Journal of Physics: Conference Series **513** (2014) 062007 doi:10.1088/1742-6596/513/6/062007

Automatic Tools for Enhancing the Collaborative Experience in Large Projects

D Bourilkov[1], **J L Rodriquez**[2]

[1] University of Florida, Gainesville, Fl, USA
[2] Florida International University, Miami, Fl, USA

E-mail: bourilkov@phys.ufl.edu, jrodrig@fiu.edu

Abstract. With the explosion of big data in many fields, the efficient management of knowledge about all aspects of the data analysis gains in importance. A key feature of collaboration in large scale projects is keeping a log of what is being done and how - for private use, reuse, and for sharing selected parts with collaborators and peers, often distributed geographically on an increasingly global scale. Even better if the log is automatically created on the fly while the scientist or software developer is working in a habitual way, without the need for extra efforts. This saves time and enables a team to do more with the same resources. The `CODESH` - COllaborative DEvelopment SHell - and `CAVES` - Collaborative Analysis Versioning Environment System projects address this problem in a novel way. They build on the concepts of virtual states and transitions to enhance the collaborative experience by providing automatic persistent virtual logbooks. `CAVES` is designed for sessions of distributed data analysis using the popular `ROOT` framework, while `CODESH` generalizes the approach for any type of work on the command line in typical UNIX shells like bash or tcsh. Repositories of sessions can be configured dynamically to record and make available the knowledge accumulated in the course of a scientific or software endeavor. Access can be controlled to define logbooks of private sessions or sessions shared within or between collaborating groups. A typical use case is building working scalable systems for analysis of Petascale volumes of data as encountered in the LHC experiments. Our approach is general enough to find applications in many fields.

1. Introduction

In today's interconnected world, collaboration is a rapidly growing phenomenon. Sharing files and know-how is commonplace between geographically distributed contributors, who may never meet face-to-face. At the same time a lot of information is simply stored on web pages, typically accessed by cutting and pasting text onto the command line and then executing it. Producing these web pages requires lots of effort and is time consuming.

We explore a way to improve this sharing by building on the concept of a virtual session: the process of doing some computer work by moving between well defined initial and final states. The concept of "virtuality" with respect to existence means that we can define states that may be produced in the future, as well as record the "history" of states that exist now or have existed at some point in the past. By keeping an *automatic log* of initial states and transformations (knowledge about how to transform to desired final states) we have a general tool to track the evolution of a project. Such a tool will be equally useful for software development or any scientific work done on computers. A good logging system will enable a collaborating group to create and/or recreate virtual states on demand. The ability to reproduce a state can have

many implications: it may be more practical (e.g. much less space consuming) to keep the initial states and the transformation knowledge than all final states. The decomposition in sessions can describe complex processes and procedures as a sequence of many small steps at the desired level of "atomicity".

The idea of virtual logbooks of sessions complements the idea of virtual data (data with a recipe how to reproduce it). We support the interaction of users working in their habitual ways, by capturing the contents of the session automatically and on the fly as the work progresses without interfering with the users' activities. There is no need per se to provide any code in advance, but users can execute/modify preexisting programs if desired. When a piece of work is worth recording, users log it in a persistent session repository with a unique identifier for later use/reuse. The `CAVES` and `CODESH` projects [1, 2, 3, 4] build tools to address this problem. Our goal is to enable virtual communities to easily build their ecosystems, at the same time retaining full control over their infrastructure.

2. The `CAVES` and `CODESH` Projects

The basic idea behind the two projects is the same: to intercept in a transparent way the activity (commands and scripts) of users. Consequently, they share the same architecture. `CAVES` is designed specifically for users performing data analysis with the widely popular analysis package `ROOT` [5], by providing a plugin which extends the way `ROOT` handles the commands typed in by the users. `CODESH` is a generalization of the same approach for any type of work done on the command line, like scripting in typical shells, e.g. `bash` or `tcsh`.

The primary use case is a 'virtual session'. Each user works on a per session basis in an open environment. The work of the user for a session is recorded in the 'virtual logbook', while some external dependencies like project software are provided by the collaborating group. The split between information stored in the logbook and externals is to some extent arbitrary. We will call it a *collaborating contract*, in the sense that it defines the responsibilities of the parties involved in a project. The fundamental concept is to store information in enough detail to provide users with a share/replay mechanism at any time or place for sessions of collaborators, optionally modifying the session inputs.

This is achieved by maintaining a virtual logbook, which records the initial state (pre-conditions of a session), all the commands typed by the user, all the outputs generated and all the programs executed to produce the results. Also the changes made to the environment i.e. environment variables and aliases are recorded.

When a user's session ends, or when the user would like to checkpoint the work done so far, he/she logs it to a repository with a uniquely generated tag. If the user desires, the log can automatically collect the source code of executed scripts with pre-defined file extensions (e.g. `.sh`, `.py`, etc.), and optionally also the data used. The repositories can accumulate thousands of stored sessions. Reproduction of a session is possible by extracting the log, data and source code files, executing the commands listed in the log files and running the scripts that have been downloaded. The environment changes can be carried across sessions.

For example, if a user executes a script `echo.csh` that echoes its input, e.g.

```
user@node:tcsh$ echo.csh Hello_Collaboration!
```

`CODESH` will log the command line *and* the script used, as well as the result of the command. The result can be reproduced at any time, and compared to the log of the original run.

The repositories of such sessions can be on local machines for personal use or on shared servers for use by collaborating groups. Generally users store their sessions locally and 'publish' selected important sessions to shared repositories. Users may also extract and re-produce sessions stored by other collaborators. We provide tools to setup servers using different backends.

A spin-off from our project is a feature, aptly called 'Snapshot', which allows logging entire

directory structures under the current working directory. These can later be retrieved and thus provide a virtual working directory. This feature was developed before Dropbox became popular, and provides a handy way to share whole directory trees hosted on private infrastructure or private clouds.

Using this concept, a virtual session can be copied to any place on the same machine or even across machines and re-started or modified. Of course this is possible if the user works relative to the root of the snapshot directory and avoids using absolute paths.

We have identified three distinct Tiers in the architecture:

- A. The User Tier
- B. The Main `CODESH` or `CAVES` Tier
- C. The Backend

Each Tier is completely independent of the others and makes use only of the interfaces provided by them. Thus we can change one Tier without affecting the operation of the others. As most architecture details are common for the two projects, we will concentrate on the `CODESH` description, mentioning `CAVES` only where necessary to highlight the differences between the projects. Lets examine each Tier in turn:

A. The User Tier: This Tier provides the User Interface, similar to the UNIX/Linux command line shell or to the `ROOT` command line. The user can work in interactive or batch mode. In interactive mode, the user types shell (or `ROOT`) commands just as in a UNIX/Linux shell. In addition `CODESH` (`CAVES`) commands are available for session logging and similar tasks. All the input from the user is parsed and fed to the second Tier, which is the main `CODESH` (or `CAVES`) Tier. The execution results are displayed on the screen for the user to view.

B. The Main `CODESH` (`CAVES`) Tier: This is the heart of our system. It is responsible for getting the user input, logging the user sessions, maintaining state information, delegating the shell (`ROOT`) commands to the under-lying shell (analysis package) and all the communication with the backend Repositories.

Based on the logical separation of the tasks, we have identified 4 different modules that comprise this Tier. They are:

i. `CODESH` (or `CAVES`) client class: It is the main controller module that interacts with the remaining 3 modules for the successful execution of tasks in this Tier. It delegates shell commands directly to the shell or through the Extract module, which is described later. It reads and updates the state information stored in the State Information module. It also interacts with the `CODESH` backend module for the storing and retrieving of the sessions.

ii. `CODESH` (`CAVES`) Backend: This module interacts with the backend repositories to provide the storage and retrieval of session information and status information e.g. a listing of all stored sessions. It provides a backend independent interface, which is used by the `CODESH` module.

iii. State information: This module stores and maintains all the configuration information during any active user session. We broadly classify this state information in two categories: system information (aliases and environment variables used during the session) and user configuration information (various user-selected options).

iv. Extract module: This module is responsible for the extraction of sessions i.e. re-executing them and getting the desired outputs. It delegates the shell (`ROOT`) commands and scripts to the under-lying shell (analysis package) for execution. Currently we support the `bash` and `tcsh` shells. But support for other shells can be added easily.

C. The Backend: The Backend stores the sessions for later reuse by users who extract selected sessions. For each session, we store the log files, the source code files and optionally the data files. Each session is identified by a unique identifier which consists of 3 parts: the user's name, the current universal date and time and a user supplied name for the session. Our

architecture makes it easy to support different types of backends; currently four types and a metadata option are supported.

i. `ASCII`: We use flat files (text or binary) as the simplest backend for storing the sessions. The repository can be local (on disk volumes mounted on the user machine) or remote. In the latter case we provide remote access using `ssh` and `scp`. The user can utilize private and public keys or the `ssh` agent technique to avoid being asked to type in a password several times per session.

ii. `CVS`: We use `CVS` [6] as a version control type backend for storing the sessions. Our commands, e.g. **log** session and **extract** session, use `CVS` commands like checkin and checkout 'under the hood' to talk to the backend. A comprehensive listing of `CODESH` commands is given in [4]. We provide tools for using Remote `CVS` as a backend, implemented as a pserver.

iii. `SVN`: We use `Subversion` [7] as an alternative version control type backend for storing the sessions. This backend can handle transparently both text and binary files and is gaining in popularity. We provide tools for setting up a Remote `SVN` as a backend, using svnserve.

iv. `Git`: Another version control system [8], which is used by Linux and is gaining wider traction. This backend is under active development and the first release will be out soon.

v. `MySQL`: This optional backend stores only the metadata information regarding each session in a database. Users can annotate sessions for fast searches of particular types of work. After a matching session is found, a local or remote repository can be contacted to extract the complete session.

Every user may have local repositories for personal sessions. Typically the user will want to commit some of the sessions to shared remote repositories and extract some sessions stored by other users in the shared repositories. For the remote `ASCII` back-end users need shell access, for the other backends a server user/password is enough for access, without UNIX accounts on the server.

Controller of the Repositories: This module takes care of the maintenance, recovery and similar tasks related to the different repositories. We provide support for copying and moving sessions between repositories and for deleting sessions stored in a particular repository.

3. Project Status

Currently `CODESH` is implemented in Python, and `CAVES` in C++. For `CAVES` the user compiles an executable using the `CAVES` code and the `ROOT` libraries. Once started, this executable has all the functionality and behavior of the normal `ROOT` executable, including in addition the `CAVES` commands. Typically a user starts `CODESH` by running the Codesh.py file. Different loglevels and other customizations are specified in a Codesh.conf file or typed interactively at the start of a `CODESH` run. In interactive mode, the user has a command line interface to run the session. Optionally, batch mode can be used, by specifying a file containing all the commands that are to be executed. For logging work and re-creating the results of previously stored sessions, the main `CODESH` commands are:

i. browse: list all stored sessions or optionally restrict the search depending on user specified criteria; used also to browse metadata associated with the sessions.

ii. inspect: view the contents of a particular session and optionally download all the associated source code files.

iii. extract: execute a stored session and re-create the results.

iv. log: log a session between user defined checkpoints along with (optionally) all or some of the programs executed during the session.

The interactive and batch command line interfaces are fully tested and in production both for local and remote repositories. In addition, web interfaces can be deployed to browse directly the sessions stored in `CVS, SVN` or `Git` backends, or to get "Snapshots" of whole directory trees. The web access mode is restricted to read only.

We provide an expandable test suite for automatic testing of the basic functionality of `CODESH`. During development cycles passing all tests is a precondition for making new releases public. The tests serve a dual purpose as varied examples of using `CODESH` in different modes of operation and at different levels of sophistication.

4. Outlook

Our ongoing and future work is focused on:

- implementing the robust functionality available with the `ASCII`, `CVS` and `Subversion` backends in the `Git` case
- automatically converting session logs to workflows
- enabling the extraction of sessions from web pages, without using the command line, by developing the ability to run not only locally, but in the cloud, thus delivering the results of remote runs directly to a web browser; this can also enhance the educational experience of students by providing tools for interactive hand-on sessions.

Information about public releases of our functional systems for automatic logging of typical working sessions is available from [9]. The `CODESH` project is open source, available from [10, 11].

In summary, our projects take a pragmatic approach in assessing the needs of a community of collaborators by building a series of working packages with increasing sophistication. By extending with automatic logbook capabilities the functionality of typical UNIX shells (`tcsh` or `bash`) - the `CODESH` project, or a popular analysis tool as `ROOT` - the `CAVES` project, these packages provide an easy and habitual entry point for researchers or software developers to explore new concepts in real life applications and to give valuable feedback for refining the system design and further hardening of the already robust performance. Users of our systems are most welcome.

References

[1] D. Bourilkov, "The CAVES project: Exploring virtual data concepts for data analysis," http://arxiv.org/abs/physics/0401007, arXiv:physics/0401007; D. Bourilkov, "THE CAVES Project - Collaborative Analysis Versioning Environment System; THE CODESH Project - Collaborative Development Shell," http://arxiv.org/abs/physics/0410226, Int. J. Mod. Phys. A **20** (2005) 3889 [arXiv:physics/0410226]; D. Bourilkov, "Virtual States and Transitions, Virtual Sessions and Collaboration," ICCS 2005 conference, Atlanta, USA, 2005; V.S.Sunderam et al. (Eds.): ICCS 2005, LNCS 3516, pp. 342-345, 2005, Springer Verlag Berlin Heidelberg.

[2] D. Bourilkov and V. Khandelwal, "CODESH: An Intelligent Development Shell for Seamlessly Logging, Exchanging and Reproducing Results and the Methods used in Obtaining Them," WMSCI 2005 conference, Orlando, USA, 2005; published in the Proceedings, ed. N.Callaos, W.Lesso and K.Horimoto, ISBN 980-6560-60-4, vol. VIII, p.175, IIIS 2005.

[3] D. Bourilkov *et al.*, "Virtual Logbooks and Collaboration in Science and Software Development," IPAW06, International Provenance and Annotation Workshop, Chicago, Illinois, USA, May 3-5, 2006;' L.Moreau and I.Foster (Eds.): Provenance and Annotation of Data, LNCS 4145, pp. 19-27, 2006, Springer Verlag Berlin Heidelberg.

[4] D. Bourilkov and S. Sonapeer, "Enhancing Collaboration in Large Scientific Projects through Virtual Logbooks," Nuclear Science Symposium, Honolulu, Hawaii, 2007. Conference Record, 2007. NSS '07. IEEE Volume: 1 Digital Object Identifier: 10.1109/NSSMIC.2007.4436473 Publication Year: 2007 , pp. 901-906.

[5] Brun, R. and Rademakers, F.: ROOT - An Object Oriented Data Analysis Framework. Nucl. Inst. & Meth. in Phys. Res. **A 389** (1997) 81–86

[6] `CVS`: The Concurrent Versions System `CVS`, `http://www.cvshome.org/` .

[7] The `Subversion` version control system, `http://subversion.tigris.org/` .

[8] `Git` – distributed-is-the-new-centralized, `http://git-scm.com/` .

[9] CODESH/CAVES home page, `http://cern.ch/bourilkov/caves.html` .

[10] `http://freecode.com/projects/codesh/` .

[11] `http://sourceforge.net/projects/codesh/` .