CONTROL ROOM APPLICATION DEVELOPMENT USING .NET*

H. Nishimura[1] and C. A. Timossi[2], LBNL, Berkeley, CA 94720, U.S.A

Abstract

Many of the current ALS control room Windows applications make use of pre-existing ActiveX controls that encapsulate the detailed logic needed for control of complex devices in the accelerator. We have found that this methodology of Win32 application development based on these controls useful both because they are truly re-usable and also because they are accessible to most of the popular development tools available for Windows. We examine the .NET platform as the basis for future development.

EPICS AT ALS ON WINDOWS

Migration to EPICS

The Advanced Light Source (ALS)[1] control system has been in service since 1991. At that time, it was based on small I/O controllers built in-house. At this time, the control system is gradually migrating to EPICS[2] with IOCs replacing the original controllers. The original system[3] is still in use in various subsystems including all the injector controls, but enhancements to the accelerator are typically accompanied by upgraded control hardware and software. At this time, most of the storage ring magnet controls have been moved to IOCs. Automated controls such as orbit feedbacks have been supported by Matlab programs with the access to EPICS channels which will be covered by another paper[4]. Here we focus on the Windows based operator consoles where the graphical and interactive programs run for machine operations. These programs were developed over a decade in various languages for control of specific machine functions and must be migrated to EPICS or replaced.

Simple Channel Access (SCA)

The original control system applications were built using a basic application program interface (API) to get and set accelerator data. The design goal was to hide the complexity of callbacks and other asynchronous mechanisms typical of distributed programming from the user. In migrating our routines to EPICS channel access (CA), we made use of a development effort that was already in progress called Simple Channel Access[5] (SCA) which had an API that presented a simpler interface than native CA but also took advantage of the strengths of CA as a high performance and efficient network protocol.

SCAcom

SCA for windows was originally deployed as a Windows dynamic link library (DLL) which meant that nearly all Windows development tools (and even Microsoft Office applications) could access accelerator data. However, each of these applications also has their own syntax for accessing a DLL creating a management. An ActiveX control, SCAcom[6], was created to help this situation. An Active X Control is a DLL with extra information about its contents that is registered with the operating system. This extra information can be used by Active X *container* applications to discover the capabilities of the library (such as the names of the routines and types of parameters). Since most Windows based applications have container capabilities, programs such as Visual C/C++, Visual Basic, Borland Delphi and C++ Builder, and Labview, can access SCAcom routines using this container mechanism. At this time, SCAcom is used by most all of the Windows applications involved in machine and beam line controls.

One optimization for requesting accelerator data is noteworthy. Often, especially for passive status displays, it is efficient to *group* requests for data (this is a capability fundamental to CA) rather than to issue a synchronous request for each item one at a time. SCAcom provides a mechanism to toggle between these two access methods depending on the requirements of the application.

Operation of the accelerator and migration from the current control system to EPICS, has now become critically dependent on the availability and performance of SCAcom.

SCAitem

A *process variable* (PV) in CA is a string used to uniquely identify data items. This addressing is also used by SCAcom. For example, here is how to register and access the item named " SR13C_QF7__AC17".

```
scacom.addDoubleItem("SR13C_QF7__AC17");
double SP=scacom.getDouble("SR13C_QF7__AC17");
```

Here scacom is an object of the SCAcom class. Note that items are looked up by the string representing the PV (item) name for simplicity.

When the number of items increases, the use of strings for lookup has becomes an issue. Therefore, on a client side, a class (not a component) called SCAitem is created to hold the name as a data member.

With the SCAitem object, scaitem in this case, the above example becomes:

```
scaitem.initAsDouble("SR13C_QF7__AC17");
SP=scaitem.getDouble();
```

In addition to the use of SCAitem, its collection class is always created wherever multiple items are accessed as a group.

*Work supported by the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

1. H_Nishimura@lbl.gov. 2. CATismossi@lbl.gov.

SCAS

SCAS[7] is an ActiveX control providing the functionality of a CA server but with basically the same interface as SCAcom. It has been used to give the CA server functions to application programs that are locally controlling the devices. For example, Labview programs locally controlling the beam line devices become accessible from EPICS by SCAS.

USING .NET

The .NET Framework

.NET is a major new framework that is destined to be the default development and application platform for Windows. .NET and development languages such as C# have been available for some time for Windows 2000 and are included with all new versions of Windows XP. Although current applications will continue to run unmodified, whether or not they are built using the .NET framework, we would like to investigate the potential benefits and performance implications of the new framework.

The concerns are:

- Compatibility with SCAcom.
- Runtime performance.
- Development tools including libraries.

Here we focus on the first two items.

In contrast to the *managed* world of .NET where an application space is subject to security policies set at a higher level, all the existing libraries and components that use the Win32 API directly are classified as *unmanaged.* There are different ways that can be used in deploying SCAcom to handle crossing the managed/unmanaged boundary. We wanted to compare three options for deploying SCAcom looking for performance differences. First we examined our current configuration leaving SCAcom as a Win32 ActiveX control (since ActiveX controls are backwards compatible with .NET). Second, we imported the control into a .NET wrapper class. Finally, we wrote a new .NET class (SCAnet) as a native .NET class that supports the old SCAcom interface.

SCAcom on .NET

As mentioned above, ActiveX controls are backwards compatible with .NET. in the sense that, once registered with the OS, they are immediately accessible to the .NET platform. In order to check the runtime performance, we set up the following test case.

We first measure the performance of a client (using SCAcom) communicating with a server (using SCAS) where both applications run on the same machine.

The server is a simple Win32 C++ application built on SCAS fielding 540 PVs to emulate the storage ring magnet analog values. The length of the names is 19 characters.

Two test client programs were created; one with C++ using SCAcom on Win32, the other with C# by importing SCAcom to a .NET class (it is straight forward to import an ActiveX controls to a .NET application as long as the parameters are all primitive types). This work was done in Visual Studio 2003 by using the automated tools of the environment.

For the speed measurement, clients read 540 PVs as doubles as a group ten times consecutively. This test was repeated 20 times to get an average speed. Table 1 shows the result.

Table 1: Time to read 540 double items from the local dummy server.

	Win32	.NET
SCAcom	20.8 msec	21.8 msec

This measurement was done Windows 2000 running on a PC with 2.4 GHz Pentium 4 dual CPUs. The result shows that runtime performances of both applications are essentially identical.

SCAnet

SCA has been in development for many years and while this development effort has paid off in performance, it has also resulted in some code bloat. We decided that with C#, we could create a .NET class that could be a very thin layer over CA and still support the familiar SCA interface. With this new class we call SCAnet, we hoped to achieve reductions in the size of our SCA code base by using the higher level language features of C#. We also wanted to enhance .NET compatibility by deploying a native .NET class rather than an imported Active X control. Finally, we hoped to make improvements in data access by looking at different strategies for calling CA. On the other hand, we also realized that by calling into the CA DLL directly, instead of through the SCA DLL, there would be potentially some loss in performance due to the more frequent calls from managed to unmanaged code.

SCAnet uses two C# modules. The first module, CA.cs is a very thin layer between the unmanaged code of ca.dll and .NET. The second module, Sca.cs, implements the SCA interface.

Although there are no fundamental problems developing a layer between managed and unmanaged code, it is still a tedious exercise. C# is able to call into DLL's but special care must be taken to deal with pointer arguments. In dealing with these pointers it is crucial to understand to which areas in memory these pointers refer and use the appropriate .NET system calls to either marshal values across the managed/unmanaged boundary or to directly dereference the pointer using the 'unsafe' keyword.

We tested the performance of the same application getting 540 PVs as doubles using SCAnet. The result shown in Table 2 indicates about a 50% increase in access time. However, it should be noted that this result includes the effect of the server running on the same machine although the PC has two processors.

Table 2: Time to read 540 double items from the local dummy server.

	Win32	.NET
SCAnet	N.A.	32.3 msec

PERFORMANCE OVER THE NETWORK

Finally, we make a real access to the control system. In this configuration, the CA servers are now multiple IOCs with embedded processors running the VxWorks OS. On the 2.4 GHz PC we read 540 of analog values from the IOCs controlling the magnets (Table 3). The performance on the .NET is 25% slower with SCAcom but the same with SCAnet.

Table 3: Time to read 540 double items from the real control system on the 2.4 GHz PC.

	Win32	.NET
SCAcom	16.4 msec	19.8 msec
SCAnet	N.A.	16.6 msec

We have also measured the performance on a 500 MHz PC used as a operators console. This case, as shown in table 4, SCAcom shows the same performance on both of the platforms. On the other hand, SCAnet works 3.5 times faster than SCAcom.

Table 4: Time to read 540 double items from the real control system on the 500 MHz PC.

	Win32	.NET
SCAcom	74.4 msec	76.6 msec
SCAnet	N.A.	20.4 msec

CONCLUSION

SCAcom has shown a good compatibility on .NET therefore ideal for the early transition period. A newly developed .NET component SCAnet demonstrated even better performance on .NET than SCA on Win32 in the real environment. These two components will allow smooth transitions of the control room application programs to the .NET platform.

AKNOWLEDGEMENTS

We thank D. Robin and A. Biocca at LBNL for their encouragements. We also thank C. Ikami for system management and Loren Shalz for SCA development.

REFERENCES (NOT COMPLETED)

[1] LBL PUB-5172 Rev. LBL,1986.
A. Jackson, IEEE PAC93, 93CH3279-7(1993)1432
[2] L. R. Dalesio, et al., ICALEPCS '93, Berlin, Germany, 1993.
http://www.aps.anl.gov/epics/
[3] S. Magyary et al, NIM A 293, p36, 1990.
S. Magyary, IEEE PAC'93, 93CH3279-7, p1811, 1993.
[4] G. Portmann, this conference.
J. Corbett, A. Terebilo, G. Portmann, IEEE PAC'03, 0-7803-7739-9, p2369, 2003
G. Portmann, J. Corbett, A. Terebilo, "An Accelerator Control Middle Layer Using Matlab Manual," to be published in IEEE PAC'05
[5] http://www-controls.als.lbl.gov/epics_collaboration/sca/
[6] C. Timossi and H. Nishimura, IEEE PAC'97, 0-7803-4376-X/98, p805, 1998
http://www-controls.als.lbl.gov/epics_collaboration/sca/win32
[7] C. Timossi, unpublished.