Support for Hypothetical Initiation and Dynamic Exploration using History of Operation in ViennaTalk

Tomohiro Oda[1]

Software Research Associates, Inc. (tomohiro@sra.co.jp)

Abstract. The specification phase of software development involves exploratory efforts with trial and error, especially in its early stages. While IDEs and engineering tools support the authoring and analysis of specifications, the burden of managing exploratory activities—such as enumerating possible design choices and testing each one—remains with specification engineers. Hi-De-Ho is a library-based framework that stores the history of design artifacts to support exploratory tasks in various fields, such as simulation modeling and data analysis. ViennaTalk Milan, released in April 2025, adopts Hi-De-Ho as a personal versioning tool, complementing Git as a collaborative versioning tool. This report provides an overview of Hi-De-Ho, its integration with ViennaTalk, and the division of roles between Hi-De-Ho and Git.

1 Introduction

Developing a formal specification is a creative process that involves both problem solving and the effective presentation of design intent. Specifications are often refined through iterative exploration and learning from failure. Thus, progress in the specification phase is not always incremental. A specification may sometimes face breakdowns, requiring some design decisions made so far to be retracted.

Deciding how to present a design is not straightforward, either. Specification languages allow multiple ways to define design constituents and the relationships among them. Finding a concise presentation of a specification is often exploratory and involves trial and error, since it is difficult to predict which form will best support further development.

ViennaTalk has been developed to support the exploratory nature of the specification phase by making effective and visual use of specification animation techniques [3]. It offers GUI prototyping, intuitive animation interfaces, and unit testing frameworks to promote agility in specification tasks. Two version management tools, Git and Hi-De-Ho, were integrated into ViennaTalk in its April 2025 release to strengthen support for exploration. Git is a widely used version control system for large-scale and distributed software development projects, such as the Linux kernel [1]. With Git, users can manage version propagation across multiple repositories to coordinate contributions from multiple developers.

Hi-De-Ho (Hypothetical Initiation and Dynamic Exploration using History of Operation) is a personal version management framework for creative work. Whereas Git is a command-line tool for managing source files written in general-purpose languages, Hi-De-Ho is a framework that client applications can use to implement language-specific

and task-specific versioning functionality. With every modification, evaluation, loading, or reverting operation in ViennaTalk, its integration with Hi-De-Ho automatically stores the specification source, the internal state of the specification animation, and metadata such as the date, the type of operation, and links to predecessor snapshots. Users can later refine these automatically generated versions by editing comments, adding tags, organizing them into folders, reverting to a previous version, or removing unnecessary ones.

This report explains ViennaTalk's integration with Hi-De-Ho. Section 2 describes the objectives of Hi-De-Ho and how it is integrated with ViennaTalk. Section 3 presents the implemented functionality along with its user interface. Finally, the current development status and future work are discussed in Section 4.

The source code of ViennaTalk is available at Github repository `https://github.com/tomooda/ViennaTalk` and its development branch at `https://github.com/tomooda/ViennaTalk/tree/dev`. The public repository of Hi-De-Ho is also at Github `https://github.com/tomooda/HiDeHo`. The installation packages can be downloaded from ViennaTalk's distribution site `https://viennatalk.org/builds/viennatalk/milan` for Milan and `https://viennatalk.org/builds/viennatalk/dev` for the development version.

2 Objectives

ViennaTalk is an IDE for VDM-SL [2] designed to support the exploratory stage of the formal specification phase. VDM engineers often face uncertainty, where multiple design choices are available but it is difficult to determine the best one at the time. Specifiers may also need to revert the specification to an earlier version, prior to the point where a wrong decision was made. The integration of ViennaTalk with Hi-De-Ho aims to provide lightweight and flexible version management as a personal tool to support such exploratory tasks.

Version management tools are generally expected to help users move back and forth between versions. Conventional tools like Git support both personal and collaborative development. For instance, Git was originally designed to support the development of the Linux kernel. It is a powerful tool that enables skilled engineers to coordinate code contributions across a development community. However, using Git requires users to follow specific steps: staging files, committing with messages, and pushing to remote repositories. Moreover, Git treats all files as plain text or binary data without knowledge of language-specific structures. Its power comes at the cost of requiring careful and deliberate workflows.

Exploratory work, involving trial and error, does not always align with such planned workflows. Sterman et al. [4] identified the four roles of version management tools in creative work: versions as a palette of materials, confidence to explore, lightweight snapshotting, and long-term reflection. Sterman also pointed out that conventional version management tools do not align well with creative workflows and fails to adequately capture the explorative process. Hi-De-Ho was integrated into ViennaTalk to offer more flexible versioning: it automatically records every modification, presents changes in terms of VDM-SL modules and top-level definitions rather than file names and line

numbers, and supports history management without requiring explicit user actions or predefined workflows.

In Hi-De-Ho, a version is called a *snapshot* of the design artifact. Each snapshot includes *metadata* such as timestamps, the type of operation, and predecessor links, enabling effective browsing, searching, and organizing of the version history. A snapshot is automatically recorded not only when the specification is modified, but also when expressions are evaluated, statements are executed, commits are made to Git, files are loaded from disk, or older versions are restored. Snapshots are captured as a side effect of these actions to minimize the user's effort in managing history while working on specifications. Users can later browse, search, and annotate the stored snapshots with comments and tags, if needed.

3 ViennaTalk's Integration with Hi-De-Ho

ViennaTalk is designed to support users in exploratory modeling, helping them to better understand the system under development and to incorporate feedback into their models. To achieve this, ViennaTalk provides refactoring operations such as extracting value definitions and renaming identifiers, allowing users to make modifications quickly and focus on semantic-level changes rather than low-level text editing.

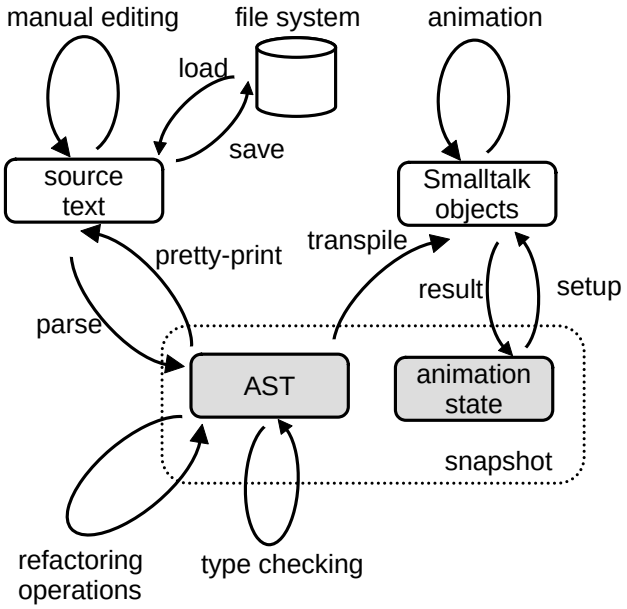

Fig. 1. AST-centric architecture of ViennaTalk

Figure 1 illustrates how the abstract syntax tree (AST) of a VDM-SL specification plays a central role in ViennaTalk's functionality. Unlike conventional IDEs that let users edit source files and pass them to compilers or interpreters, ViennaTalk maintains the specification internally as an AST and generates other representations on demand. For example, when a user edits the textual definition of an operation, ViennaTalk pretty-prints the corresponding AST node with syntax highlighting. After editing, the text is parsed back into an AST, replacing the corresponding part of the specification. Refactoring operations directly manipulate the AST. The type checker performs inference on expression nodes and annotates them with static type information.

Animation is also implemented in an AST-centric manner. When evaluating an expression, ViennaTalk transpiles the AST to Smalltalk classes, instantiates them with the current animation state, and evaluates the corresponding Smalltalk expression. The result is returned along with the updated animation state and shown in the browser.

ViennaTalk maintains both the specification AST and the animation state. Snapshots, which include both, are recorded to support reverting not just code but also the animation state during exploratory testing.

In the rest of this section, the term *source specification* refers to an AST of a VDM-SL specification. The term *animation state* refers to a mapping from global state variable names (e.g., *module-name* `*identifier*) to their runtime values. A *model* is a pair of a source specification and animation state. A *snapshot* is a model enriched with metadata and supplemental data, stored as a persistent record in the filesystem.

3.1 Storage

Hi-De-Ho's storage layer assigns a unique ID to each snapshot and stores them in a space called the *chronicle*. The chronicle maintains a linear sequence of snapshots based on creation time and provides an API to enumerate snapshot metadata, enabling host applications to locate desired snapshots efficiently. In addition to the chronicle, Hi-De-Ho offers *narratives*—folders that store only user-associated snapshots. While the chronicle contains all snapshots, narratives are populated via the `addToNarrative` API.

ViennaTalk stores snapshots under the `.hideho` directory at the base of the model. Snapshots are placed in subdirectories under `chronicle`. Each snapshot directory contains the files listed in Table 1.

Table 1. Contents of a snapshot directory

Filename	Format	Description
`metadata.json`	JSON	Metadata describing the snapshot
`source.vdmsl`	VDM-SL	The complete source specification
`state.json`	JSON	Mapping of state variables to their values

Each snapshot includes metadata fields as shown in Table 2. This metadata enables snapshot selection using tags, timestamps, or comments without requiring the full specification or state to be loaded.

Table 2. Metadata fields in a snapshot

Field	Type	Description
`id`	String	Unique snapshot ID
`timestamp`	String	ISO 8601 creation timestamp
`tags`	Array of String	User-defined tags
`comment`	String	User comment describing the snapshot
`class`	String	Type of operation that created the snapshot
`prev`	String	ID of the previous snapshot
`destination`	String	Reversion target snapshot ID (if applicable)

Snapshots are created during various operations in ViennaTalk. The `class` field indicates the operation that triggered the snapshot, as summarized in Table 3.

Table 3. Snapshot types and corresponding user operations

Value	Description
`"Vanilla"`	Created when initializing a new model
`"Loading"`	Loaded from disk
`"Modification"`	Edited manually or via refactoring
`"Initialization"`	State initialized
`"Evaluation"`	Expression evaluated
`"Execution"`	Statement executed
`"Reversion"`	Reverted to an earlier snapshot

3.2 Timelines

Like other version control tools, ViennaTalk maintains relationships between snapshots and lists them in chronological order using link structures. ViennaTalk distinguishes between two types of links: *predecessor* and *previous*. Figure 2 illustrates both the *artifact timeline* (based on predecessor links) and the *user timeline* (based on previous links) from the latest version `modification2`. The artifact timeline reflects the evolution of the model itself, while the user timeline reflects the sequence of user interactions. Please note that Fig 2 shows only predecessor links traceable from `modification2`. Predecessor links among all versions together form version trees.

Predecessor links are conceptually similar to those in systems like Git. When snapshot `A` is modified to produce snapshot `B`, `B`'s predecessor points to `A`. All snapshots except the initial `Vanilla` have a predecessor, typically created through specification changes, expression evaluation, or statement execution. The *previous* link, on the other hand, connects to the snapshot taken immediately prior to the current user operation—capturing the workflow history akin to the undo/redo chain in authoring tools.

Figure 3 shows ViennaTalk's specification browser UI. The upper section contains three lists: modules (left), sections (center), and top-level definitions (right), from which users can select items to edit. The lower section has tabbed pages: `Source`,

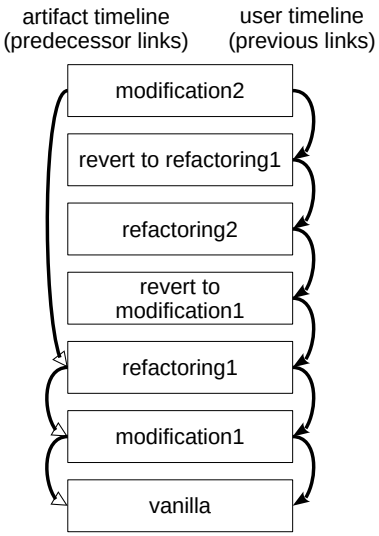

Fig. 2. Example of user timeline and artifact timeline

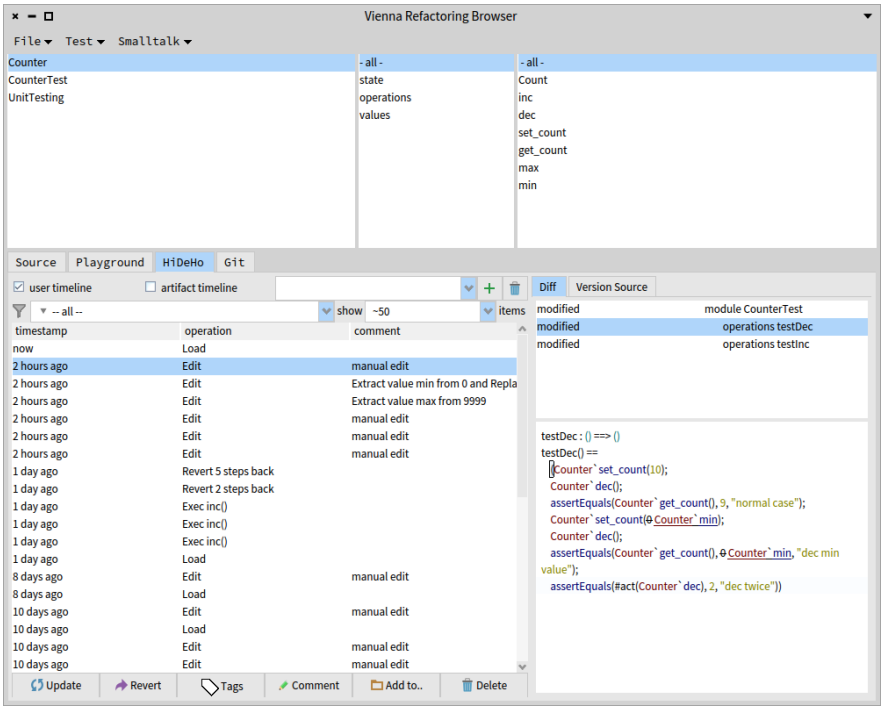

Fig. 3. ViennaTalk's UI with Hi-De-Ho integration

`Playground`, `HiDeHo`, and `Git`. The `HiDeHo` tab provides the version history interface.

At the top of the Hi-De-Ho page in the lower half of the browser window, users can choose a timeline—user timeline, artifact timeline, or a narrative. Snapshots in the selected timeline are listed on the left with timestamps, operation types, and comments. Selecting a snapshot displays the diff or full specification on the right-hand panel.

At the bottom of the Hi-De-Ho page, buttons to manage snapshots are horizontally layouted. The user can update the list, revert to the selected snapshot, add or remove tags at the snapshot, edit a comment, add the snapshot to a narrative, or delete the snapshot from the chronicle.

3.3 Collaboration with Git

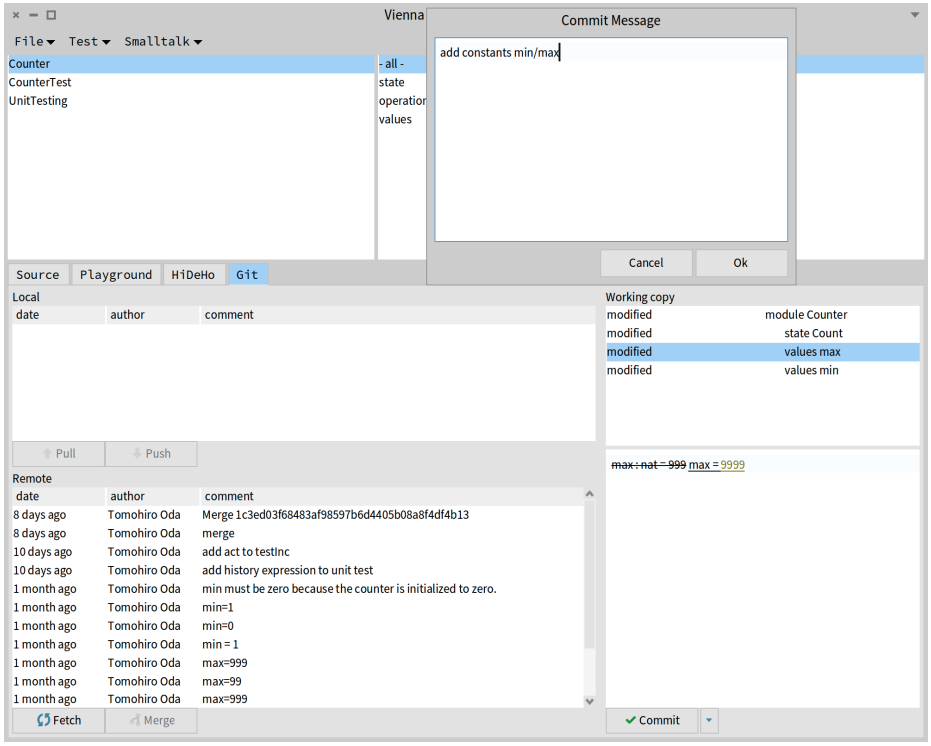

Fig. 4. A screenshot of UI for git

ViennaTalk is also integrated with Git. The user interface is shown in Fig. 4. Users can clone repositories from GitHub, pull from and push to the origin repository, commit the current working copy, and merge commits. These operations are explicitly per-

formed by the user, whereas Hi-De-Ho automatically records snapshots and allows the user to manage them afterward.

Although both Hi-De-Ho and Git are version management tools, they serve different purposes in ViennaTalk. The Git interface is intended for collaborative development, while Hi-De-Ho is designed to support personal exploration. Users can utilize Hi-De-Ho to try out ideas and refine models through trial and error before committing to the Git repository. Once a user is satisfied with the current version, they can commit it to Git and push it to the upstream repository. Snapshots stored in Hi-De-Ho can optionally be cleared using the checkbox at the bottom-right of the Git interface.

4 Concluding remarks and future plans

Version management tools are widely used in software development. However, the explorative nature of creative tasks—such as writing formal specifications—often makes version management a burden for individual developers. The integration of Hi-De-Ho into ViennaTalk aims to bridge this gap by providing lightweight, automatic, and semantically meaningful version control tailored to personal exploration.

While formal methods are typically studied from an engineering perspective, the author believe there is growing value in viewing formal specification as a form of creative authoring. Hi-De-Ho embodies a design philosophy that embraces uncertainty and iteration, supporting a more natural flow of creative thinking in formal modeling tasks. Further research in this direction – exploring tools and environments that support experimentation, reflection, and iteration – may lead to broader adoption and deeper engagement with formal modeling practices.

Acknowledgements

A part of this research was supported by JSPS KAKENHI Grant Number JP 23K01632 and 24K09052). The author thanks anonymous reviewers for their valuable feedback.

References

1. Chacon, S.: Git. In: Brown, A., Wilson, G. (eds.) The Architecture of Open Source Applications, Volume 1. aosabook.org (2012), `https://aosabook.org/en/git.html`
2. Larsen, P.G., Lausdahl, K., Battle, N., Fitzgerald, J., Wolff, S., Sahara, S., Verhoef, M., Tran-Jørgensen, P.W.V., Oda, T.: VDM-10 Language Manual. Tech. Rep. TR-001 (2013)
3. Oda, T., Araki, K., Larsen, P.G.: A formal modeling tool for exploratory modeling in software development. IEICE Transactions on Information and Systems 100(6), 1210–1217 (2017)
4. Sterman, S., Nicholas, M.J., Paulos, E.: Towards creative version control. Proceedings of the ACM on Human-Computer Interaction 6(CSCW2), 1–25 (2022)