmanaged code within the CA libraries. These declarations enable applications to create and destroy CA contexts, create and destroy CA channels, create and destroy subscriptions and to query various attributes of a CA channel.

Many of the methods called within the unmanaged CA libraries will receive and pass data back to managed code via opaque pointers. An opaque pointer allows the calling application to pass complex data into, and receive complex data back from, the unmanaged library without concerning itself with the format or contents of the complex data.

In the .NET Framework pointers are considered to be unsafe and as such cannot be used within managed code. The framework provides the type *IntPtr* to allow pointers to be passed and received, although no pointer arithmetic is allowed. The *IntPtr* type can be used to marshal opaque pointers to and from the unmanaged CA libraries enabling the user of the .NET CA interface to be unconcerned about the use of pointers.

To provide asynchronous event notification via CA, the unmanaged libraries utilise callbacks. For these callbacks to be received within managed code callback delegates must be declared within the interface. As mentioned earlier, these callback delegates must be declared with the *CDecl* calling convention. Once a callback delegate has been defined within managed code according to the signature contained within the interface declaration it's address can be passed to the unmanaged CA library enabling asynchronous event notification.

The methods of the unmanaged CA libraries also pass and receive data in custom data structures. These data structures must be defined within the .NET interface to allow data to be exchanged with the unmanaged libraries. As these data structures are used to marshal data between managed and unmanaged code certain types must be declared as unmanaged types and marshalled with care. Character arrays are an example of a type that must be marshalled this way.

In the implementation of the unmanaged CA library many useful macros are defined within C header files. An example application of one of these macros is to convert EPICS database types to text. In the development languages of the .NET framework no support exists for

defining C style macros. To create a full CA implementation for .NET these macros must be implemented as methods and declared within the interface as public and static.

To complete the implementation constant declarations, for example DBF types and DBF_TEXT types, and enumerations, such a channel connection state, have been added.

FUTURE DEVELOPMENT

The .NET CA interface has now reached a state where it is being used to develop client applications for the Windows platform. Due to the complete unmanaged CA implementation being expansive and complex, required CA functionality above and beyond what is already implemented in the .NET interface is added as required. The interface will grow as more client applications are developed and the interface is expected to become a complete .NET implementation of CA functionality.

To simplify client application development a .NET Process Variable (PV) class library is being developed. This PV library allows client application developers not to require an understanding of the methods needed to create and destroy CA contexts, channels and subscriptions, as the class library will take care of this for them.

The class library follows an object oriented approach and contains a base PV class which exposes the common properties of an EPICS PV and allows an application to asynchronously connect to an EPICS PV and receive data and events. This base PV class is then extended via inheritance to include data formatting (for enumerated types etc), alarm handling and colour PV implementation.

The PV class library has been used to begin building a library of EPICS controls for the .NET framework [3]. These controls are intended to give an EDM-like way of generating client applications for the Windows platform. The control library allows developers to generate CA client applications within any Visual Studio .NET language without the need for any code to be written. The control library currently contains a number of EDM-like controls e.g. text box, label, symbol, button, menu button, spin button, combo box, byte, related display. This will be extended to add more controls to allow client applications to be created in an EDM-like way on Windows platforms.

CLIENT APPLICATIONS

The .NET CA interface with the class library and components built on top of it are currently being used to generate client applications for the ALICE control system. They will also be used in future to generate client applications for the EMMA and MICE control systems. An example client application for the ALICE control system is shown in Figure 2.

In the future it is intended to develop more generic Windows applications for EPICS using the .NET CA interface. These would include applications for alarm handling, archive data retrieval and real-time plotting, backup and restore tools etc.

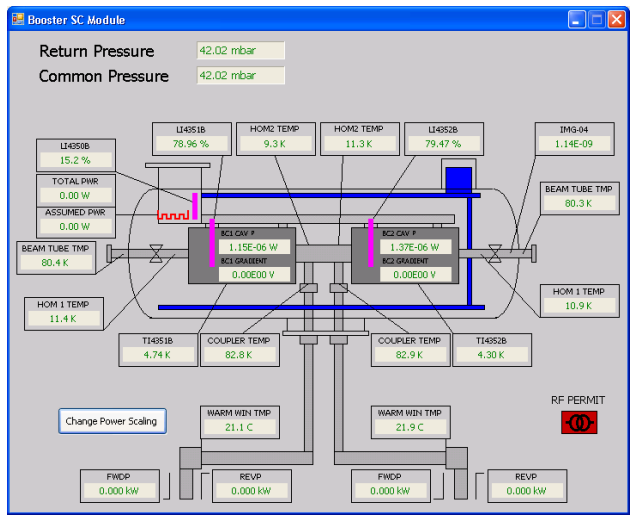

Figure 2: Example C# application built with the .NET control library.

CONCLUSION

The decision to use Microsoft Windows as a possible platform for operational application software for ALICE and EMMA introduced the requirement to have a reliable and efficient interface into CA for the Microsoft Windows operating system. After investigating and evaluating the current options for CA Windows client development the decision was taken to develop a .NET interface for CA.

Several technical issues were encountered during the development of the interface, however these have now been resolved and we now have a useable and reliable .NET CA interface. The performance of the interface has proven sufficient for all applications created so far, and with the PV class library and .NET EPICS control library we now have a simple and efficient way to develop CA applications for the Windows platform.

REFERENCES

[1] M. Sekoranja, “Native Java Implementation of Channel Access for EPICS”, ICALEPCS’05, Geneva, October 2005, PO2.089-5

[2] Jeffrey O. Hill, “EPICS R3.14 Channel Access Reference Manual”, http://www.aps.anl.gov/epics/base/R3-14/9-docs/CAref.html

[3] G. Cox, B.G. Martlew, A. Oates, “Channel Access Clients on the Microsoft Windows Platform”, ICALEPCS’07, Knoxville, October 2007, TPPA30