ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA129

SYNCHRONISING LabVIEW DEVELOPMENT AND DEPLOYMENT ENVIRONMENT

O. Ø. Andreassen†, C. Charrondière, H. Reymond, A. Rijllart, M. Miskowiec
CERN, Geneva, Switzerland

Abstract

LabVIEW™ with its graphical approach is suited for engineers used to design and implement systems based on schematics and designs. Being a graphical language, it can be challenging to keep track of drivers, runtime engines, deployments and configurations since most of the tools on the market aimed towards this are implemented for textual languages. Configuration management is possible in the development environment via version control systems such as Perforce™, however at CERN and in the open source software development community in general, the tendency is moving towards Git. In this paper we demonstrate how the combination of automated builds, packaging, versioning and consistent deployment can further ease and speed up development, while ensure robustness and coherency across systems. We also show how an in-house built tool called "RADE Installer" synchronises both development environments and drivers across workstations, empowering graphical development at CERN, by merging the open source toolchains with the workflow of LabVIEW. RADE installer represents a solution for LabVIEW to keep track of drivers, runtime engines, deployments and configurations.

BACKGROUND

The LabVIEW™ based Rapid Application Development Environment (RADE) was conceived as a result of an increasing need to quickly prototype and release controls, analysis and test-bench tools in the CERN accelerator domain (Fig. 1). The framework was designed to reduce the traditional development time without compromising the applications' stability and longevity. RADE's multi-tier, plugin-based architecture made it possible to develop simple control applications in hours and at the same time maintain them for years [1].

As the framework grew, so did its dependencies and complexity. Adding a change to the framework typically would take a day to release. With several changes being added every day, it could take weeks to months before they were distributed, depending on priority and available resources. This led us to invest in build automation and Continuous Integration (CI) [2].

With the introduction of CI, the release process was reduced from one day to about 1 hour (53 minutes) unattended and automated, freeing up the development time.

In addition, automating the tasks removed typical "operator errors" from repetitive work and made it possible to introduce new toolkits in the framework continuously. An added bonus from adopting CI was also the early feedback it gave. Since all unit test is executed on every release, the developer gets immediate feedback if a change broke a module and can start working on solving the issue at once [3].

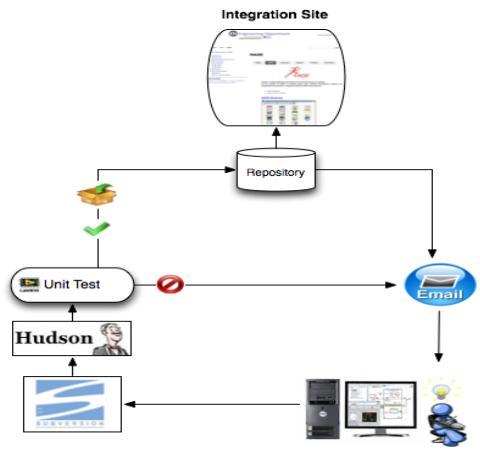

Figure 1: RADE Continuous Integration Process early version.

When the libraries have passed all the unit tests, the CI process builds and bundles the libraries in to several single target specific installers.

Reducing the release time made it possible to integrate the latest changes and any new features immediately. This change in the build philosophy worked well for libraries and applications the development team controlled, however it made it challenging for external users to be in synch with the constant change, especially if the release introduced changes to the dependent interfaces that where not backwards compatible. This led us to rethink the strategy of bundling all RADE libraries in to one big versioned installer and was the main drive behind investing efforts in to developing the RADE installer [3][4].

THE CHALLENGE

Version control and tools that automatically updates software is nothing new in the development world. There are hundreds if not thousands of different tools available on the market. The LabVIEW community even has two tools that is aimed against installing third party libraries, however CERN's unique infrastructure and platform requirements are limiting factors in what one can choose [5].

Selecting the Right Tools

There are several considerations one has to take when integrating network-based components in the CERN infra-

† odd.oyvind.andreassen@cern.ch

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA129

structure. There are exotic configurations, (Some components of the accelerators date back to the 60's and the infrastructure, though renewed, in portions reflect the early decisions made at the time), vast network with more than 150.000 devices (as 2019) and there are multiple layers of private and public networks you have to cross to get from one point to another [6][7].

As a result of that we had to make some considerations when selecting the right tool for the job. In our lab and in the capacity as a support team, we needed to have a tool that could aid in distributing packages that would suit our users, at the same time be compatible with the chosen standard of NI and LabVIEW: Using the NI Package Manager. We wanted a tool that would:

- Add value to both the team and our users
- Run on CentOS, Windows and Mac OS
- Be a well-known and supported tool
- Handle cross installation of applications, drivers and libraries regardless of language and environment
- Support different package types
- Call external resources and use third party installers, not being limited to a custom or dedicated environment
- Configurable per project
- Be able to proxy connections and work across all internal networks via dedicated trusted nodes
- Work as a binding agent between other existing solutions
- Work with a GUI interface, callable from LabVIEW (both indirectly or directly)
- Work on future versions of LabVIEW

Table 1: Evaluation of Package Managers

Name	License	Local Server	Cross Platform	Mixed Language
VIPM [8]	Paid	Paid	Yes	No
NIPM [9]	NI	Yes	No	Yes
OPKG [7]	MIT	Yes	No	Yes
Yum [7]	GPLv2	Yes	No	Yes
Pip [7]	MIT	Yes	Yes	No
Dpkg [7]	GPLv2	Yes	No	Yes
Conda [7]	BSD	Yes	Yes	Yes

After evaluating several different commercial and open source products (see Table 1), it would initially seem like many of the package managers on the market could be suitable for our needs, however any cross-platform tool that was close to offering what we wanted, either would add a higher cost than the added value or it would be tailored to its dedicated environment and therefore not suitable for the job [7-9].

After a few internal review rounds, evaluating tools and looking at the results forming from the data, we concluded that the best path for us to go, fully being able to control and adapt the environment to our need, would be to make a custom-tailored tool that could hook in to other existing technologies and the same time being compatible with existing environments, ensuring compatibility and reducing risk of cross pollution between installers.

Versioning and Version Control

At CERN the IT department has been working towards making Git and GitLab the main source control solution for the whole lab. In the past we used SVN, but have lately adopted Git as our main provider. This decision was mainly based on the added value of having the IT department supporting and maintaining the repositories, and lately reinforced by the many new functionalies added in the GitLab ecosystem [10].

By using GitLab's built in rest API in combination with our existing CI engine the transition from SVN to Git was fairly painless, but took about 4 months to complete.

"Software upgrade versioning is the process of assigning either unique version names or unique version numbers to unique states of computer software. Within a given version number category (major, minor), these numbers are generally assigned in increasing order and correspond to new developments in the software. At a fine-grained level, revision control is often used for keeping track of incrementally different versions of information, whether or not this information is computer software" [11].

In the RADE framework we follow the same principle for versioning. In addition, module names are shortened/converted to an acronym and the acronym is used in all tools that interacts or is related to the module. For instance, if we have a project called "PowerControlSoftware", the acronym becomes "pcs", and this acronym is then used both on Git, in the project's documentation page, and in any support or correspondence done that relates to the module. Then whenever a release is done (after unit, integration and requirement testing depending on the module criticality), we assign a version number to the release, tag it and start the release procedure.

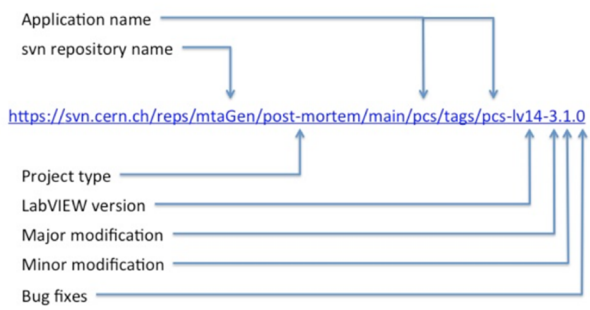

Figure 2: SVN Naming scheme.

In the transition from Git to SVN we kept the same naming scheme (Fig. 2) so the CI engine would be backwards compatible with the previous scheme.

The module tag number is kept all the way from the source to the repository. In addition, we have a fixed project structure for all developers, making it easier to identify where to find the sources and relevant files for each project.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA129

Repositories

As with the package managers there are also many different repository tools on the market. Our requirements for the repository had many similarities with the package manager environment, however in this case we decided that all packages should be hosted in the same environment, removing the cross-platform requirement, and should have as much built in logic as possible, moving most of the logic from the client to the repository.

For the repository we wanted a tool that:

- Had a Representational State Transfer (REST) interface (or similar) for ease of integration [12]
- Automatically indexes packages
- Supports listing and querying
- Supports upload and download of both binary and source-based packages
- Supports hosting of any software language and file format
- Fits in the existing CERN OpenStack infrastructure
- Compatible with GitLab
- Compatible with Jenkins

Table 2: Evaluation of Repository Tools

Name	License	API	Mixed Package	Server-Side Management
VIPM [8]	Paid	No	No	No
EOS [13]	N/A	No	Yes	No
GitLab [10]	MIT	Yes	Yes	Yes
WebSite	N/A	No	Yes	No
Nexus [14]	GPL	Yes	Yes	Yes
PyPI [15]	GPLv2	Yes	No	No

After evaluating and testing several repository tools, some of which are shown in Table 2 above, we narrowed the list down to 2 possible candidates that would suite our needs: Sonatype Nexus and GitLab. If the selection where done today, GitLab would be the clear choice, but at the time the comparison was done, GitLab was only meant to store source files and not intended to be a repository solution, and Nexus was the clear choice given its API and features. [10, 14].

Sonatype Nexus

Sonatype Nexus Open-Source Software (OSS) is a repository manager. It allows you to proxy, collect, and manage your dependencies so that you are not constantly juggling a collection of packages. It makes it easy to distribute your software. Internally, you configure your build to publish artefacts to Nexus and they then become available to other developers. You get the benefits of having your own local repository at the same time as you can proxy, link and share artefacts between other repositories and developers [14].

RADE INSTALLER

The RADE installer, following the philosophy of the RADE framework, was implemented with ease of use and efficiency, in terms of workflow, in mind (Fig. 3). The developer should not need to focus on versioning and inter module compatibility, rather get the necessary dependencies installed with as little interactions as possible. If you check out an existing project from git that contains any package in the RADE eco system, you only have to reference the project and the installer will find the implicit dependencies.

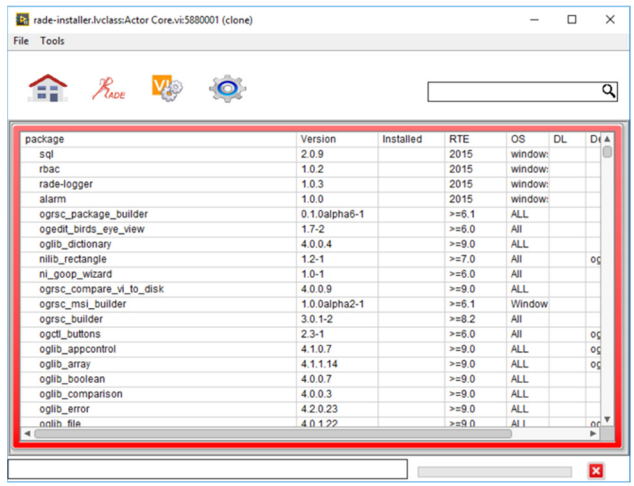

Figure 3: RADE Installer Main interface.

Any update and new package are visible from the interface, and the user can toggle between “public”, “RADE” and “VIPM” packages (Figure 3). The user can also choose if updates should be installed automatically (hidden) or manually.

Environment Integration

The RADE Installer can be launched from within the LabVIEW development environment. An option available from the “Tools -> RADE” menu will launch a statically compiled version of the installer, built using packed project libraries. This allows for instant installation of a desired package without having to restart LabVIEW. Only in cases where a system package requiring restart of the operating system does one have to restart the environment [16].

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA129

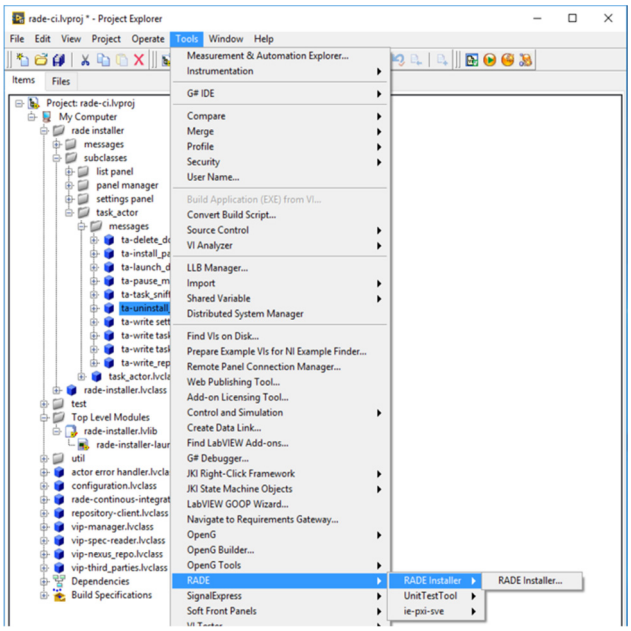

Figure 4: RADE Installer Launcher from LabVIEW Environment.

Installation and Updates

The RADE installer (Fig. 4) is built like any other RADE application package in the CI engine, and will at startup check for new releases on the Nexus repository. If there is a new version available and the user currently launching LabVIEW is authorized to install a new version (same user name as previous installation or administrator rights), the package will automatically be downloaded and installed (Fig. 5). We were thinking of making this feature optional, and let the user decide when and where to install the updates, however past experiences have shown that many users post-pone the updates and quickly gets out of synch with the maintained version. Since the installer itself does not have any dependency with the packages or applications being developed, we deemed it safe to automate the installation. In the case of a network interruption or timeout, the installer will simply time out. If anything was missing or incomplete, the installer does not complete the update.

BUILD CYCLE

The RADE build cycle has not changed much after introducing the RADE installer and Nexus repository manager. The main challenge was breaking all the different libraries into individual installers and map their interdependencies. Since everything in the past was shipped as one big package, we didn't have to manage the interoperability and compatibility between packages, however with the new release scheme, we always have to take care that none of the libraries break or fail when doing a release. As an added bonus, the release time has gone down even more, and we can now release stable packages within minutes and add new packages incrementally without affecting users.

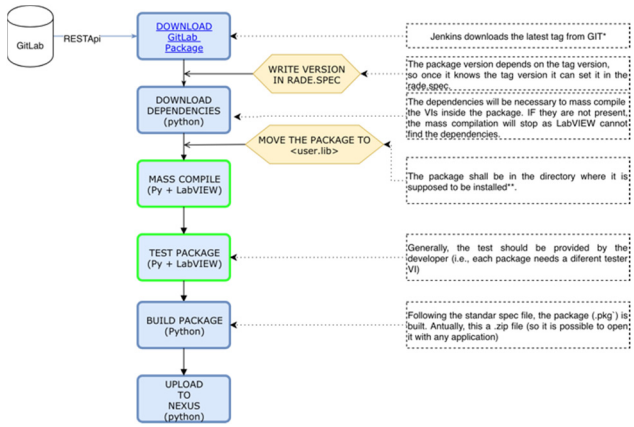

Figure 5: RADE Package Creation Workflow.

Package Compatibility

New packages added to the build cycle are either compatible with the NI or VI package manager if they are intended to be installed in the LabVIEW environment. This means that any package can be installed on a user's machine even if he or she does not have the RADE installer available or cannot connect to the repositories. This was a particularly important requirement in case the development environment is outside CERN which often is the case for test system.

REMARKS AND ISSUES

One of the main challenges with the new installer has been to manage user environments and file permissions.

At CERN we have terminal server with multiple users, central development machines and virtual environments used by many people.

The LabVIEW environment relies on relative file structures, naming and known paths, and it is therefore important to retain the permissions and structure both the environment and user expects. This gave us some aha moments when developing the installer. This was solved by having the user "log in" to the installer and use his (encrypted) credentials when interacting with the file system.

Also migrating all the packages from one big bundle to individual packages both compatible with the NI installer and the VI Package manager and at the same time serving our needs, was more manual labor than expected. Once done we do not need to maintain or change it, but the initial effort should not be neglected.

Moving from SVN to Git was also a task that proved to be more challenging than expected. SVN can be used as a monolithic source control system and any subsection of the repository can be checked out. Git on the other hand is decentralized, distributed and by nature not monolithic. This means that the user cannot check out sub portions of the repository (could be done with submodules, but this requires quite a bit of hands on work), and it also means that the user will have all versions of a project installed locally when checking it out/cloning. Fortunately, GitLab had the possibility to do break projects into groups and with some simple scripting and organization we managed to get a

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA129

functionality which was as close to what we had before the transition, reducing the effort and cost of the migration.

The RADE Installer still hasn't been released to all CERN users and is still being evaluated internally in the team.

CONCLUSION

Introducing Nexus and the possibility to break packages into singular elements in the RADE CI engine has greatly improved our capacity to both release and keep track of packages. Cross dependent developments using both Java, C++ and LabVIEW have benefitted from the new structure and we have become more conscious in designing packages with test and traceability in mind. The introduction of the RADE installer in the team is still an ongoing process, but the benefits outweighs the efforts so far. It has become easier to share and reuse code, and adding packages to the installer encourages a workflow that reduces errors in deliveries.

FUTURE IMPROVEMENTS

The RADE installer still does not support full environment synchronization, so the plan is to add this functionality in the next release. We also have to follow closely what National Instruments plans to do with their Next Generation environment and make sure the changes we do are compatible with future releases.

Finally, as time has gone by, more and more features have been added to GitLab and it is now (2019) a full "DevOps" solution that supports the whole software workflow from development, build to release and we are considering replacing both Jenkins and Nexus with Gitlab.

REFERENCES

[1] O. O. Andreassen, D. Kudryavtsev, A. Raimondo, A. Rijllart, S. Shaipov, and R. Sorokoletov, "The LabVIEW RADE Framework Distributed Architecture", in *Proc. ICALEPCS'11*, Grenoble, France, Oct. 2011, paper WEMAU003, pp. 658-661.

[2] M. Rose, "Continuous Integration (CI)", (2008), http://searchsoftwarequality.techtarget.com

[3] O. O. Andreassen and A. Tarasenko, "Continuous Integration Using LabVIEW, SVN and Hudson", in *Proc. ICALEPCS'13*, San Francisco, CA, USA, Oct. 2013, paper MOMIB08, pp. 74-76.

[4] Wikipedia, "Comparison of version-control software" https://en.wikipedia.org/wiki/Comparison_of_version-control_software

[5] B. Frammery, "The LHC Control System", in *Proc. ICALEPCS'05*, Geneva, Switzerland, Oct. 2005, paper I1_001.

[6] Z. Zaharieva, M. Martin Marquez, and M. Peryt, "Database Foundation for the Configuration Management of the CERN Accelerator Controls Systems", in *Proc. ICALEPCS'11*, Grenoble, France, Oct. 2011, paper MOMAU004, pp. 48-51.

[7] Wikipedia, "List of software package management systems", https://en.wikipdia.org/wiki/List_of_software_package_management_systems

[8] National Instruments, "VI Package Manager", http://www.ni.com/tutorial/12397/en/

[9] National Instruments, "NI Package Manager", http://www.ni.com

[10] GitLab, "A Full DevOps Tool", https://gitlab.com

[11] Wikipedia, "Software Versioning" https://en.wikipedia.org/wiki/Software_versioning

[12] Wikipedia, "Representational State Transfer", https://en.wikipedia.org/wiki/Representational_state_transfer

[13] CERN, "EOS Open Storage", http://eos.web.cern.ch

[14] Sonatype, "Nexus Repository OSS", https://www.sonatype.com

[15] PyPi, "Python Package Index", https://pypi.org

[16] National Instruments, "Packed Project Libraries", https://zone.ni.com/reference/en-XX/help/371361R-01/lvconcepts/packed_libraries/