THE LABVIEW RADE FRAMEWORK DISTRIBUTED ARCHITECTURE

O. Ø. Andreassen, D. Kudryavtsev, A. Raimondo, A. Rijllart, V. Shaipov, R. Sorokoletov, CERN, Geneva, Switzerland

Abstract

For accelerator GUI applications there is a need for a rapid development environment to create expert tools or to prototype operator applications. Typically a variety of tools are being used, such as Matlab or Excel, but their scope is limited, either because of their low flexibility or limited integration into the accelerator infrastructure. In addition, having several tools obliges users to deal with different programming techniques and data structures.

We have addressed these limitations by using LabVIEW, extending it with interfaces to C++ and Java. In this way it fulfils requirements of ease of use, flexibility and connectivity, which makes up what we refer to as the RADE framework.

Recent application requirements could only be met by implementing a distributed architecture with multiple servers running multiple services. This brought us the additional advantage to implement redundant services, to increase the availability and to make transparent updates.

We will present two applications requiring high availability. We also report on issues encountered with such a distributed architecture and how we have addressed them.

The latest extension of the framework is to industrial equipment, with program templates and drivers for PLCs (Siemens and Schneider) and PXI with LabVIEW-Real Time.

INTRODUCTION

Integrating equipment and software in CERN accelerators like the LHC, which contains more than 1200 dipole magnets, more than 8000 other superconducting magnets operating at temperatures down to 1.9 K and with currents up to 12000 A is a great challenge [1]. When one also adds its size into the equation (27 Km) it doesn't ease the task of operating this marvellous machine.

With the RADE (Rapid Application Development Environment) framework and LabVIEW we have managed to ease the job of integrating new and existing projects into all the different disciplines at CERN [2]. With our distributed architecture the user does not have to update their RADE libraries, we maintain them on our server and ensure compatibility with new releases as long as the underlying protocol or library does not become deprecated or replaced.

ARCHITECTURE

In computer programming, a software framework is an abstraction in which software providing generic functionality can be selectively changed by user code, thus providing application specific software. It is a collection of software libraries providing a defined application-programming interface (API) [3].

LabVIEW already gives you the foundation needed to do fast programming by providing an integrated development, debug and deployment environment based entirely on graphical programming, as well for the code, as for the GUI development. In addition it gives you a powerful and flexible set of analysis and math libraries out of the box.

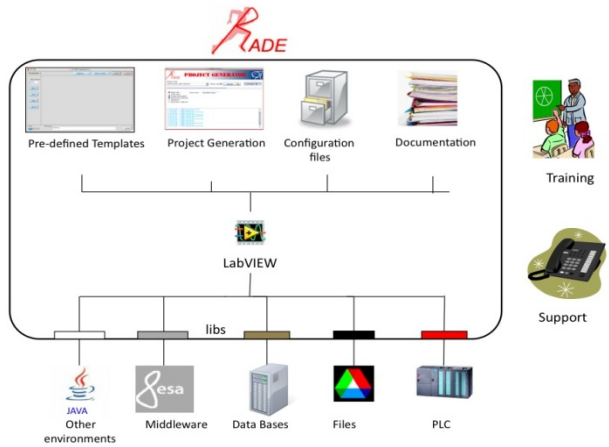

Figure 1: RADE Architecture.

Our development effort is thus concentrated on the integration into the CERN accelerator control and storage infrastructures, with an objective of making the deployment and maintenance of the framework transparent for the users, yet always up-to-date with the latest dependencies and additions.

Total Package

The RADE framework also aims to give users a total package for development, maintenance and support (Fig 1), making it quick to implement yet highly stable, maintainable and flexible through well defined development templates, guidelines and documentation.

Currently we are working on expanding this framework and do a "total integration" in LabVIEW at CERN.

Templates, documentation, source control, updates, and libraries are being added to the foundation as the testing of them finishes.

By adding a layer on top of LabVIEW we can synchronize the client installation with our Subversion (SVN) [4] and data repositories.

Project Generator

In addition a project generator (Fig. 2) has been created which can be called through the native menus of LabVIEW.

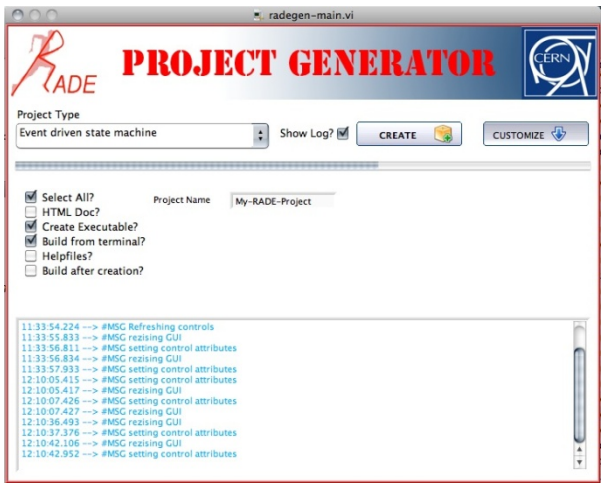

Figure 2: RADE Project Generator.

The Project Generator will create a skeleton project and folder structure for the user, and if desired store this project on our SVN repository dedicated for user projects. It also automatically generates documentation, executable files, help files, and integrates system and RADE components into the project so that if the user at a later point would like to update or retrofit his sources to a later version of RADE, he can do so.

RADE LIBRARIES

The RADE libraries consist of several communication layers (Fig. 3), adapted to the necessary protocols. It also makes use of a distributed architecture where several redundant servers host the communication and analysis libraries. These can potentially be written in any software language, but we mainly use the CERN Java packages and native LabVIEW servers that communicate with the LabVIEW client through sockets.

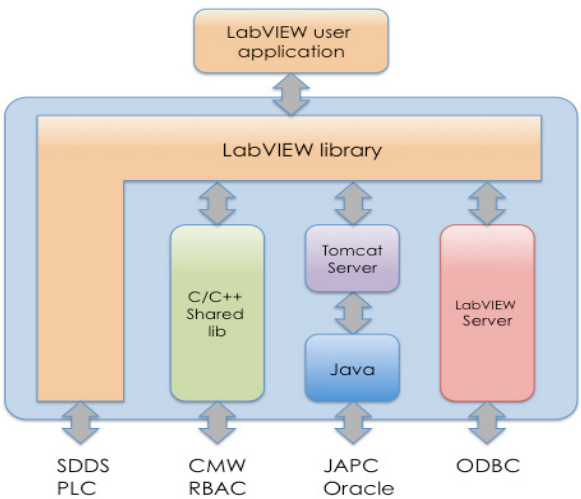

Figure 3: Library layers in RADE.

Java interface

The Java API for Parameter Control (JAPC) is a communication layer to control accelerator devices from JAVA [5]. Client programs can access JAPC parameters with set and get actions or by subscribing to the data. JAPC is a unified Java API for almost all parameters present in the control system. The diversity of devices is handled below the JAPC interface where each kind of device has its own implementation.

In the RADE framework we have implemented a LabVIEW to JAPC interface using a Tomcat server (Fig. 3) as mediator. The same mechanism is used to access ORACLE databases.

CMW Wrapper

The Common MiddleWare (CMW) API incorporates Set, Get and monitor actions like the JAPC API, but it is written in C++ and runs locally with the client [6].

We implemented the LabVIEW to CMW interface (CMW wrapper) by wrapping the C++ API into call back functions and turning it into a shared library, which is called by LabVIEW using the call library interface node.

The Toolkit was designed to be cross platform so the dependant libraries can be compiled on any platform supported by the CMW API (Mainly Linux and Windows).

SDDS Library

SDDS stands for Self Described Data Set [7]. This file format is commonly used at CERN and can contain any kind of data. This standard is used in the LHC to store equipment data (currents, voltages, levels etc.), used to inform the domain experts about the system health.

Since a failure in the machine can produce more than 10.000 files in a single dump, the SDDS library was designed to index and read only the requested portion of a file, improving reading speed and saving time when performing analysis.

PLC Library

The communication to the PLC is written in standard LabVIEW using the "Fetch-Write" protocol from Siemens through TCP-IP. The "Fetch-Write" has to be declared in the PLC to authorise external devices accessing its data blocs. For more flexibility we are developing a wrapper for LIBNODAVE [8], an open source library to access Siemens PLCs.

RADE IN THE LHC DASHBOARD

The LHC Dashboard project aims to provide a single entry point to the monitoring data collected from the experiments and equipment associated with the LHC.

The Dashboard collects information from multiple sources, treats the data and renders it as a set of web

pages. It covers various activities of the LHC making up a complete picture of what goes on in the machine. By using RADE in the dashboard we have managed to make a very flexible and powerful data-mining tool where we, with little effort, can obtain new data and display this for our users on demand. Currently development is ongoing using RADE for expanding the dashboard, making a fully flexible, user driven web interface where we can render any data from the various activities at CERN on request. The page will consist of a large library of gadgets where the users can select what to see in a customized view.

RADE IN HARDWARE COMMISSIONING

Hardware Commissioning is the phase where tests are carried-out by the specialized teams to qualify the individual systems of the LHC for operation (vacuum, cryogenics, quench protection, interlocks, powering, etc.). The interaction of each system and its partners are verified and the circuit will be powered up to nominal current [9]. This is repeated for every circuit. Commissioning the LHC requires powerful diagnostics to identify faults in the equipment protection systems. This diagnostics tool, called herein the post-mortem system (Fig. 5), has the role of grouping and analysing transient data recorded in the LHC equipment.

The Hardware Commissioning software architecture consists of three parts. The first is the Sequencer written in JAVA, executing the tests. The second is the data collection and storage. The third is the data analysis made with RADE and LabVIEW.

With RADE the Post Mortem Framework is highly flexible. It allows us to quickly adapt to changes and new demands, and validate them on a set of reference tests, when performing the crucial analysis of the LHC equipment. All RADE facilities have been used during the implementation of the Post Mortem analysis.

In 2011, 6092 tests where executed (Fig. 6) (3 week campaign). From these, 3204 where automatically approved by software using RADE.

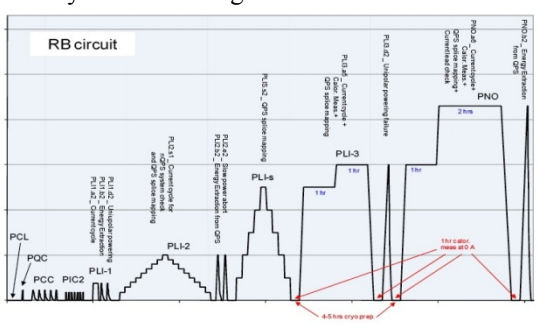

Figure 6: HWC Dipole Tests.

ISSUES

Having a distributed architecture is highly flexible. You can re-route clients to one server while performing updates on the other. On the other side one is vulnerable as well. If the server goes down, if there is a critical bug

Figure 4: The LHC Dashboard.

in the software running or by any other means a problem in the chain that blocks, all users are affected.

Since most of the software dependencies are developed and maintained by third parties, who again have dependencies on other repositories and machines, we have to make sure that the source running on the server is up to date and in synch with versions. Since the servers are redundant, users will transparently connect to the available source. This can be a problem since a

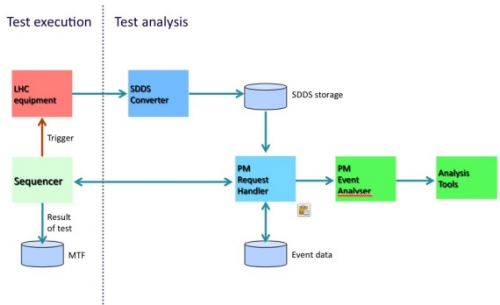

Figure 5: Post Mortem System.

connection or request that manages to block the server could potentially be re-directed to the secondary machine and block that one as well if executing the same request.

To cope with these issues we have to constantly analyze the system integrity and add logic, which identifies problems and prevents them from happening.

CONCLUSIONS

The flexibility and short development time experienced when using RADE in, amongst others, the presented project examples show that the RADE framework is an excellent and powerful tool that can be used to cope with challenges in an environment that quickly and constantly changes. One has however to take care when using a distributed architecture since introduction of new "features" or changes can quickly cause unforeseen problems that affect many users. On the other side the flexibility and convenience of having a distributed architecture trumps the downsides by far.

REFERENCES

[1] L. Evans, "The Large Hadron Collider – Present Status and Prospects", IEEE Trans. Appl. Supercond., Vol. 10 No. 1 (2000), 44-48
[2] A. Raimondo, "Rapid Application Development Environment Based on LabVIEW" - edms.cern.ch/document/904425/1
[3] Definition software framework http://en.wikipedia.org
[4] Subversion - subversion.apache.org
[5] V. Baggiolini, "JAPC-the java API for parameter control", ICALEPCS2005, Geneva, Switzerland.
[6] K. Kostro, "The control middleware (CMW) at CERN status and usage", ICALEPCS2003, Gyeongju, Korea.
[7] R. Soliday, "New features in the SDDS tool kit", PAC2003, Portland, Oregon, US.
[8] LIBNODAVE, "Exchange data with Siemens PLCs" http://libnodave.sourceforge.net
[9] R. Saban, "LHC Hardware Commissioning Summary", EPAC08, Genoa Italy.