EPICS PORTABLE CHANNEL ACCESS SERVER FOR MULTIPLE WINDOWS APPLICATIONS

E. Tikhomolov, G. Waters, R. Keitel
TRIUMF, 4004 Wesbrook Mall, Vancouver, BC, V6T 2A3, Canada

Abstract

The RF control systems for the ISAC Radioactive Beam Facility are running on dedicated PCs under MS Windows operation system. For historical reasons they were interfaced into the EPICS based ISAC control system with a simple TRIUMF developed UDP/IP protocol. In order to improve the integration of the RF systems, it was decided to implement an EPICS Portable Channel Access Server (PCAS) on the Windows machines using the PCAS API provided from the collaboration in form of a WIN32 DLL. The main challenge for this implementation was the requirement to support multiple RF controls applications on a single PC. In addition, problems due the to different name decoration schemes of Borland C++ used in the RF controls applications and Microsoft C++ had to be overcome. The TRIUMF version of PCAS (TRPCAS) consists of a WIN32 DLL, which interfaces the RF applications to the EPICS DLL. An additional application uses the same DLL to provide information on all processes running. Shared memory is used to implement support for multiple RF applications. The API to TRPCAS is kept simple to avoid name decoration problems. TRPCAS uses XML configuration file. The advantages and limitations of different models of shared memory, the synchronization mechanism for local/remote control, and other details of the implementation are discussed.

INTRODUCTION

The integration of controls applications developed by different groups on different platforms is usually a challenging problem. At TRIUMF, the RF control sub-systems of the ISAC Radioactive Beam Facility are developed by the RF group. They must be integrated into the EPICS based central control system. The RF control sub-systems are running on dedicated PCs under the MS Windows operation system [1]. The EPICS system uses Sun workstations and Motorola MV162 VME based IOCs (input/output controllers). In the existing system, communication between an EPICS IOC and the RF sub-systems is provided by a TRIUMF developed UDP/IP protocol [2]. Figure 1 shows the components of the overall distributed system, including the Linux based operator consoles.

In EPICS there exists a framework for integrating non-EPICS control systems, the Portable Channel Access Server (PCAS). Under MS Windows PCAS is an application programming interface (API) distributed as WIN32 DLLs. By using this API it is possible to implement a control application, which runs on a PC and communicates with EPICS applications using the standard Channel Access (CA) protocol.

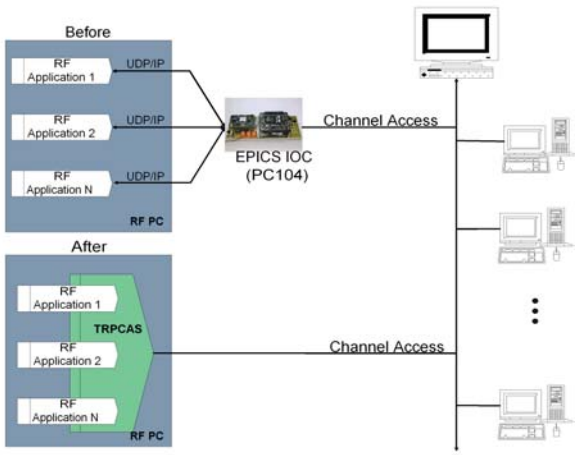

Figure 1: Run-time configuration of RF Controls System with TRPCAS

One limitation of PCAS is that only one server instance is allowed per internet node. In addition, the PCAS API, although very flexible, is non-trivial and puts a considerable burden on controls programmers who are not familiar with EPICS. We decided therefore to develop a TRIUMF version of PCAS (TRPCAS), which has asimplified API and which can support more than one instance of a controls application on a PC.

REQUIREMENTS FOR TRPCAS

The implementation of TRPCAS was based on the following requirements:

- Simple API for interface between Windows control applications (WCA) and TRPCAS
- Support of single and multiple control applications mode
- Easily configurable
- Can be switched between local and remote control modes
- Manages crashes of Windows control applications

IMPLEMENTATION AND ISSUES

TRPCAS was implemented as a single DLL that has a set of simple API functions for the Windows control application (WCA) developer. In addition it incorporates several diagnostic API functions.

TRPCAS API

TRPCAS had to address the fact that the RF control applications are developed using Borland C++, whereas the EPICS PCAS DLLs were implemented with Microsoft Visual C++. The difference of these tools is quite dramatic in their name decoration schemes. This makes it difficult to use some powerful features of C++, such as virtual functions, in interfacing to the TRPCAS DLL. The TRPCAS API was therefore implemented with C-like global functions.

The following API functions handle basic operations for the WCA developer:

- int TrPcasStart(). Starts the CA server. Returns status. If operation is successful status is zero, otherwise function returns error number.
- int TrPcasStop(). Stops the CA server and performs necessary clean up of resources. Returns status.
- int TrPcasRegisterPv (char * pName). Registers a process variable (PV) specified by name. Returns a handle to the PV.
- int TrPcasRegisterPvWithCallback(char *name, void (*f)(int, double)). Registers a PV and callback function associated with this variable. Any time when the PV is updated by the EPICS application, the callback function is called. Returns a handle to the PV.
- int TrPcasSetPvValue(int n, double *pvalue). Sets value using the handle of a registered PV. Returns status.
- int TrPcasGetPvValue(int n, double *pvalue). Gets a value using the handle of a registered PV. Returns status.

The following functions set and get values of process variable attributes, such as HOPR or LOPR, using a PV handle, the name of the attribute and the value as character string:

- int TrPcasSetPvAttr(int n, char *, char *value).
- int TrPcasGetPvAttr(int n, char *, char *value).

For setting remote or local control the following functions are used:

- int TrPcasSetLocal(bool mode). Disables writing to all PVs by EPICS applications.
- int TrPcasSetPvLocal(int handle, bool mode). Disables writing to a single PV by the EPICS applications. Returns status.

Single and Multiple Application Mode

In single application mode, the TRPCAS server DLL shares the data space of the WCA and the server is started transparently by the DLL.

In multiple application mode, shared memory is used by all applications which call the TRPCAS DLL. An additional server thread that uses this shared memory was implemented. Although a transparent server start would be possible in this mode, it was decided to provide a separate "server application", which also uses the TRPCAS DLL. This allows implementation of server diagnostics and keeps all WCAs symmetric. TRPCAS prevents possible mode conflicts and requires different PV names for each WCA.

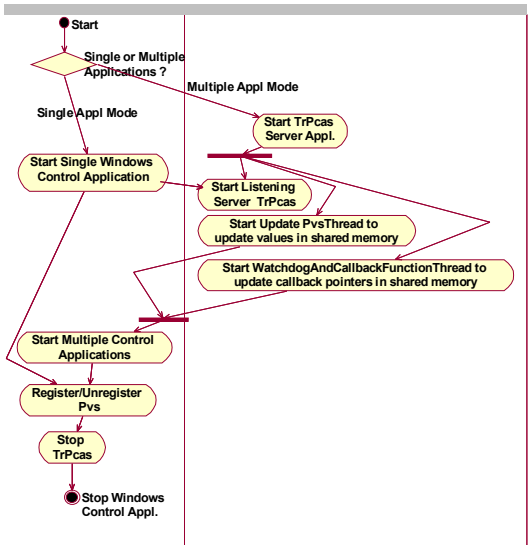

Figure 2: Starting TRPCAS in single and multiple application mode

Figure 2 shows that in multiple application mode additional threads for updating values and pointers to callback functions in shared memory are started.

Management of Shared Memory

We evaluated two methods of shared memory management: The first method uses pragma instructions to the linker. Its main advantage is that program modules can easily access variables placed into global memory, but the global memory is declared at compile time and therefore less flexible. The second method of shared memory management uses memory-mapped files. These can be created and deleted at run-time and are more flexible than the pragma-model. In this case, instances of classes can be easily serialized into shared memory.

TRPCAS was implemented using the pragma-model with a shared list of pre-allocated process variable data structures.

Remote and Local Control

The ISAC RF applications and most other WCAs require local control, i.e. from the Windows-based GUI at the PC location. In this mode it is necessary to disallow process variable changes from EPICS applications. TRPCAS implements this by keeping track of the origin of change requests. It is possible to disable writing from EPICS applications to individual or all PVs. Local/Remote switching is from the WCA GUI. Users at the EPICS consoles are informed about the mode change through a "remote/local" process variable.

Managing Crashes of Windows Applications

Each WCA registers its own set of process variables and deregisters them at the time of exiting. If a WCA crashes in multiple application mode, its registered process variables remain in an orphaned state. The TRPCAS server application addresses this situation by keeping track of all running WCAs with watchdogs. If a WCA crashes, its process variables are deregistered automatically by the server application after a timeout period. A crashed control application can therefore be restarted without closing TRPCAS and disrupting other live applications.

Figure 3 shows the console window of the TRPCAS server application window for the state when three control applications are running. The state of these applications is immediately visible from the watchdog counters. The console window also shows diagnostics channels that are included with TRPCAS.

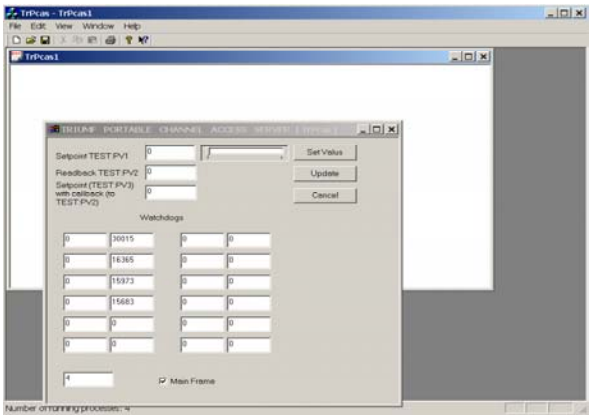

Figure 3: Console window of the TRPCAS server application

COMPONENTS

A TRPCAS user must install the following components on the Windows PC:

- EPICS DLLs: ca, cas, com, and gdd
- the TrPcas.dll which is the subject of this paper
- the TrPcas.exe server application (only needed for multiple application support)
- the files PvCfg.xsl and PvCfg.dtd which contain XML definitions
- the TrPcasCfg.xml XML configuration file defining the process variables, which are accessible by EPICS
- support components: MFC42.dll is used for error messages from TrPcas.dll, winsock32.dll for network support, and xerxes.dll for XML parsing.

This component set is shown in Figure 4.

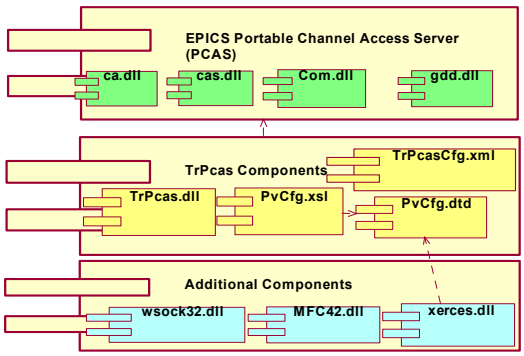

Figure 4: Components for running Windows control applications

Figure 5 shows the include and .lib files that are necessary for development of a WCA and for a TRPCAS server application.

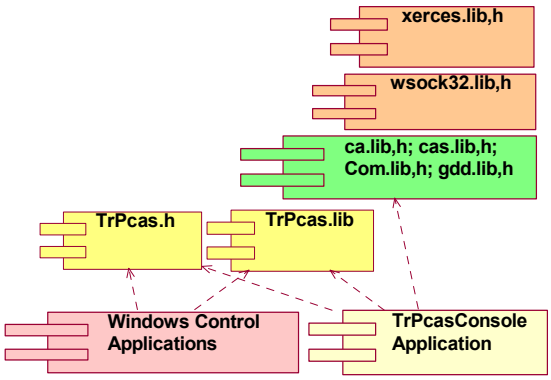

Figure 5: Components needed for development

SUMMARY

TRPCAS has been used on an RF cavity test setup for approximately one year since its first release. During this time, the only modification needed involved the XML configuration file. TRPCAS will be fully tested when the more than 40 RF systems of the ISAC-II facility are installed.

REFERENCES

[1] A. K.Fong, M. Laverty, S. Fang, "RF Control Systems for the TRIUMF ISAC RF," APAC'01, Beijing, September 2001, p. 642.

[2] R. Keitel et al., "Status Update on the ISAC Control System", ICALEPCS'01, San Jose, November 2001, p. 125

.