Hydra Version Control System

Christoph P. Neumann, Scott A. Hady, Richard Lenz
Computer Science 6 (Data Management)
Friedrich-Alexander University, Erlangen, Germany
christoph.neumann@cs.fau.de

***Abstract*—The Hydra project offers version control for distributed case files in α-Flow. Available version control systems lack support for independent versioning of multiple logical units within a single repository each with its own version history and head. Our use case also requires mechanisms for labeling versions by their validity and for validity-based navigational access. Hydra is a multi-module and validity-aware version control system.**

***Keywords*-Versioning, Multi-Module, Case Files, Valid Paths**

I. Introduction & Background

The research project α-Flow (cf. [1], [2]) provides an approach to distributed electronic case files, called α-Docs. α-Docs are distributed by storing a local copy for different cooperating parties and by synchronizing updates and extensions among these copies (cf. [3]). The α-Doc contains its own execution environment (cf. [4]); its subsystem implementations must be as small as possible. The motivation for α-Docs is not important in this context. From the perspective of version control systems (VCS), the α-Doc contains a repository that is structured into data modules, so called α-Card units. Each data module, i.e. each α-Card unit, is an independent set of hierarchically structured files.

Hydra provides embedded versioning for the α-Flow engine within an α-Doc. Hydra has been implemented as an autonomous component that can also be used by a command-line interface as a stand-alone VCS. The unique functional features of Hydra, 1) multi-module support and 2) validity-awareness, are derived from our use case, yet, the reasoning is of general concern.

II. Objectives

Healthcare processes are paper-based and there exist logical units (LU) of paper artifacts. Each such set exhibits an owner who is at least the contact person or who even takes legal responsibility. In an inter-institutional medical process, several LUs constitute the patient's case file. Each LU can be considered as a kind of data module. In an electronic document analogon, the version history of each LU must be available independently for data provenance purposes. Thus, each data module requires its independent VCS history, however, the overall team progress, i.e. data production over all data modules, must also remain track-able. This is basically the same situation as in parallel software development with conflicting updates (on medical content files instead of source code files) and with grouping artifacts via LUs. The 'LU' is a unified term for 'data module' or 'software module'.

The Hydra objective is to provide a generic VCS concept for (1) managing multiple LUs within a single repository. A LU is defined as an arbitrary set of hierarchically structured files. Within a single repository, both (i) an independent version history, navigation, and check-out head must be kept for each LU, and (ii) a common version state over all LUs must be provided for module-interdependency maintenance. The second Hydra objective is to allow for (2) labelling versions by a valid/invalid flag and to enable *validity-based version navigation*. This means to provide both (i) a *system path navigation* with any and all version states as it is provided by common VCS navigation and (ii) a *valid path navigation* that operates only on all valid versions.

Validity-aware version navigation is important in healthcare because in inter-institutional environments physicians are willing to provide to their peers preliminary information. Preliminary versions are invalid in terms of "not signed off" and their content should only be consumed if treated with discreet caution. Validity can imply acceptability instead of formal correctness.

The validity facilities of Hydra are also required by the α-Flow protocol for synchronizing distributed α-Doc nodes (cf. [3]). Hydra facilitates the protocol implementation: in case of global conflicts the conflicting versions can simply be marked as invalid. The invalidated versions are required for reconciliation, for which they remain accessible by Hydra's system path navigation. Yet, until the conflict is reconciled, the valid path provides the team members with access to the latest globally valid, i.e. conflict free, version.

III. Methods

The Hydra versioning is inspired by Git (e.g. [5]) and its object model[1]. Hydra also adopts full copy storage

[1] e.g. http://eagain.net/articles/git-for-computer-scientists

 DOI: 10.1109/XXXXXXXXX The original publication is available at http://ieeexplore.ieee.org/xpls/abs_all.XXXXXXXXXXXx

and content-addressable storage via hashing. Further versioning approaches such as SCCS, RCS, CVS, SVN, and Mercurial had been evaluated; a survey can be gained by Mukherjee in [6].

Validity is highly related to visibility. Common databases apply "validity-first" based on their well-known ACID[2] properties. Validity implies visibility: new data can only be made visible by a commit if the changes are valid by not failing any constraints or assertions. Decoupling validity and visibility in databases had been subject to *database conversions* by Kirsche (e.g. [7]). Common VCS apply "visibility-only", new data is made visible without a notion of validity. If invalid versions need to be avoided then they must not be checked in. Branching support in VCS allows to separate alternate variants of a common ancestor[3]. Branches are reused for separating different scopes with different consensus on their validity criteria. The problem remains the same: there is still no way to articulate whether a branch check-in matches the agreed upon criteria or not. Validity assertion relies on user-applied constraints and checks. In contrast, a *validity-aware VCS* needs to provide two levels of visibility based on validity differentiation. In combination with branching, different validity levels are possible. Integrating validity-awareness into the VCS layer has the potential to ease the implementation of a gated check-in build process in which check-ins are only made publicly visible if they pass automated code checks.

Support for multi-moduled versioning is required in any modular data architecture. A first alternative would be to manage modules by subdirectories. It mingles their version history. Updating several modules to the head version but letting other modules remain in a concerted version state becomes cumbersome and requires user discipline. A second alternative would be to separate modules into distinct repositories. Then, supporting a superproject repository requires mechanisms to reference distinct external repositories and to virtually merge them into a single working space. The distinct-repositories-alternative encumbers restructuring between the modules and interrupts version history at relocations. It does not provide a module-comprehensive version state. In contrast, a *multi-module VCS* needs to provide three levels of versioning granularity: the artifact, the logical unit, and the repository.

IV. Result

Hydra extends the Git object model. The original model consists of the class Object with subtypes Commit, Tree, and Blob as well as Reference as a named relationship. Trees and blobs form a hierarchical structure; trees have reflexive parent associations. Commits link to multiple trees; additionaly, commits have reflexive previous associations.

We reimplemented the versioning object model in Java. Then we refined the Commit into subtypes LogicalUnit and Stage. The stage reference LUs, each with an arbitrary state, thus, the stage manages the module-interdependency and represents concerted superproject progress. Validity tracking is implemented by adding a second reflexive validPrevious association between Commit classes. A detailed Hydra description is given in [9] with all the subtle implications.

Hydra is available as open-source software. Its executable size is 213 kb instead of Git's 19 MB. In a stress test with 2,874 files comprising 983 MB of mixed binary and text documents, our NIO-based Zip-compressing Java implementation is 2.9 times slower than zlib-compressing C/C++-based Git; our repository is with 173 MB slightly smaller than Git's 187 MB.

[2] Atomicity, Consistency, Isolation, Durability

[3] Their exists a debate (cf. [8]) whether versions and variants are synonymous or orthogonal concepts. We consider the 4Vs (versions, variants, visibility, and validity) as orthogonal.

References

[1] C. P. Neumann and R. Lenz, "The alpha-Flow Use-Case of Breast Cancer Treatment – Modeling Inter-Institutional Healthcare Workflows by Active Documents," in *Proc of the 8th Int'l Workshop on Agent-based Computing for Enterprise Collaboration (ACEC)*, Jun. 2010.

[2] C. P. Neumann, P. K. Schwab, A. M. Wahl, and R. Lenz, "alpha-Adaptive: Evolutionary Workflow Metadata in Distributed Document-Oriented Process Management," in *Proc of the 4th Int'l Workshop on Process-oriented Information Systems in Healthcare*, Aug. 2011.

[3] A. M. Wahl and C. P. Neumann, "alpha-OffSync: An Offline-Capable Synchronization Approach for Distributed Document-Oriented Process Management in Healthcare," in *Lecture Notes in Informatics: Sem. 11*, L. Porada, Ed. GI, Mar. 2012, accepted for publication.

[4] A. Todorova and C. P. Neumann, "alpha-Props: A Rule-Based Approach to 'Active Properties' for Document-Oriented Process Support in Inter-Institutional Environments," in *Lecture Notes in Informatics: Sem. 10*, L. Porada, Ed. GI, Mar. 2011.

[5] J. Loeliger, *Version Control with Git.* O'Reilly, 2009.

[6] P. Mukherjee, "A Fully Decentralized, Peer-to-Peer Version Control System," Ph.D. dissertation, Technische Universität Darmstadt, 2005.

[7] T. Kirsche, R. Lenz, T. Ruf, and H. Wedekind, "Cooperative problem solving using database conversations," in *Proc of the 10th Int'l Conf on Data Engineering.* IEEE, 1994, pp. 134–143.

[8] H. Wedekind, "Are the terms "version" and "variant" orthogonal to one another?" *SIGMOD Rec.*, vol. 23, pp. 3–7, December 1994.

[9] S. A. Hady, "alpha-VVS: An integrated Version Control System as a Component of Process Support based on Active Documents," Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2011.