CROSS PLATFORM DEVELOPMENT USING DELPHI AND KYLIX*

J.L.McDonald, H. Nishimura and C. Timossi

LBNL, Berkeley, CA 94720, USA

Abstract

A cross platform component for EPICS Simple Channel Access (SCA) has been developed for the use with Delphi on Windows and Kylix on Linux. EPICS GUI application programs developed on Windows can run on Linux by simply rebuilding them, and vice versa. This paper describes the technical details of the newly developed component.

1 INTRODUCTION

The Advanced Light Source (ALS) control system has been continuously migrating from the original x86-based system [1] to the EPICS-based system [2], which involves numerous application programs written in various programming languages to be rewritten or updated seamlessly as background tasks. A component library is going to play a major role in this migration task [3].

The first library supporting EPICS SCA as a class at the ALS is JSca [4] in Java, which was followed by a C++ version, diverging to cover other languages including Delphi. Since, it has been remodeled as an ActiveX component, SCAcom [5] supports most of the Windows application programming in an organized manner. As the original control system does not use a grouped access to the devices, SCAcom implements easy switching of grouped and ungrouped access to encourage the migration. However, Linux is out of the class scope.

SCAclx is a port of SCAcom to cover both Windows and Linux in a same manner. It is a cross-platform component for the use with both Borland [6] Delphi 6 and C++ Builder 6 on Windows, and Kylix 2 and 3 on Linux. A program developed by one of them becomes portable including GUI and EPICS access on both platforms.

2 COMPONENT DEVELOPMENT

2.1 CLX Programs

Component Library for Cross-Platform (CLX) is a part of Borland development environments (Delphi, C++ Builder and Kylix) to generate applications that can be built and run natively on either Windows or Linux. All the non-EPICS parts can be programmed as CLX applications including GUI that becomes native on both platforms. Therefore, if an EPICS component becomes available on both of them, we can develop portable and native EPICS client programs. The programming languages for CLX programming are Delphi, a Borland version of Pascal, and Borland C++. However, we will focus on the Delphi language in Delphi 6 on Windows 2000 and Kylix 2 on Redhat Linux 7.3.

2.2 CLX and Native Routines

CLX programs are built only by using libraries that are portable on both platforms. When developing a CLX program in Delphi or Kylix environment, the component palette does not contain any non-portable component. However, this does not mean that a platform component cannot be used in a CLX program.

We can use any component or library routine in a CLX program sacrificing the portability. A CLX program on Windows can include ActiveX controls by manually importing and instantiating it. When these external components or routines are not written in Delphi, they will be imported as dynamic link libraries (DLL) on Windows, or shared objects (SO) on Linux. Here the ActiveX control is a special case of a DLL.

When a CLX component is to cover a function that involves calls to platform-dependent lower-level routines, there will be separate implementations and an EPICS call is one such case.

2.3 CLX Component to Wrap ActiveX Controls

The ActiveX control SCAcom has been in use for years, and considered to be well established. Therefore, SCAclx, a CLX port of SCAcom, uses the latter internally on Windows. We use the same source code QSCAclx.pas for SCAclx on both platforms by using the compiler directives {$IFDEF ..} and {$ENDIF} to distinguish the platform. Here is an example of unit imports at the beginning of the file to use different units depending on the platform.

```
unit QSCAclx;
 interface
uses
{$IFDEF MSWINDOWS}
  Windows,Messages,
  SysUtils, Classes, QControls, QStdCtrls,QExtCtrls,
  ActiveX,  SCACOMLib_CLX_TLB;
{$ENDIF}
{$IFDEF LINUX}
  SysUtils, Classes,
  libsharedsca;
{$ENDIF}
```

The ActiveX control SCAcom is written in C++ and the interface to it has to be in Delphi. If SCAcom has been already imported to the environment for the use in its

* This work was supported by the Director, Office of Energy Research, Office of Basic Energy Sciences, Material Sciences Division, U. S. Department of Energy, under Contract No. DE-AC03-76SF00098

Win32 programming mode, its Delphi interface file SCACOMLib_COM_TLB.pas will be available to be an ActiveX wrapper.

SCAclx contains a SCAcom object as its private data member and accesses it by using the wrapper SCACOMlib_CLX_TBL.pas that is a modified version of SCACOMlib_COM_TBL.pas. All the calls to SCAclx are forwarded to the SCAcom object.

The two extra steps on Windows are: (1) calls to CoCreate and CoUncreate at the proper moment, and (2) instantiation of the SCAcom object. The former is done in the creator and the destructor of the component. The second must be done prior to any SCAclx call, therefore it is managed by using an initialization status flag. An overall porting process is shown in Fig.1.

2.4 SCAclx on Linux

SCAcom is not usable on Linux since it is an ActiveX control. SCAclx has to be implemented on Linux at much lower level than using SCAcom. We created a plain C shared object library libscaclx.so in gcc to wrap SCAcom but not as a class. The unit that imports this library to the Delphi language on Linux is libsharedsca.pas. SCAclx uses routines through this import library on Linux.

2.5 Inside SCAclx

Each member function of SCAclx has dual implementation for the two platforms. Here is an example of getFloat.

```
function TSCAclx.getFloat(
                const pv_name: String): Single;
   var x:single;
begin
{$IFDEF LINUX}
  ErrorCode:=_getFloat(PChar(theGroupName),mode,
                    PChar(pv_name),x);
{$ENDIF}
{$IFDEF MSWINDOWS}
   x:=Sca1.getFloat(PChar(pv_name));
   ErrorCode:=Sca1.error_code;
{$ENDIF}
 result:=x;
end;
```

All the platform-dependent calls are done in this way by keeping the interface identical. Client programs can be entirely portable even including the project itself if the file paths and case dependencies are considered. An EPICS application program developed on Windows can be simply rebuilt on Linux and runs natively there.

We have ported about 80% of SCAcom public members to SCAclx and comfirmed on both platforms. It does not show any visible degradation of the run-time performance compared with that of SCAcom, which is the merit of native programs.

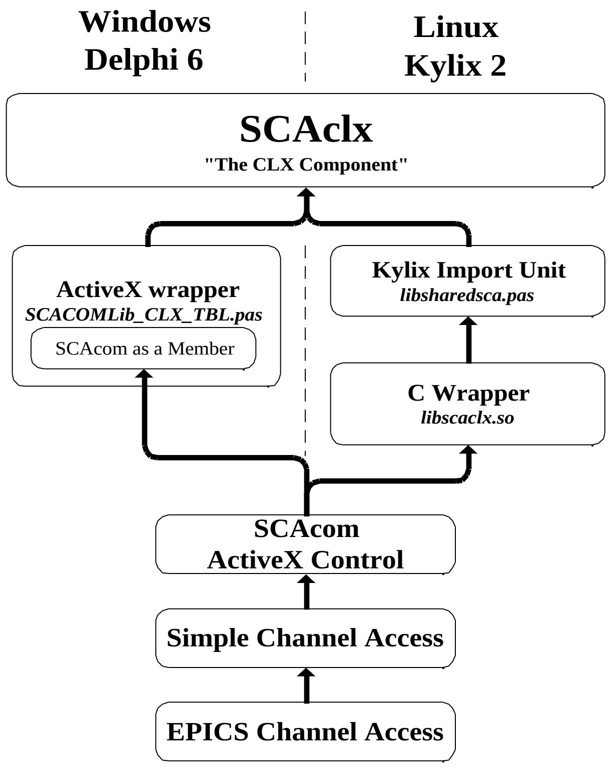


Fig.1 SCAclx Porting Process

2.6 Higher level classes

SCAclx is a class for an SCA access. It keeps multiple items initiated by a channel name (PVname). However, the channel itself is not supported as a class. A PVname is always required to get values from the channel. When the number of items is large, it becomes inconvenient to attach the PVname to each call. A higher level class is desired on top of SCAclx. SCAitem class is an example. It can be initialized with a PVname and can be accessed without it. The merit of using SCAclx is that these efforts at higher level can be completely portable.

2.7 Support for C++ Programs

Borland C++ 6.0 has been supporting CLX and the C++ version of SCAclx is in progress. The porting process is the same as that in Delphi. On Linux, Kylix has started supporting the Linux version of Borland C++ since Kylix 3. Therefore, SCAclx is just going to cover C++ in a similar manner.

3 APPLICATIONS

3.1 Basic Features

Let us show how to use SCAclx by using a simple example. It is found on the palette of the development environment. By allocating it on a Delphi Form, it will be declared in the program as:

SCAclx1 : TSCAclx;

This should be linked to a particular PVname at the beginning of the execution, for example, in the FormCreate routine:

SCAclx1.addDoubleItem(‘cmm:beam_current’);

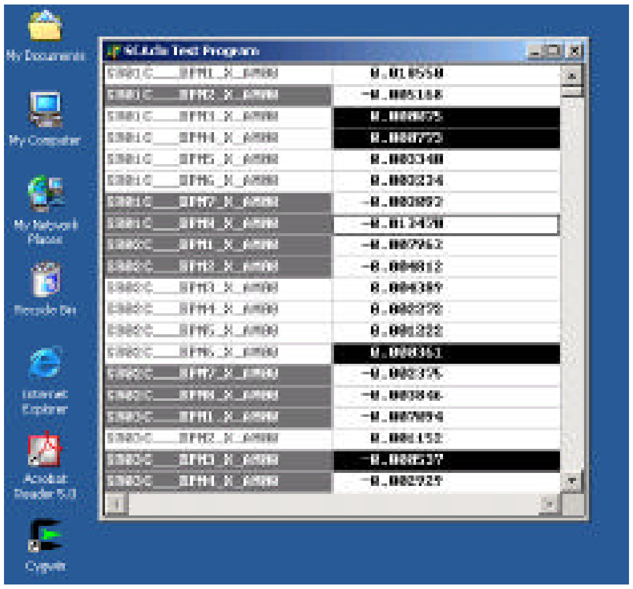
There are also methods to associate a PVname as float or integer. We have tested using more than 13,000 items to cover most of the EPICS channels at ALS. These values can be accessed, for example, as:

X:=SCAclx1.getDouble(‘cmm:beam_current’);

For efficiency, the grouped access becomes crucial when reading a large number of items. The scagroup property determines the group mode. We can seamlessly switch between grouped and ungrouped modes.

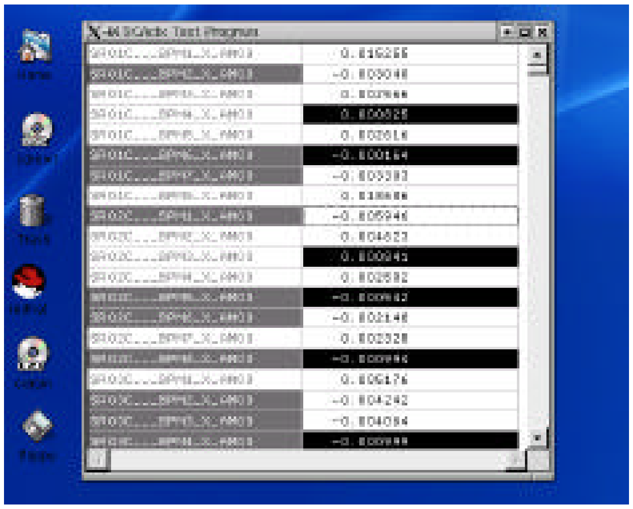
3.2 An Example on Both Platforms

A simple program was developed on Windows. It reads the beam position monitors via SCAclx and displays the values in a string grid. The cells of the grid change their colors depending on their values using a given formula. After confirming its function on Windows as shown in Fig.2, the program was rebuilt and run on Linux. as shown in Fig.3. During that process, no source code was changed.



```
SR01C___BPM1_X_AM00      0.010550
SR01C___BPM2_X_AM00     -0.005168
SR01C___BPM3_X_AM00      0.000075
SR01C___BPM4_X_AM00      0.000773
SR01C___BPM5_X_AM00      0.003340
SR01C___BPM6_X_AM00      0.003234
SR01C___BPM7_X_AM00     -0.003892
SR01C___BPM8_X_AM00     -0.012450
SR02C___BPM1_X_AM00     -0.007962
SR02C___BPM2_X_AM00     -0.004812
SR02C___BPM3_X_AM00      0.004389
SR02C___BPM4_X_AM00      0.000372
SR02C___BPM5_X_AM00      0.001322
SR02C___BPM6_X_AM00      0.000361
SR02C___BPM7_X_AM00     -0.002375
SR02C___BPM8_X_AM00     -0.003846
SR03C___BPM1_X_AM00     -0.007094
SR03C___BPM2_X_AM00      0.001152
SR03C___BPM3_X_AM00     -0.000537
SR03C___BPM4_X_AM00     -0.002929
```

Fig 2. A Test Program on Windows



```
SR01C___BPM1_X_AM03      0.016266
SR01C___BPM2_X_AM03     -0.009048
SR01C___BPM3_X_AM03      0.002966
SR01C___BPM4_X_AM03      0.000028
SR01C___BPM5_X_AM03      0.002616
SR01C___BPM6_X_AM03     -0.000164
SR01C___BPM7_X_AM03     -0.005383
SR01C___BPM8_X_AM03      0.010686
SR02C___BPM1_X_AM03     -0.005946
SR02C___BPM2_X_AM03      0.004623
SR02C___BPM3_X_AM03      0.000941
SR02C___BPM4_X_AM03      0.002592
SR02C___BPM5_X_AM03     -0.000842
SR02C___BPM6_X_AM03     -0.002146
SR02C___BPM7_X_AM03      0.002328
SR02C___BPM8_X_AM03     -0.000986
SR03C___BPM1_X_AM03      0.006176
SR03C___BPM2_X_AM03     -0.004242
SR03C___BPM3_X_AM03     -0.004094
SR03C___BPM4_X_AM03     -0.000988
```

Fig 3. A Test Program on Linux

4 DISCUSSION

There have been several cross-platform environments that are interpreter-based such as Perl, Python, LabView and Matlab. When the run-time performance is not severely required, or for a quick prototyping or simple tasks, they can be quite sufficient. On the other hand, once a run-time performance and GUI are both required, native programs has to be developed. SCAclx allows one to develop such programs effectively on both platforms. It is complementary to these interpreters.

5 ACKNOWLEDGEMENT

We thank to A. Biocca and N. Smith for their advices and suggestions, and to ALS operators for their useful comments.

6 REFERENCES

[1] S. Magyary et al, ‘The Advanced Light Source Control System,’ Nuclear Instruments & Methods in Physics Research A 293 (1990) 36-43, North Holland. S. Magyary, “Anatomy of a Control System; A System Designer’s View”, IEEE PAC93, 93CH3279-7,1811,1993.

[2] L.R. Dalesio, et al., “The Experimental Physics and Industrial Control System Architecture”, ICALEPCS ’93, Berlin, Germany, 1993.

[3] C. Timossi and H. Nishimur, “Accelerator Control Software Construction Based On Software Components”, PAC 1997

[4] C. Timossi, www-controls.als.lbl.gov/epics_ collaboration/ sca/win32/JSca/SCA.JSca.html

[5] C. Timossi, J. McDonald and H. Nishimura, “Experience with ActiveX Control for Simple Channel Access”, These Proceedings.

[6] http://www.borland.com