Software Development Infrastructure for the FAIR Experiments

F. Uhlig, M. Al-Turany, D. Bertini, R. Karabowicz
GSI, Plankstrasse 1, 64291 Darmstadt, Germany
E-mail: `f.uhlig@gsi.de`

Abstract. The proposed project FAIR (**F**acility for **A**nti-proton and **I**on **R**esearch) is an international accelerator facility of the next generation. It builds on top of the experience and technological developments already made at the existing GSI facility, and incorporate new technological concepts. The four scientific pillars of FAIR are NUSTAR (nuclear structure and astrophysics), PANDA (QCD studies with cooled beams of anti-protons), CBM (physics of hadronic matter at highest baryon densities), and APPA (atomic physics, plasma physics, and applications).

The FairRoot framework used by all of the big FAIR experiments as a base for their own specific developments, provides basic functionality like IO, geometry handling etc. The challenge is to support all the different experiments with their heterogeneous requirements.

Due to the limited manpower, one of the first design decisions was to (re)use as much as possible already available and tested software and to focus on the development of the framework. Beside the framework itself, the FairRoot core team also provides some software development tools. We will describe the complete set of tools in this article. The Makefiles for all projects are generated using CMake. For software testing and the corresponding quality assurance, we use CTest to generate the results and CDash as web front end. The tools are completed by subversion as source code repository and trac as tool for the complete source code management.

This set of tools allows us to offer the full functionality we have for FairRoot also to the experiments based on FairRoot.

1. Introduction

The FairRoot framework [1,2,3] is an object-oriented simulation and analysis framework for experiments planned to be built at FAIR.

It is based on ROOT [4] and the Virtual Monte-Carlo (VMC) [5] interface. It includes core services for detector simulations and offline analysis. Thanks to VMC, it is possible to switch between different transport codes without changing the user code. The framework is designed to optimize the accessibility for beginners and developers, to cope with future developments, and to be usable for more than one physics community. This in the end leads to an enhancement of synergies between the different physics experiments at or outside the FAIR project.

Beside the framework itself, the FairRoot core team also provides some software development tools for the experimental groups using FairRoot as a base for their own developments. This is depicted in figure [1], where the user is shown in the middle of the software tools provided by the FairRoot core team. These tools should help the user to concentrate on his main topic, the development of the experiment specific simulation and analysis software, which includes also the independence of a specific code development tool. Each user can work with the tool he

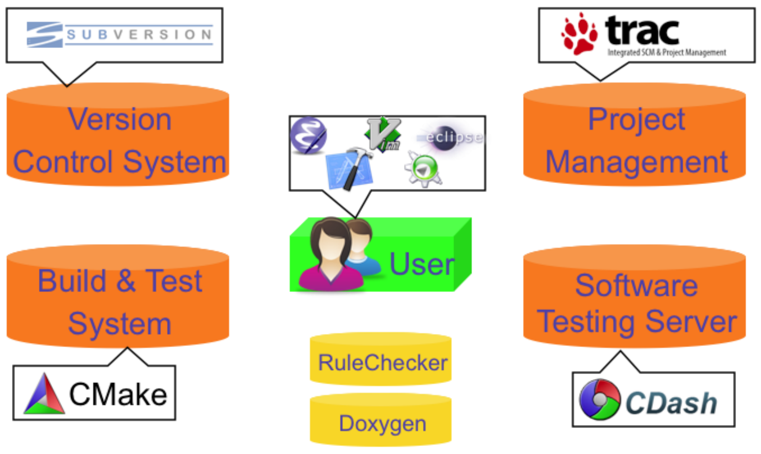

Figure 1. Supported software tools

likes. The tools used by the developers range from a simple editors like *vi* to IDEs (Integrated Development Environments) like *XCode*.

The main components of the software development infrastructure are the source code repository and the test and build system. The so-called software testing server is used to store and view the information produced by instances of the test and build system. The project management software provides a web interface to the source code repository, a bug tracking, and a wiki system, which are tightly integrated. This makes it for example possible to link bug tickets directly to the problem in the code, which allows the user to move with one click in the web browser from the description of a problem to the erroneous code block. We will describe all components in more detail below.

Shown in figure [1] are also the class documentation software Doxygen [6] and the RuleChecker developed by IRST [7] for coding convention compliance checks, which are not further described in this article.

2. Source Code Revision Control System

As revision control system we use Subversion (SVN) [8], as most of the LHC experiments do. The main SVN repository at GSI hosts beside the FairRoot framework also the source code of all supported experiments. The benefit for the experiments is that we can provide features like continuous integration and backup of the source code on every commit to the repository. This support is triggered by the repository itself.

In contrast to this central repository structure, there are also distributed version control systems available. Since these systems provide some advantages, we did some tests with one specific system, namely GIT [9]. The major advantage of GIT is that with a clone one gets the entire history of the project. This means that nearly any operation is now done with data on the local disk, which is much faster than the access to the central svn server and allows working offline. Because git does not depend on a single centralized server, but does have the ability to push and pull changes between different git repositories, one can setup completely new workflows. For example, one can have a central master repository from which the developer pulls the data. For each sub-team one has a different repository into which only members of the sub-team can push. Then a *software manager* can pull the work of the different sub-teams from their repositories and push the changes back to the central master repository. This workflow is

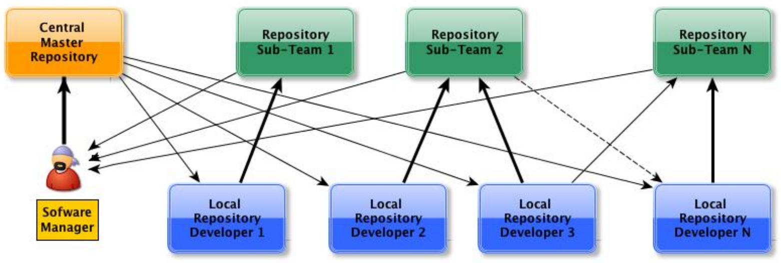
Central Master Repository
Repository Sub-Team 1
Repository Sub-Team 2
Repository Sub-Team N
Sofware Manager
Local Repository Developer 1
Local Repository Developer 2
Local Repository Developer 3
Local Repository Developer N

Figure 2. GIT workflow as described in the text. Thin lines indicate pulls from a repository, bold lines a push. Developers may have rights to push to more than one repository. There is also the possibility to pull from other repositories beside the central repository, which is indicated as dotted line.

indicated in figure [2].

On one hand this makes it possible to implement incredible workflows using git, on the other hand this can make the workflow complicated. In our opinion to get used to the complexity of git, and to work effectively, one has to use git frequently, which is not the case for our developers. Due to this reason, we do not switch to git as revision control system but keep using svn.

For experienced developers it is still possible to use git. They can create their own local git repository using the tool git-svn, which clones a git repository from the central svn repository. From this git repository it is possible to create a new working branch or pull it from another computer, do the development, and push back the changes to the *master* git repository. From this *master* git repository the changes are then committed to the central svn repository, where the many git commits show up as only one svn commit. This workflow allows combining the flexible workflow of git with the existing central svn repository.

3. Build System

As build system for FairRoot we use CMake [10,11], which is an open-source system that manages the build process in an operating system and compiler independent manner. Simple configuration files in a portable script language are used to generate native build files. These can be either project files for Integrated Development Environments such as Eclipse, Apple XCode or Microsoft Visual Studio, as well as UNIX, Linux or NMAKE style makefiles. Due to its modular design, it is easily extensible. Therefore, one should correctly say that CMake is not a build system but a build file generator.

CMake comes with a quite powerful integrated scripting language, which has the ability to create complex custom commands. If this is not sufficient, CMake also includes the functionality to execute other external programs and store the return values in variables. These commands can for example be used to generate new source files during the build process that are then compiled into the software. This feature is also used to generate the class documentation of FairRoot automatically on a daily basis using Doxygen [6].

4. Automatic Tests

Since FairRoot is used by many experiments as a base for their own specific developments, it is essential to know the status of the project at any time in order to find problems, which enter the code base as early as possible. This includes especially changes in the external dependencies and problems that occur only on some of the supported platforms. Platform in this sense means, a combination of operating system, operating system flavor, compiler, and compiler version. The supported operating systems are Linux (in nearly any flavor), MacOS X 10.5, and 10.6. There are no known restrictions concerning the compiler. The code was already tested with the GNU compiler collection (gcc), the Intel compiler collection (icc), and the Sun Studio compiler suite, however the mainly used compiler is gcc.

To produce the needed quality assurance (QA) information we use the tool CTest that is part of the CMake suite. This tool automizes the configuration, the compilation, and testing procedure on any computer.

CTest can be used to support different build modes. We support the so-called *Continuous Build*, *Nightly Build*, and *Experimental Build*.

The most important one is the *Continuous Build*, which is triggered on a dedicated build server by the source code repository whenever there is a change in the repository. This mode updates the local working copy from the repository, configures, compiles, and tests the project. To decrease the build time the *Continuous Build* recompiles only source files which have changed since the last update. Once per day (normally during night when the machines are idle) a *Nightly Build* is started on all available platforms. As only difference to the *Continuous Build*, the *Nightly Build* always starts with a clean build directory. The *Experimental Build* can be used at any time to build and test changes without an update of the local working copy.

After any of the three test modes the generated information about the configuration, compile and test stage are sent to the central web server for further usage. The information contains the output from the generation of the Makefiles, all warnings and errors from the compilation stage, and the results from the test stage. The build stage will not fail completely if there is an error but will go on with the next target, which is not affected by the error. In the end, one gets all error messages, which occur during the build stage. Missing targets may affect the test stage.

The results from the test stage include information if the test passed, failed, or ran into a timeout together with the execution time for each test. In case of a failure or a timeout, also the complete output of the test is sent to the central server.

The fact that the results from CTest are pushed to a central server allows each user to setup his own test system, which makes it possible to test many different platforms without the necessity to have direct access to the system. The results of the CTest run even allow debugging user problems remotely. So using the client server architecture each user can help to improve the project when setting up a test system for an up to now untested platform. This setup is done adjusting one template according to the setup of the specific machine and creating a cron job for automatic execution of the CTest script.

5. Dashboard

The second and not less important part of the quality control is to collect all the produced information, to process it, and finally to display it in an easy to use fashion. For these tasks, the software package CDash [12] is used together with a database to store the collected information. CDash on request generates dynamic web pages out of the stored information. The main page provides a fast overview about the status of the project. From here one gets more detailed information e.g. about updates or problems during the configure, the build, and the test stage.

From the information sent by the clients, the server can produce additional aggregated information as for example the execution time for a specific test on a specific machine as a function of time. This information can be used to find changes in the code which result in a slowdown of the execution time of a test, a problem which is not easy to spot without this aggregated information. All the available information enables the developers to spot problems in a fast and convenient way.

The server can also produce automatic alerts to administrators or developers in case of problems, e.g. an e-mail to a developer, who commits erroneous code.

6. Project Management Tools

Project management tools are a term covering many types of tools that are used to deal with the complexity of large software projects. Talking about project management tools one can really mean completely different things, ranging from scheduling, cost control and budget management, collaboration software, quality management to documentation. We use trac [13] as project management tool that combines the advantages of a wiki and a bug tracking system with a powerful web front end to our subversion repository. Any of these tools alone is very useful and may be even needed to manage a larger software project but the tight connection of these tools even enhances the value of each single one. All tools use a wiki markup language that allows creating hyperlinks easily. This simplifies the connection from a bug report to a line of code in a specific revision of the software or makes it possible to create a hyperlink from the commit message to a closed feature request.

The source code browser allows navigating in the source tree, checking older revisions or commit messages. This includes also a diff-like comparison between different code revisions.

The bug tracking system or better issue tracking system is necessary to keep track of the development process. This includes also the ability to report software problems or to request new features in a well-defined way and not by sending mails to the preferred developer. The latter approach always has the problem that the receiver of the mail is not responsible and forgets about the problem whereas the first approach makes the process traceable. The problem can be assigned to the appropriate developer and the bug reporter will be informed about the progress of his report.

Trac also allows defining milestones that specify states of the project like a stable version or the implementation of a feature. The tickets can be assigned to such a milestone and the roadmap shows these milestones together with the open and closed tickets and by this means shows the progress towards the milestone. The timeline shows all events like new commits to the repository or new tickets in the project ordered in time.

The wiki allows creating and organizing a collection of linked information without much effort. Due to the easy creation of cross-links, the complete system becomes a valuable source of information.

With a plug-in mechanism, it is possible to add more features such as for example support for other version control systems like git.

7. Summary

In summary we present in this paper the tools used for the development of FairRoot and the experiment code based on it (CbmRoot, PandaRoot, etc.). Each of the tools is also provided to these experiments. This includes the usage of the central svn repository, which allows the experiments to get a central backup of their code and the continuous integration without any effort. All experiments also use the CMake based build system which enables them to use all the functionality like the test support or the dashboard without installation. The results of the automatic build and test processes are summarized at one location as web pages, which allow browsing through all the necessary and relevant information. For all communities we use a unified issue tracking system that allows keeping track of bugs and feature requests from all experiments in a unified manner. This is needed because bugs or feature request reported by a specific experiment can affect all experiments.

References

[1] M. Al-Turany, D. Bertini, and I. Koenig. CbmRoot: Simulation and analysis framework for CBM experiment. In S. Banerjee, editor, *Computing in High Energy and Nuclear Physics(CHEP-2006)*, volume 1 of *MACMILLAN Advanced Research Series*, pages 170-171. MACMILLAN India, 2006.

[2] D. Bertini, M. A-Turany, I. Koenig, and F. Uhlig. The fair simulation and analysis framework. In *International Conference on Computing in High Energy and Nuclear Physics (CHEP'07)*, volume 119 of *Conference Series*. IOP Publishing, 2008.

[3] M. Al.-Turany FairRoot: http://fairroot.gsi.de.

[4] R. Brun and F. Rademakers. Root - an object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research A*, 389:81-86, Sep. 1997.

[5] R. Brun, F. Carminati, I. Hrivnacova, and A. Morsch. Virtual Monte-Carlo. In *Computing in High Energy and Nuclear Physics*, pages 24-28, La Jolla, California, 2003.

[6] Doxygen: http://www.doxygen.org

[7] RuleChecker: http://chep2000.pd.infn.it/paper/pap-f361.pdf

[8] SVN: http://subversion.apache.org/

[9] GIT: http://git-scm.com/

[10] CMake: http://www.cmake.org

[11] K. Martin and B. Hoffmann. Mastering CMake, A Cross-Platform Build System, Kitware Inc., 2007

[12] CDash: http://www.cdash.org

[13] TRAC: http://trac.edgewall.org/