CHANNEL ACCESS CLIENTS ON THE MICROSOFT WINDOWS PLATFORM

G. Cox, B.G. Martlew, A. Oates, STFC Daresbury Laboratory, U.K.

Abstract

The control system for the Energy Recovery Linac Prototype (ERLP) under construction at Daresbury uses the Experimental Physics and Industrial Control System (EPICS) and vxWorks on VME64x [1]. 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.

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 ERLP with Channel Access (CA) clients running on Microsoft Windows PC consoles.

However, the Microsoft Visual Studio development tools and ActiveX/COM technologies used for creating client side software on the SRS control system are now outdated and have been superseded by the .NET framework and associated developer tools. This paper discusses the different options currently available for developing CA clients on the Microsoft Windows platform, along with progress in creating CA clients for the .NET framework.

INTRODUCTION

CA clients for the commissioning of ERLP are currently running on Linux consoles with the Red Hat 9 operating system. We have plans to migrate to the Microsoft Windows platform for ERLP operational applications. This will allow us to run the applications and access data on the control system from office PC's, and remove the need for remote access via Exceed to ERLP console machines. It will also maintain a standard look and feel consistent with other Microsoft Windows applications and the SRS control system with which Daresbury engineers and physicists are familiar.

A number of different options currently exist for building CA clients on the Microsoft Windows platform. These include ActiveX CA, Simple CA (SCA) [2], Java CA (JCA), CA Java (CAJ) [3], 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 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. 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 building visual applications.

Microsoft is promoting .NET as its flagship development platform. As such it seems a logical way forward for developing CA clients on the Microsoft Windows platform. Currently there does not exist a full CA implementation for the .NET platform.

THE MICROSOFT .NET FRAMEWORK

The Microsoft .NET Framework is a software component that can be added to, or is included with, the Microsoft Windows operating system. It provides a large body of pre-coded solutions to common program requirements, and manages the execution of programs written specifically for the framework.

The pre-coded solutions that form the framework's class library cover a large range of programming needs in areas including: user interface, data access, database connectivity, web application development, and network communications. The functions of the class library are used by programmers who combine them with their own code to produce applications.

Programs written for the .NET Framework execute in a software environment that manages the program's runtime requirements. This runtime environment, which is also a part of the .NET Framework, 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 mechanisms, 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 CLR. Unmanaged code may perform unsafe operations such as pointer arithmetic and is used for accessing unmanaged memory, calling Windows APIs, interfacing to COM components, and coding performance-critical methods which avoid the overhead of the CLR.

Platform Invocation Services

Platform Invocation Services, commonly referred to as just 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.

We have used P/Invoke to create a comprehensive CA implementation for the .NET platform. Using this approach, unmanaged code in the CA DLLs can be called from within managed code executing within the CLR. This allows applications to be written in any of the Visual Studio .NET supported languages including C# and Visual Basic .NET.

This CA implementation allows both synchronous access to PVs along with asynchronous access using callback delegates. Data structures in CA have to be carefully marshalled via P/Invoke paying particular attention to character arrays which must be marshalled as unmanaged types (see Code Sample 1).

```
[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Ansi)]
public struct dbr_ctrl_string
{
        public short  status;
        public short  severity;
        [MarshalAs(UnmanagedType.ByValArray,
        SizeConst=40)]
        public char[] value;
};
```

Code Sample 1: Example dbr_ctrl declaration.

Development Tools

Preliminary development was carried out in C# using P/Invoke within Visual Studio 2003 (.NET 1.1), before upgrading to Visual Studio 2005 (.NET 2.0).

To allow callbacks into managed code from the CA DLLs to operate correctly in Visual Studio 2003, .NET CA DLLs had to be built using Visual Studio 2005 (thanks to Chris Timossi, LBNL). When upgrading to Visual Studio 2005, these callbacks no longer worked correctly. Switching back to 'standard' non .NET built CA DLLs seemed to improve matters, but non-trivial callbacks would still not operate correctly. To solve this problem an attribute (see Code Sample 2) has to added to the callback delegate declaration.

```
[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
```

Code Sample 2: Callback delegate attribute.

PROCESS VARIABLE CLASSES

Following an object oriented approach to implementing the CA interface for .NET, several EPICS process variable classes have been created. The first of these, which acts as a base class, is a simple implementation of PV functionality and attributes. This class exposes a channel ID, connection state, precision, engineering units and type. It also manages the connection and sets up a callback for channel value changes. The CA i/o and event polling is all handled internally allowing a developer to create a connection to a PV and receive data simply by creating an instance of the class and registering an event handler.

Further sub-classes have been created which extend the base PV class functionality to include data formatting (for enumerated types etc), alarm handling and colour PV implementation.

.NET CONTROL LIBRARY

The PV classes created to allow a consistent method of access to EPICS PVs have been used to build a library of controls for the .NET framework. 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 is in the early stages of development and currently only contains a small number of EDM-like controls.

CA Label

The CA Label control is used to display a channel's value textually. It can be alarm sensitive and also implements a colour PV and visibility PV. Data is formatted according to the precision in the database, or a user specified precision.

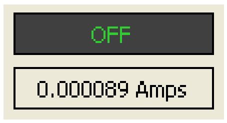

Figure 1 – Examples of the CA Label control.

CA Shape

The CA Shape control is used to display a channel's value visually. It implements a colour PV and visibility PV. A number of shapes are available including rectangles, diamonds, triangles and ellipses.

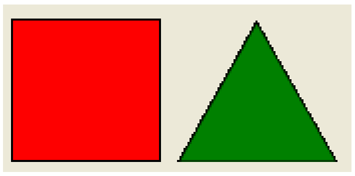

Figure 2 – Examples of the CA Shape control.

CA Byte

The CA Byte control is used to display a multi-bit channel's value either visually or textually. It can be alarm sensitive and will display up to 16 bits of a multi-bit channel.

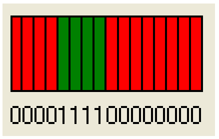

Figure 3 – Examples of the CA Byte control.

CA Symbol

The CA Symbol control is used to represent a display state. This state is selected based on the values of a number of PVs. The display state can be based upon a single PV or multiple PVs either with or without a binary truth table. Each display state is represented by an image file. These may be bitmap, jpeg, vector graphics or animated gif.

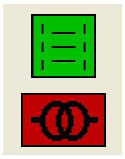

Figure 4 – Examples of the CA Symbol control.

CA TextBox

The CA TextBox control is used to control a channel's value textually. It can be alarm sensitive and has options for focus updates and lose focus writes. Data is formatted according to the precision in the database, or a user specified precision. The control also supports auto-completion which can be populated automatically for enumerated type PVs.

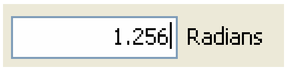

Figure 5 – Example of the CA TextBox control.

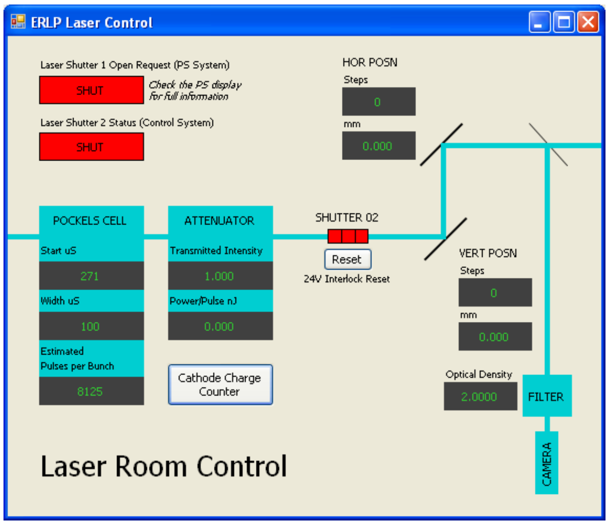

Figure 6 – Example C# application built with the .NET control library.

FUTURE DEVELOPMENT

Plans for future development of the .NET CA interface include integration of knob control for incrementing and decrementing channel values. We have two Universal Serial Bus (USB) knob devices which could be incorporated into the interface. One is a standard USB commercial video editing device. The other a bespoke controller currently used via a PCI bus on the SRS control system. The PCI bus will be replaced with a USB interface allowing simpler integration into the .NET CA interface.

The .NET control library will also be extended to introduce more EDM-like controls to allow displays for ERLP operations to be quickly and easily generated for the Microsoft Windows platform.

CONCLUSION

The decision to use Microsoft Windows as a possible platform for operational application software for ERLP introduced the need to have a reliable and efficient interface into CA for the Microsoft Windows operating system. After investigating and evaluating the existing options the decision was taken to develop a .NET interface for CA.

A number of problems were encountered during the development of the interface, however these have now been overcome and we now have a useable and reliable .NET CA interface. The performance of the interface has proven sufficient for all applications generated so far, and with the PV class library and .NET control library we now have a simple and efficient way to develop CA applications for the Windows platform.

REFERENCES

[1] B. Martlew, G. Cox, S. Davis, A. Duggan, A. Oates, A. Quigley, R. Rotheroe, "Status of the ERLP Control System", ICALEPCS'07, Knoxville, October 2007, TPPB38

[2] C. Timossi, H. Nishimura, J. McDonald, "Experience with ActiveX control for simple channel access", PCaPAC'02, Frascati, October 2002, LBNL-51591

[3] M. Sekoranja, "Native Java Implementation of Channel Access for EPICS", ICALEPCS'05, Geneva, October 2005, PO2.089-5