Virtual States and Transitions, Virtual Sessions and Collaboration

Dimitri Bourilkov

University of Florida, Gainesville, FL 32611, USA
`bourilkov@phys.ufl.edu`

Abstract. A key feature of collaboration is having a *log* of what and how is being done - for private use/reuse and for sharing selected parts with collaborators in today's complex, large scale scientific/software environments. 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, building on the concepts of *virtual state* and *virtual transition* to provide an automatic persistent logbook for sessions of data analysis or software development in a collaborating group. Repositories of sessions can be configured dynamically to record and make available in a controlled way the knowledge accumulated in the course of a scientific or software endeavor.

1 Introduction

Have you sifted through paper folders or directories on your computer, trying to find out how you produced a result a couple of months (or years) ago? Or having to answer a question while traveling, with your folders safely stored in your office? Or a desperate collaborator trying to reach you about a project detail while you are hiking in the mountains? It happened many times to me, and there must be a better way.

2 Automatic Logbooks

We explore the possibility to create an automated system for recording and making available to collaborators and peers the knowledge accumulated in the course of a project. The work in a project is atomized as *sessions.* A session is defined as a transition T from an initial $|I>$ to a final $|F>$ state: $|F>=T|I>$. The work done during a session is represented by the transition operator T. The length of a session is limited by the decision of the user to record a chunk of activity. A state can be recorded with different level of detail determined by the users: $|State>=|Logbookpart, Environment>$. The system records automatically, in a persistent way, the logbook part for future use. 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. A good splitting is characterized by the transition operator T acting in a meaningful way on the logged part without affecting substantially the environment.

At sufficient level of detail, each final state $|F>$ can be reproduced at will from the log of the initial state $|I>$ and the transition T. The ability to reproduce states bring us to the notion of *virtual state* and *virtual transition*, in the sense that we can record states that existed in the past and can recreate them on demand in the future. Virtual states serve as *virtual checkpoints*, or delimiters for *virtual sessions*. In a traditional programming language, the user typically codes the transition T, i.e. provides in advance the source of the program for producing a given final state, following the language syntax. The key idea in our approach is that the user works in a habitual way and the log for the session is created *automatically, on the fly,* while the session is progressing. There is no need per se to provide any code in advance, but the user can execute preexisting programs if desired. When a piece of work is worth recording, the user logs it in the persistent session repository with a unique identifier [1].

Let us illustrate this general framework with an example. We want to log the activities of users doing any kind of work on the command line e.g. in a UNIX shell. At the start of a session the system records details about the initial state. Then the user gives commands to the shell (e.g. ls, cd, cp, find etc). During the session users could run some existing user code, possibly providing input/output parameters when invoking it. The system collects and stores all commands and the source code of all executed scripts as used during the session, automatically producing a log for the transition T. Later users may change any of the used scripts, delete or accidentally lose some of them, forget which parameters were used or why. When the same or a new user wants to reproduce a session, a new sandbox (clean slate) is created and the log for a given unique identifier is downloaded from the repository. In this way both the commands and the scripts, as well as the input/output parameters, "frozen" as they were at the time of the session, are available, and the results can be reproduced.

3 The CODESH/CAVES Projects - Working Implementations of Automatic Logbooks

The CODESH [2] and CAVES [3] projects take a pragmatic approach in assessing the needs of a community of scientists or software developers by building series of working prototypes 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 [4] - the CAVES project,

[1] The virtual data idea explores similar concepts, putting more emphasis on the transition - abstract transformation and concrete derivations, coded by the users in a virtual data language [1].

these prototypes 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.

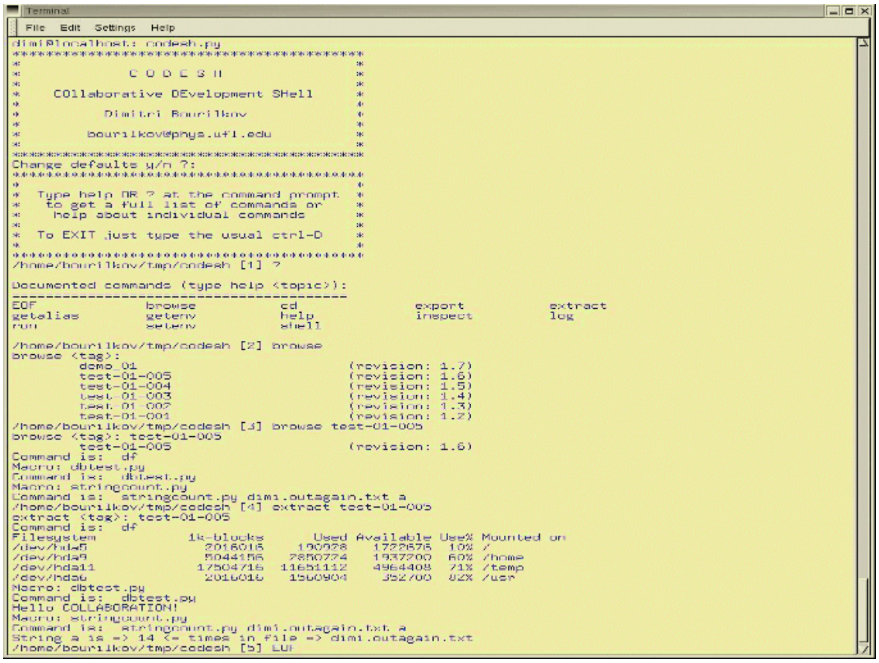
Both projects use a three-tier architecture, with the users sitting at the top tier and running what looks very much like a normal shell or identical to a `ROOT` session, and having extended capabilities, which are provided by the middle layer. This layer is coded in `Python` for `CODESH` or in `C++` for `CAVES`, by inheriting from the class which handles the user input on the command line. The implemented capabilities extend the example in Sect. 2. The command set is extensible, and users can easily write additional commands. The lower tier provides the persistent back-end. The first implementations use a well established source code management system - the Concurrent Versions System `CVS`, ideal for rapid developments by teams of any size. The `CVS` tags assume the role of unique identifiers for virtual sessions, making it possible to extract session information identical to the one when the session log was first produced.

The `CAVES/CODESH` systems are used as building blocks for collaborative analysis/development environments, providing "virtual logbook" capabilities and the ability to explore the metadata associated with different sessions. A key aspect of the project is the distributed nature of the input data, the analysis/development process and the user base. This is addressed from the earliest stages. Our systems are fully functional both in local and remote modes, provided that the necessary repositories are operational and the datasets available. This allows the users to work on their laptops (maybe handhelds tomorrow) even without a network connection, or just to store intermediate steps in the course of an active analysis/development session for their private consumption, only publishing a sufficiently polished result. This design has the additional benefit of utilizing efficiently the local CPU and storage resources of the users, reducing the load on the distributed services (e.g. Grid) system. The users have the ability to replicate, move, archive and delete session logs. Gaining experience in running the system will help to strike the right balance between local and remote usage.

A possible scenario is that a user chooses during a session to browse through the tags of other users to see what work was done, and selects the log/session of interest by extracting the peers log. Here two modes of operation are possible: the user may want to reproduce a result by extracting and executing the commands and programs associated with a selected tag, or just extract the history of a given session in order to inspect it, possibly modify the code or the inputs and produce new results.

Our first releases are based on a quite complete set of commands providing interesting functionality. Screenshots from a real session are shown in Fig. 1.

In this paper we outline the main ideas driving the `CODESH/CAVES` projects. The history mechanism or the `script` utility of `UNIX` shells can just log the commands and standard output of a working session in a private file, leaving all the rest to the users. Our automatic logbook/virtual session 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 re-



```
dimi@localhost: codesh.py
*********************************************
*                                           *
*                C O D E S H                *
*                                           *
*    COllaborative DEvelopment SHell        *
*                                           *
*            Dimitri Bourilkov              *
*                                           *
*          bourilkov@phys.ufl.edu           *
*                                           *
*********************************************
Change defaults y/n ?:
*********************************************
*                                           *
*  Type help OR ? at the command prompt     *
*   to get a full list of commands or       *
*    help about individual commands         *
*                                           *
*  To EXIT just type the usual ctrl-D       *
*                                           *
*********************************************
/home/bourilkov/tmp/codesh [1] ?

Documented commands (type help <topic>):
========================================
EOF             browse          cd              export          extract
getalias        getenv          help            inspect         log
run             setenv          shell

/home/bourilkov/tmp/codesh [2] browse
browse <tag>:
        demo_01                              (revision: 1.7)
        test-01-005                          (revision: 1.6)
        test-01-004                          (revision: 1.5)
        test-01-003                          (revision: 1.4)
        test-01-002                          (revision: 1.3)
        test-01-001                          (revision: 1.2)
/home/bourilkov/tmp/codesh [3] browse test-01-005
browse <tag>: test-01-005
        test-01-005                          (revision: 1.6)
Command is:  df
Macro: dbtest.py
Command is:  dbtest.py
Macro: stringcount.py
Command is:  stringcount.py dimi.outagain.txt a
/home/bourilkov/tmp/codesh [4] extract test-01-005
extract <tag>: test-01-005
Command is:  df
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda5              2016016    190928   1722676  10% /
/dev/hda9              5044156   2850724   1937200  60% /home
/dev/hda11            17504716  11651112   4964408  71% /temp
/dev/hda6              2016016   1560904    352700  82% /usr
Macro: dbtest.py
Command is:  dbtest.py
Hello COLLABORATION!
Macro: stringcount.py
Command is:  stringcount.py dimi.outagain.txt a
String a is => 14 <= times in file => dimi.outagain.txt
/home/bourilkov/tmp/codesh [5] EOF
```

Fig. 1. Example of a `CODESH` session

sults or reuse them for future developments. Public releases of the first functional systems for automatic logging in a typical working session are available for interested users. Possible directions for future work include adding different back-ends - SQL, web or grid service oriented [5], using GSI security, automatically converting session logs to workflows, the ability to develop locally and seamlessly schedule more CPU/data intensive tasks on grid infrastructure.

The study is supported in part by the United States National Science Foundation under grants NSF ITR-0086044 (GriPhyN) and NSF 0427110 (UltraLight).

References

1. Foster, I. *et al.*: Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In: 14th International Conference on Scientific and Statistical Database Management (SSDBM 2002), Edinburgh, 2002
2. Bourilkov, D.: THE CAVES Project - Collaborative Analysis Versioning Environment System; THE CODESH Project - Collaborative Development Shell. arXiv:physics/0410226 ; http://xxx.lanl.gov/abs/physics/0410226
3. Bourilkov, D.: The CAVES Project: Exploring Virtual Data Concepts for Data Analysis. arXiv:physics/0401007, and references therein ; http://xxx.lanl.gov/abs/physics/0401007
4. Brun, R. and Rademakers, F.: ROOT - An Object Oriented Data Analysis Framework. Nucl. Inst. & Meth. in Phys. Res. **A 389** (1997) 81–86
5. Grid-enabled Analysis Environment project: http://ultralight.caltech.edu/gaeweb/