ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA185

# RADAR 2.0, A DRAG AND DROP, CROSS PLATFORM CONTROL SYSTEM DESIGN SOFTWARE

O. O. Andreassen, J. W. Rachucki, R. M. Knudsen, CERN, Geneva, Switzerland

*Abstract*

In the ever-growing control system at CERN, there is a need for having an easy to use, yet fast and flexible tool that interfaces with all the different middleware and communication interfaces in the accelerator, experiments and technical infrastructures. With RADAR 2.0 we wanted to address this issue, making a LabVIEW based, drag and drop visual tool that hides much of the system complexity from the user and within seconds gives the operator a ready to use, fully functional control system GUI. RADAR 2.0 interfaces with the CERN Middleware (CMW), the CERN Accelerator and Logging system (CALS), OPC-UA and DIM. With its class based implementation it can easily be extended to other data sources (Files, Databases, middleware) on demand. This paper reports how the implementation was done, the architecture, underlying technology and an outlook to other possible applications.

## MOTIVATION

Getting the knowledge needed to properly analyse or interact with the Large Hadron Collider (LHC) control system can be a lengthy process which in many cases requires years of experience. The information on where to connect and how to interact with devices is often domain specific knowledge, spread across many different equipment experts and teams. In addition, many of the temporary users and operators at CERN are professors, students and collaborating partners that only stay for a short time, which can make interacting with the infrastructure quite an undertaking [1][2].

Many initiatives have been done to consolidate and reduce the diversity of the control system, which has reduced the complexity and eased access over the years, but it is still not straight forward to retrieve information regarding device configuration, calibration, response and behaviour, especially in cases where equipment have either direct or indirect correlations [1][3][4][5]. RADAR 2.0 tries to address these issues by combining all the accelerators devices and data sources in one interface and automatically detect, and adapt to the specific interface of the equipment. In addition, the components created with RADAR 2.0 can be saved, customize and shared across projects and users. This, over time, aims to ease the effort of mapping equipment specificities, allowing the operators to focus on commissioning and running the machine.

## BACKGROUND

The application core of the RADAR 2.0 Drag and Drop toolkit was derived from two other applications called UNICOS in LabVIEW (UiL) [1][6] and RADAR 1.0 [7].

### *UNICOS in LabVIEW*

UiL was introduced as a lighter, less expensive alternative to UNICOS [1]. For some specific projects (small or initial prototypes not connected to accelerator operation or located outside CERN) a more customisable supervision application using LabVIEW was an attractive alternative. UiL provides a set of customisable re-usable components, devices and utilities [1].

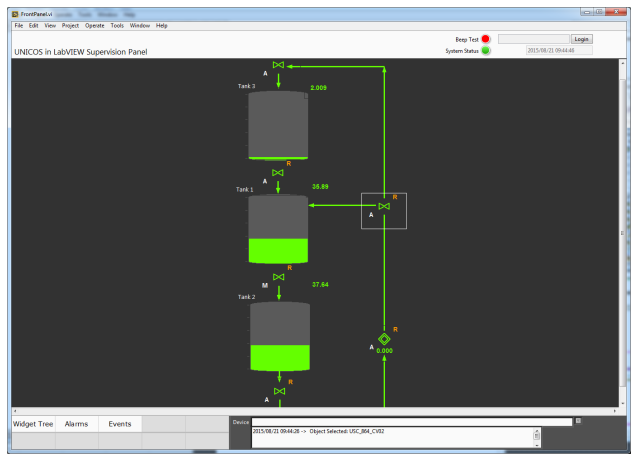


Figure 1 : UiL Example Application.

### *RADAR 1.0*

RADAR was initially developed as an extension to the CERN Open Analog Signals Information System (OASIS) which is a highly configurable virtual oscilloscope that makes it possible to configure acquisition hardware with wide analogue bandwidth and flexible distributed triggering schemes across the CERN accelerators [2].

RADAR later evolved beyond interfacing only with OASIS, and became a connectivity tool that interfaced with all CERN devices using the CERN Middleware (CMW) [2][8].

## RADAR 2.0

RADAR 2.0 consists of several stand-alone modules which intercommunicate. The main parts are described in the architecture section below. There are two modes to the application: Development and run. In development mode, the user interfaces with a dedicated infrastructure which can be deployed on any server. In run mode, the application acts as a stand-alone executable which either interfaces with the CERN and RADAR infrastructure or directly with the dedicated devices.

### *Workflow and Development*

The users' starting point when developing with RADAR 2.0 is the project generator pane. The user is presented with a login screen (See Figure 2) where he or she

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA185



navigates to the personal workspace, selects an existing or new project and location where to save. If an existing configuration is selected, the connection information is either retrieved from a dedicated configuration file or the user database, depending on what the user selects.

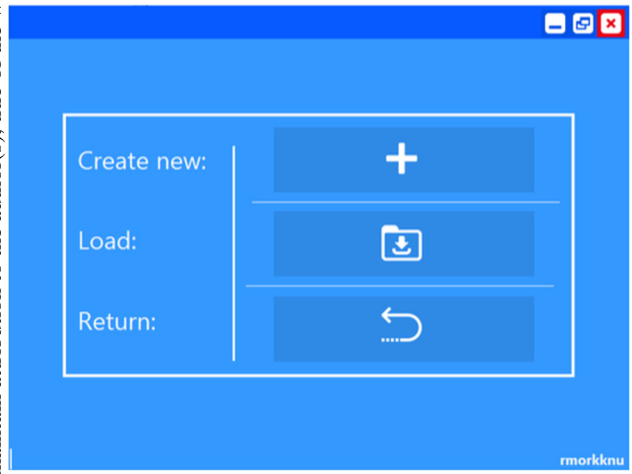


Figure 2: Starting the Application.

Building an application is drag and drop based. In the RADAR 2.0 Project window, the 'Project Manager' contains all object types, organized by accelerator domain, as well as other custom assemblies defined by the user.

During the creation and configuration of the widgets, all items are given a type identifier (based on a GUID), which is used internally by the RADAR 2.0's scripting engine. This acts as a hook used by the application for all animations and user interactions with the widget.

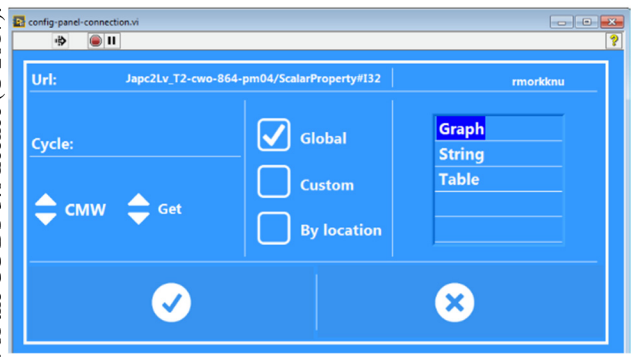


Figure 2: Device Configuration Panel.

To add specific devices into a front panel, the user drag-and-drops a widget onto the Front Panel which triggers a window where the user can select the object the widget must be linked to, choose an object representation (Graph, text, table etc.), and what device reference to connect to (Figure 3). Once configured, the widget is animated on the applications front panel.

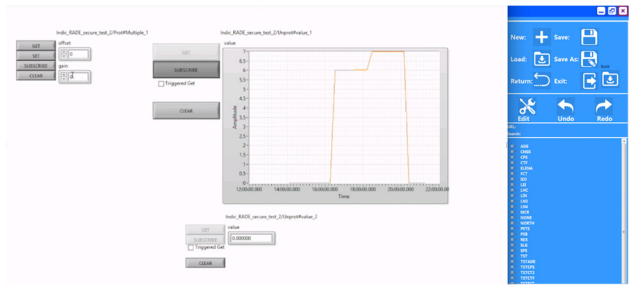

Figure 3: RADAR 2.0 Development Workspace.

## *LabVIEW Scripting*

All the generated objects in RADAR 2.0 rely heavily on LabVIEW VI Scripting. LabVIEW VI Scripting helps you programmatically generate, edit and inspect LabVIEW code. It gives you access to LabVIEW's internal server classes, properties, and methods so you can programmatically create, move, and wire objects within the graphical environment [9].

## *Architecture*

The architecture is realised using several commercial off-the-shelf software toolkits such as LabVIEW, LabVIEW Datalogging and Supervisory Control Module (DSC), REDIS and NoSQL [10][11].

The general module architecture can be seen in Figure 4. The application is built around a client, which subscribes, caches and re-serves data throughout the application using CERNs middleware (CMW) as a peer to peer communication layer. The most resent updates are stored and broadcasted through an internal notifier, which can be read (by reference) in any sub module of the application. The larger data sets (arrays and vectors for trends and graphs) are stored in an internal queue, and indexed by object type identifiers.

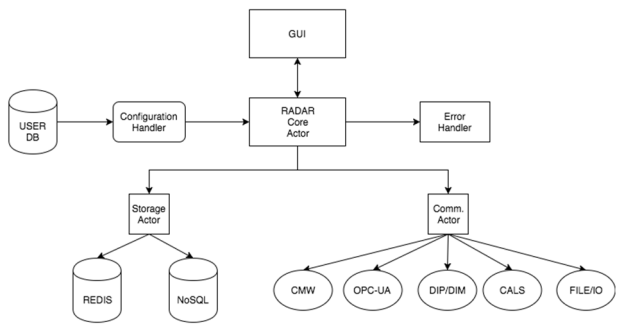


Figure 4: RADAR 2.0 Architecture.

## *Device Groups and Components*

A device group or specific custom made components can be saved as a widget and re-used by all users. Once a configuration has been made, the user can export this design to a custom SQL database which can be set up to facilitate custom grouping and access. The current implementation of RADAR 2.0 supports standard MySQL and mariaDB databases, but could easily be extended to other storage forms if needed [1][5][10][11].

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA185

### *Communication*

The communication between RADAR 2.0 and the CERN infrastructure is based on an extendible virtual class that support any add-on such as CMW, OPC UA, and JAPC via the CERN Rapid Application Development Environment (RADE) framework (See Figure 5) [7].

The internal application data is serialized to an internal common syntax. The data can be converted to string, Time/Value based data or LabVIEW clusters.

RADE is a platform- independent, service-oriented architecture specification that allows the exchange of data in the industrial automation space [7].

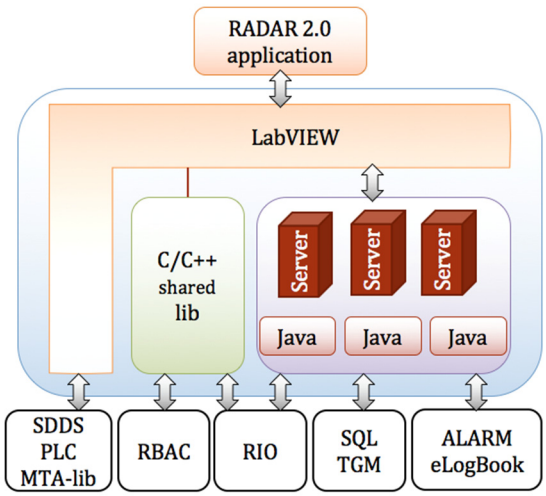


Figure 5: Communication.

### *Data Flow*

Data in RADAR 2.0 is primarily stored and accessed through the use of internal LabVIEW notifiers. For trends, the data is queued, in order to accurately display all data from any device without losing data points in the graph. The trends and events also access the REDIS Cache and NoSQL backend for the viewing of historical data (typically with a backlog of one week).

### *Archiving*

RADAR 2.0 has the capability to archive all live data and settings. When the user selects this option in the configuration for a particular device, the data is automatically cached and stored. RADAR 2.0 archives its data locally in MongoDB, a cross-platform document-oriented database.

In addition, a central cache is stored in a REDIS backend which makes it possible to interconnect several instances of RADAR 2.0 without duplicating the device connection.

MongoDB eschews the traditional table-based relational database structure in favour of JSON-like documents with dynamic schemas, making the integration of data in applications such as RADAR 2.0 easier and faster [7][10]. The method used to archive data is class based and intended to support multiple sinks through class overrides in the future.

### *Data Caching and Publication*

The REDIS and Database caching used in RADAR 2.0 has a built-in added feature that the data one client subscribes to can be re-published either as a new CMW publication or as a REDIS, reducing the load and connections used on the front-ends. The RADAR 2.0 connection backend only lets one device connect to a source (unless the sampling frequency differs) [1][7].

## VALIDATION

The initial development was tested in terms of connectivity with 30 different real devices, testing interfaces across different front ends and databases (from the CALS service).

In addition, a few hundred simulated virtual channels were used to test scalability. The test machine was a standard HP Compaq 8000 elite with 4Gb of RAM running Windows 7 64bit. The RADAR 2.0 LabVIEW instance was running in 32bit mode. The test bench had 200 active channels to the testbed and all the data was logged to its MongoDB and REDIS archive. No significant increase in CPU (was at 30% when starting and stayed at 35% after the application ran) or Memory (used 200MB at start and had 350 Mb after 48h run) was noticed while running the test. The test also provoked a random alarm every 30 minutes on random channels, and stored all the data while running. The update frequency was set to maximum 1 point every half second, fixed.

The results in terms of animation and refresh rate were satisfactory, having several dozens of devices in the same panel refreshing continuously. Interaction with the devices (via dedicated buttons) in the widgets associated to the different devices was also smooth and efficient.

The current RADAR 2.0 release only supports a single database archive per application instance. This limits the cross-application interaction although this will be addressed in next releases.

## MAINTENANCE AND UPGRADES

Maintaining the underlying communication libraries and dependencies of RADAR 2.0 will depend on the release cycle of the RADE framework. Compiled binaries will continue to function when using the 2 tier RADE Service bridge, but direct connections (applications communications without going through the RADE Services application bridge) will have to be re-compiled whenever there are changes in the API. We plan to do this partly automatically trough the automated build system used for our LabVIEW applications [7].

## CONCLUSION

The current release of RADAR UiL has been successfully deployed and is being used as a prototype at CERN. The combination of LabVIEW's intuitive drag and drop-based interaction, together with the similar look and feel of LabVIEW makes RADAR 2.0 a good choice for small to medium sized control and supervision applications.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA185



RADAR 2.0 has all of the core LabVIEW and RADE features, but hides much of the overhead development effort through its drag and drop interface. This makes it possible to make anything from quite simple monitoring interfaces in mere minutes while still having the possibility to extend and expand the application to much more advanced applications.

RADE was used as the communication layer between LabVIEW and the CERN infrastructure, which means that whenever an interface is added to the RADE catalogue, it becomes readily available to the RADAR 2.0 users, ensuring connectivity and compatibility in the future.

Our performance tests show that RADAR 2.0 can handle several hundred connections running simultaneously without any significant load to the CPU.

The main drawback with the current design is its device name based interface. If a device name changes both in name or in nature, the user will have to re-create these objects by re-defining the interface. We hope this will be solved by sharing widgets in the "community" database backend, meaning that once someone has adapted/changed one interface, other users will benefit from this change and inherit the new interface in their applications.

## FUTURE PLANS

A consolidation phase is the first step to make the RADAR 2.0 a solid and deployable solution. Up to now all the concepts have been validated during the development and test phases. The plans include the development of a full catalogue of shared widgets for most of the popular sensors, actuators and control devices at CERN. The alarm and event managers will incorporate filtering capabilities and the diagnostics of the front-ends will be properly interpreted and shown to the user.

Support of future connections will be added, but we still have to decide if this will be done at the local server level or with the RADAR 2.0 client connecting to multiple servers as a service.

Cross communication between different instances of RADAR 2.0 backends still needs some improvements in terms of different sampling rates and use cases (on change or on demand)

We are also looking into adding CERN services in future releases, these comprise the interface with LASER, the CERN central alarm system, the Post Mortem framework and many of the various CERN databases.

With the current class based data subscription model and in combination with the RADE framework, this could be done without much effort [7].

## REFERENCES

[1] O. Andreassen et. al," LabVIEW as a new supervision solution for industrial control systems" ICALEPCS (2015), Melbourne, Australia, Paper MOPGF115

[2] Trofimov N et al "Remote Device Access in the new CERN Accelerator Controls middleware" ICALEPCS 2001 (San Jose, California) 2001, Paper WE201.

[3] Zaharieva Z et al "Database Foundation for the Configuration Management of the CERN Accelerator Controls System" ICALEPCS'11 (Grenoble, France) October 2011, Paper MAU004.

[4] Bagiollini V et al "CERN PS/SL Middleware Project, User Requirements Document" CERN Note SL/99-16(CO) Issue 1 Revision 3 Geneva, Switzerland August 1999

[5] A. Dworak et al. "THE NEW CERN CONTROLS MIDDLEWARE", CHEP 2012, New York, USA Conf. Ser. 396 012017.

[6] P. Gayet et al. "UNICOS a framework to build industry like control systems" ICALEPCS 2005, Geneva Switzerland, WE3A.2-6O.

[7] O. Andreassen et al. "The LabVIEW RADE framework distributed architecture", ICALEPCS (2011), Grenoble, France, Paper WEMAU003.

[8] S. Deghaye et al. "A new System to Acquire and Display the Analog Signals for LHC", ICALEPCS 2003, Gyeongju, Korea, WP502

[9] LabVIEW Scripting webpage http://sine.ni.com/nips/cds/view/p/lang/en/nid/209110

[10] K. Banker "MONGODB IN ACTION", p. 375, ISBN 9781935182870, 2011

[11] W. Mahnke "OPC UNIFIED ARCHITECHTURE" OPC UA, (2009), https://library.e.abb.com/public/75d70c47268d78bfc125762d00481f78/56-61 3M903_ENG72dpi.pdf