# Enhancing Collaboration in Large Scientific Projects through Virtual Logbooks

Dimitri Bourilkov, *Member, APS* and Sharad Sonapeer

***Abstract*—A key feature of collaboration in large scale scientific projects is to have a *log* of what and how is being done - for private use and reuse and for sharing selected parts with collaborators and peers, often distributed geographically on an increasingly global scale. Even better if this log is *automatic*, created on the fly while a scientist or software developer is working in a habitual way, without the need for extra efforts. The `CAVES` - Collaborative Analysis Versioning Environment System - and `CODESH` - COllaborative DEvelopment SHell - projects address this problem in a novel way. They build on the concepts of *virtual state* and *virtual transition* to enhance the collaborative experience by providing automatic persistent virtual logbooks. `CAVES` is designed for sessions of data analysis using the popular `ROOT` framework, while `CODESH` generalizes the same 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. As a typical use case we concentrate on building working systems for analysis of Petascale volumes of data expected soon with the start of the LHC experiments. Our approach is general enough to find applications in many scientific fields.**

**Heraclitus asked: How can you bathe in the same river twice?**

**Quine answers: It's easy, though it is hard to bathe in the same water twice.**

## I. Introduction

THE scientific and software development processes demand the precise tracking of how a project is evolving over time, in order to be able to explore many different alleys simultaneously, moving forward to well defined states when successful - or rolling back otherwise. In this context a virtual session is the process of moving from a well defined initial to a well defined final state. 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 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 using virtual data for high energy physics analysis [1]. In our approach not only data comes complete with a recipe how to (re)produce it, but we put more emphasis on the interactive aspect of work done by users in their habitual ways, creating the log for the session *automatically, on the fly,* while the work is progressing. There is no need per se to provide any code in advance, but the user can execute/modify preexisting programs if desired. When a piece of work is worth recording, the user logs it in a persistent session repository with a unique identifier for later use/reuse.

Tools like this are vital to facilitate efficient collaboration in today's large, geographically distributed teams with their needs to be able to advance a project anytime and anywhere without space or time restrictions. Consider e.g. the scenario where thousands of researchers spread over different continents are working together on projects like the Large Hadron Collider [2], the most powerful particle accelerator built so far, expected to start data taking in 2008. In such a scenario, there is a need for efficient means of storing the data and methods used to create this data, and sharing these stored sessions between the collaborators. The `CAVES` and `CODESH` projects [3], [4], [5], [6], [7] build tools to address this problem.

## II. Project Outline

The basic idea behind the two projects [1] is the same and they share the same architecture. `CAVES` is designed specifically for users performing data analysis with the widely popular analysis package `ROOT` [8]. 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 the environment is considered as available or provided by the collaborating group. The splitting between logbook and environment parts 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

D. Bourilkov and S. Sonapeer are with the University of Florida, Gainesville, FL, 32611, USA e-mail: (see http://cern.ch/bourilkov).

Manuscript received November 21, 2007. The study is supported in part by the United States National Science Foundation under grant NSF 0427110 (UltraLight).

[1]For more details about the evolution of our design we refer the reader to [3], [4].

fundamental concept is to store information in enough detail to provide a share/replay mechanism, optionally modifying the inputs, for user's sessions at any other place or time by other users.

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 tags the complete log, which automatically collects the source program files, and optionally the data used, with a uniquely generated tag and logs it to a repository. Thus the repositories can contain hundreds and thousands of such stored sessions. Reproduction of a session is possible by extracting the log, data and source files, executing the commands listed in the log files and running the scripts that have been downloaded. Also the environment changes can be carried across sessions.

The repositories of such sessions can be on the local machine for personal usage and also on shared servers for the use of collaborating groups. Generally the user will store all his sessions locally and 'publish' selected important sessions to shared repositories. He/she may also extract and re-produce sessions stored by other collaborators.

There is also 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. 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.

## III. Architecture

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 other Tiers and it makes use of only the interfaces provided by the other Tiers. Thus we can change one Tier without affecting the operation of the others. The CODESH architecture is shown in Fig. 1. As most details when describing the architecture are common for the two projects, we will concentrate on the CODESH description, mentioning CAVES only where necessary to highlight the distinct nature of each project. Lets examine each Tier in turn:

**A. The User Tier**: This mainly implements the User Interface. We provide an interface similar to the Unix/Linux command line shell or to the ROOT command line. The user can start his session either in the batch mode or in interactive mode. In the interactive mode, the user types shell (or ROOT) commands just as he/she would on a Unix/Linux shell. He/she also types CODESH (CAVES) commands for the 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. Also the results produced 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 solely 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 the session information and also to get some status information e.g. a listing of all the sessions that have been stored. 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 sessions. We broadly classify this state information in two categories:

1. System information: This includes the aliases and environment variables that need to be kept track of during the session. We track the changes made to these during the session and provide routines for propagating them and also for logging them along with the session.

2. User Configuration information: This includes the various user-selected configurations. Some options provided for customized behavior of CODESH are:

**Loglevel**: Specifies how much to log

**Codeshloglevel**: Specifies whether to log CODESH commands in addition to the shell commands which are always logged

**Debuglevel**: Specifies how much debugging information to print on the screen

**Batchmode**: Option to enable/disable the batch mode operation

**Username**: Allows changing the user name used for tagging the sessions

**ExtensionList**: Maintains a list of all extensions treated as scripts.

iv. Extract module: This module is responsible for the extraction of the 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 easily added.

**C. The Backend**: The Backend stores the sessions, in such a way that they can be re-created later by some other user who extracts a session. For each session, we store the log files, the source files and optionally the data files. Each session is identified by a unique identifier which consists of 3 parts: the

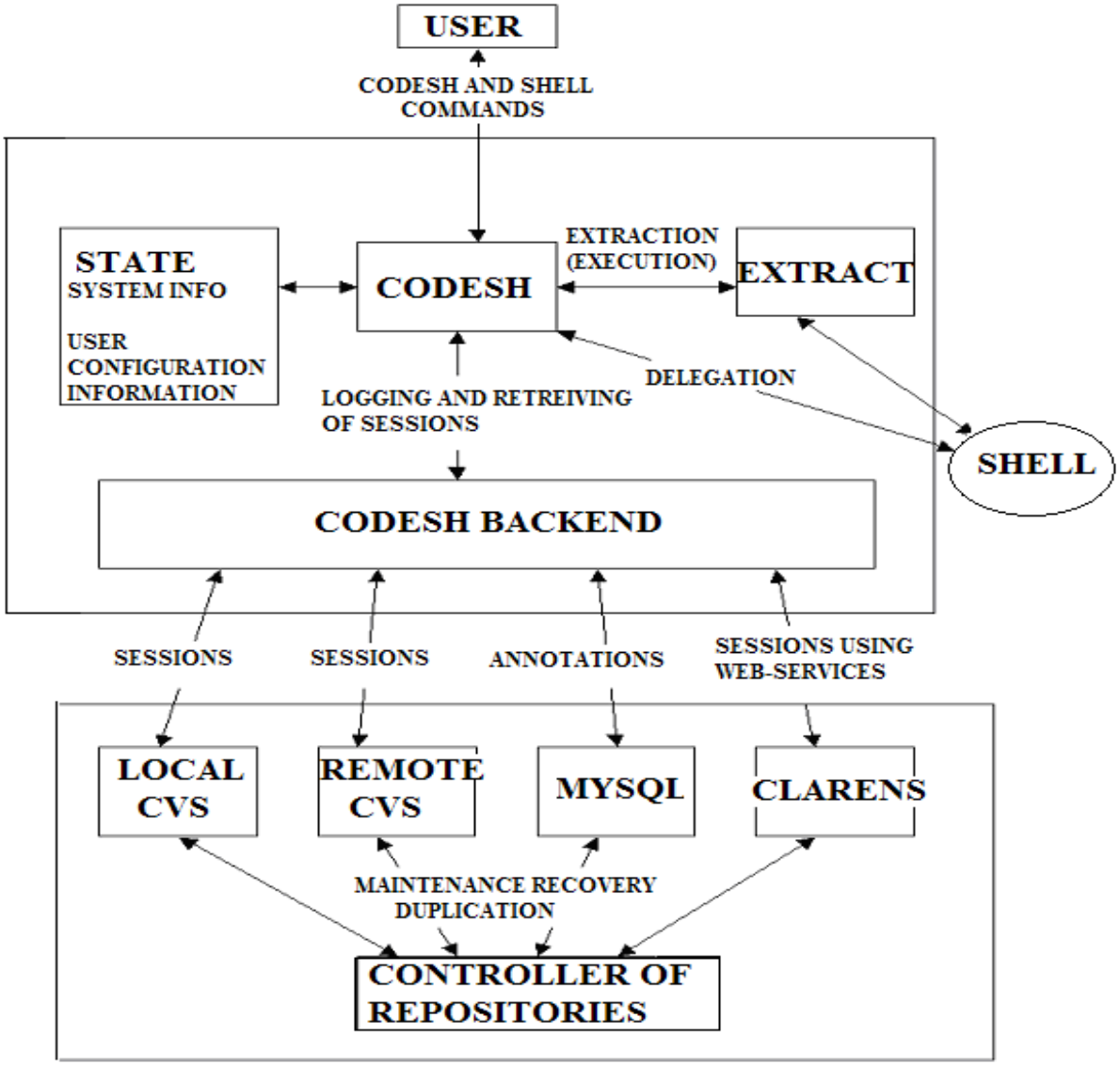


Fig. 1. *The scalable and distributed* CODESH *architecture.*

user's name, the current date and time and a user supplied name for the session. We provide support for three different types of backends:

i. ASCII: We provide the use of flat files (text or binary) as the main backend for storing the sessions. The repository can be local (on disk volumes mounted by 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 [9] as a version control type backend for storing the sessions. Using the CVS checkin, checkout and other commands we implement our commands like Log session and Extract session (for a comprehensive listing of CODESH commands refer to the next Section). We also provide an option of using Remote CVS as a backend i.e. using a pserver.

iii. SVN: We use Subversion [10] as an alternative version control type backend for storing the sessions. This backend is under active development and the first release will be out before the end of 2007.

iv. MySQL: We provide for a MySQL backend, which stores only the metadata information regarding each session in the Database. These annotations help in fast searches through stored sessions for some particular session types. After such a session is found, a local or remote CVS repository can be contacted to fetch the complete session.

Every user may have local e.g. ASCII or CVS repositories where he/she stores all personal sessions. Typically the user will want to commit some of the sessions to shared remote repositories and also extract some sessions stored by other users at the shared repositories. Thus we provide support for copying and moving sessions between repositories and also deleting sessions stored at some particular repository. We also plan to provide a way of cloning entire repositories or

converting them between different types.

Controller of the Repositories: This module takes care of the maintenance, recovery and similar tasks related to the different repositories. Our design structure is distributed in nature and thus we can have numerous controllers instead of just one centralized controller.

## IV. Typical Usage Scenario

Currently CODESH is implemented in the Python, and CAVES in the C++ programming language. 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. He can specify the different loglevels and other such customizations in a Codesh.conf file or specify them after he runs CODESH. In the interactive mode, he then views a command line interface on which he can start his session. Optionally he can use the batch mode and specify a file, which contains all the commands that are to be executed in a batch. To assist in logging his work and re-creating the results of previously stored sessions, he can use the various CODESH commands provided. Some of these are:

i. Browse: To list all the stored sessions and also optionally restrict the search depending on date/time or user. Also used to browse metadata associated with the sessions.

ii. Inspect: To view the contents of a particular session and optionally download all the source files.

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

iv. Log: To log a session between user defined checkpoints along with (optionally) all or some of the programs executed during the session. The level of logging depends on user defined log levels for that particular session.

v. Tagcopy: To copy a stored session from the source repository to the destination repository.

vi. Tagdelete: To delete a stored session.

vii. Takesnapshot: To work in a separate sandbox and store the entire sandbox in a repository. This complete sandbox i.e. all the files and directories under the working directory, can then be retrieved later by the same user or some other user and worked upon.

viii. Getsnapshot: To retrieve any previously stored sandbox. This is very useful in cases where a previously stored sandbox can be re-created at various places, by various people and all of them can start working from the place where the original user had left.

ix. Browsesnapshots: This command lists all (or a selected subset of) the sandboxes committed by all the users.

x. Setenv, Getenv: To access and modify environment variables independent of the underlying shell.

xi. GetAlias: To get all the active aliases.

## V. Test Results

Tools like CODESH or CAVES, designed to be used for collaborative development by many users, have to deal with different styles or work preferences and customizations by many individual users. These differences can be e.g. in the underlying shells used, level of logging/debugging desired, kind of work, user permissions and so on. We have developed CODESH and CAVES as flexible tools, easy to customize and extend with new user defined commands. We have tried to exhaustively test them with many different customizations using as backend local or remote ASCII and CVS repositories.

We have also done Stress testing for scenarios where the size and quantity of the logged data was really overwhelming. We have tested CODESH till the size of the repository was 10,000 sessions. Our code is resilient enough that even with 10,000 sessions in the repository the performance was only marginally slower than with very few sessions in the repository. All the sessions stored were of similar type and size. Specifically with 10 sessions stored in the repository, the inspection of a session took around 1 second, and even with 10,000 sessions stored, the inspection of a session took only 2 seconds on a 1 GHz Pentium III machine for a local repository.

We have built a fully distributed data analysis system based on ROOT and CAVES. The virtual sessions are stored in local or remote (pserver based) CVS repositories. The input and output datasets are stored using xrootd [11] servers. In this way users can browse, inspect and reproduce the sessions of their colleagues starting from a completely clean slate on a new desk- or laptop. All the necessary knowledge, code and data are delivered from the remote servers. An example of an event display from a Monte Carlo simulation for the CMS experiment [12] at LHC produced in this way is shown in Fig. 2.

## VI. Related Work

Archives typically keep final states. Often it is unclear how they were created. In computers the initial and final states are transient. Knowledge is not recorded systematically. Paper logs are hard to produce, hard to search, hard to share. They need transfer to a computer before making the knowledge widely available.

The history mechanism in typical shells like tcsh or bash logs a pre-defined number of commands in a file. But it provides no persistency mechanism for storing sessions or for exchanging them between collaborators. The scripts executed during a session, the pre- and post- conditions are not logged. The script [13] utility goes one step further, logging the standard output from the commands as well, all the rest is left to the user. Our automatic logbook/virtual sessions approach does much more by managing private and shared repositories of complete session logs, including the commands and the programs, and the ability to reproduce the results or reuse them for future developments.

At the other end contrary to most existing provenance systems which use disclosed provenance like annotations, transformations or workflows, an observed provenance approach at the kernel and system call level can be developed. Here substantial challenges like provenance granularity, versioning, provenance pruning, overheads etc. need to be overcomed. Our approach taken with the CODESH project offers flexibility and

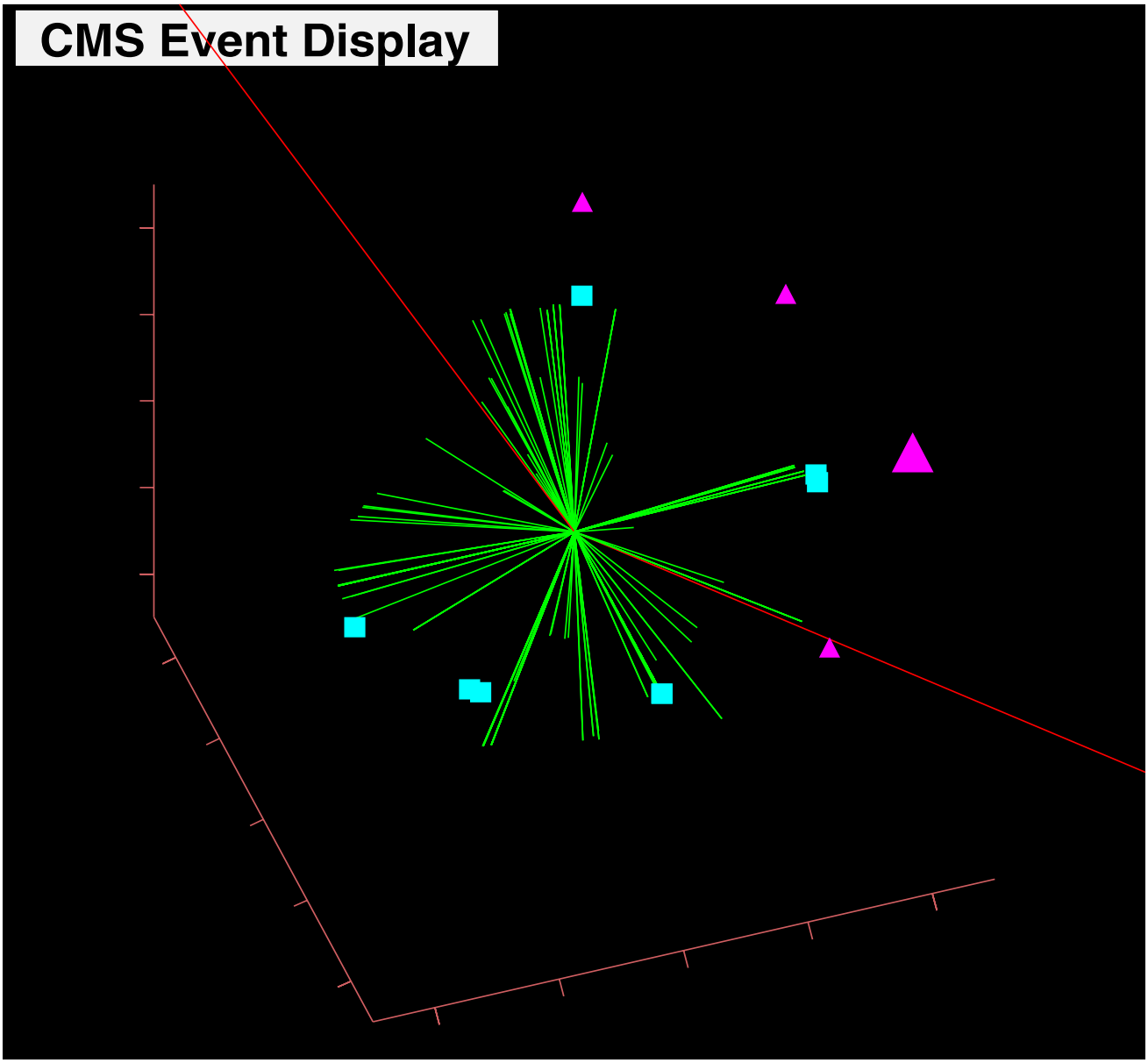


Fig. 2. *Event display from a Monte Carlo simulation for the CMS experiment. Charged particle tracks are shown in green, muon tracks in red, reconstructed jets with different algorithms as triangles and squares.*

good balance between observed and disclosed provenance: the users of our system can select the splitting between logbook and environment parts, which we call *collaborating contract*, depending on their needs. We provide observed provenance at the shell level, which, due to its proximity to the users, offers a rich semantic knowledge by seamlessly observing the work done in a virtual session. The adoption of a versioning system like `CVS` or `Subversion` as a persistent backend helps in solving the versioning problem and provides an elegant approach to pruning by only storing the differences between a potentially large number of similar sessions. In summary, the `CODESH` project combines in a natural way some of the key desired features of the two extremes outlined above.

## VII. Current and Future Work

We have used `CODESH` to record the production and analysis of data for large scale simulations in high energy physics, and for various software development and configuration tasks at several locations distributed across the United States. `CAVES` was used to record analysis sessions of the produced data, including analyses demonstrated at the Supercomputing conferences in 2005 and 2006.

Our ongoing and future work consists e.g. of implementing:

- all the robust functionality available with the `ASCII` and `CVS` backends in the `Subversion` case
- the full set of administrative tasks for the management, maintenance, conversion, cloning and control of private and shared repositories
- Web interfaces for users to browse through the reposito-

ries

- automatically converting session logs to workflows, and the ability to develop locally and seamlessly schedule more CPU/data intensive tasks on grid infrastructure.

Information about public releases of our functional systems for automatic logging of typical working sessions is available from [14], and the projects are hosted as open source [15].

In summary, our projects take a pragmatic approach in assessing the needs of a community of scientists or software developers by building series of working packages with increasing sophistication. By extending with automatic logbook capabilities the functionality of a typical `UNIX` shell (like `tcsh` or `bash`) - the `CODESH` project, or a popular analysis package as `ROOT` - the `CAVES` project, these packages provide an easy and habitual entry point for researchers 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] A. Arbree *et al.*, "Virtual data in CMS analysis," *In the Proceedings of 2003 Conference for Computing in High-Energy and Nuclear Physics (CHEP 03), La Jolla, California, 24-28 Mar 2003, pp TUAT010* [arXiv:physics/0306008].

[2] The Large Hadron Collider close to Geneva, Switzerland, will collide proton-proton beams at energies of 14 TeV starting in 2008; http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/.

[3] D. Bourilkov, "The CAVES project: Exploring virtual data concepts for data analysis," http://arxiv.org/abs/physics/0401007, arXiv:physics/0401007.

[4] 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].

[5] 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.

[6] 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.

[7] 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.

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

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

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

[11] xrootd home page, http://xrootd.slac.stanford.edu/ .

[12] The CMS experiment at CERN, http://cms.cern.ch/iCMS/ .

[13] The script utility appeared in Berkeley Unix 3.0 BSD.

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

[15] http://freshmeat.net/projects/codesh/ http://sourceforge.net/projects/codesh/ .