

試行錯誤を支援するための履歴管理フレームワークの試作

小田 朋宏
株式会社 SRA

tomohiro@sra.co.jp

中原 孝信
専修大学

nakapara@isc.senshu-u.ac.jp

Gaël Dur
静岡大学

dur.gael@shizuoka.ac.jp

要旨

現在、ソフトウェア開発やデータ分析をはじめ、多種多様な創造的作業を支援するためのアプリケーションソフトウェアが開発され利用されている。創造的作業の特徴として、多くの試行錯誤を含む探索的な作業であることが挙げられる。すなわち、創造的であるが故に、目指すべき具体的成果は明確ではなく、作業が完了して初めてどのような成果を目指した作業であったのが判明することがしばしばある。探索的な作業であることが創造的作業の困難の一つであるが、現状の創造的な作業のためのアプリケーションソフトウェアは作業対象への操作を提供する一方で、創造的作業における探索を十分に支援しているとは言えない。本論文では、探索的な作業における作業履歴に着目して、探索を支援するための機能の設計指針を提示して、その実装としてのアプリケーションフレームワークを紹介し、3つの作業環境に適用して評価する。

1. 背景

現在、多くの創造的作業環境が実用されているが、その多くは創作対象に対する対話的操作を効率よく実行するように、機能およびユーザインターフェイスが設計されている。本論文では、創作対象に対する対話的な操作によって創造目的に適合した記述を創作する環境を、創造的作業環境と呼ぶ。そして、創造の目的に適した記述を生み出す仕組みについて提案し議論する。

創造的作業の特徴として、多くの試行錯誤を含む探索的な作業であることが挙げられる [1]。創造的作業でどのような成果を得るのか、創造的であるが故に、目指すべき具体的成果は明確ではない。作業が完了して初めて、どのような成果を目指していたのかが明らかになることも多い。多くの創造的作業では、作業開始時点で成果物が明確に決まっていない。そのため、目指す成果を試行錯誤する作業と、実際に成果物を作成する作業が並行して進む。本論文では、目指す成果を試行錯誤するプロセスを『探索作業』と定義する。

現在の作業環境の多くは、創作対象を対話的に操作する機能は充実しているが、作業者が試行錯誤しながら成果を模索する『探索作業』の支援が不十分である。例えば、作業のある段階で進むべき方向が複数考えられる場合、それぞれを試し、比較・検討することが求められる。作業者は探索作業を俯瞰して、自ら保存した多数の作業状態を管理して、探索を遂行する。この種の探索作業の俯瞰と遂行は多くの創造的作業に共通するが、作業環境の多くはこれら探索作業の俯瞰と遂行を直接支援する機能を提供していない。本研究では、作業履歴に着目して、探索作業に適合した柔軟なバージョン管理を通して探索作業への支援を行うことを目指している。

本論文では、第2節で関連する先行研究を紹介し、それを受けて第3節で探索作業を支援するための履歴管理の設計指針を示す。第4節で本研究で試作した履歴管理フレームワーク Hi-De-Ho の具体的な設計と実装を説明し、第5節で3つの異なる作業環境への適用を紹介し、第6節でその結果を評価し議論する。最後に第7節にまとめと今後の展望を示す。

2. 関連研究

Sterman らは創造的作業におけるバージョン管理の役割として、素材のパレット、探索リスクに対するセーフティネット、迅速で柔軟なスナップショット作成、長期的振り返り、の4つを示した [2]。特に、創造的作業の特徴である繰り返し作業と非線形性に着目し、現在のバージョン管理システムの問題点を指定した。それは、プログラミング作業への支援に偏っていること、創造的な作業の進行に必ずしも適合していないこと、ユーザによる複雑かつ明示的なコマンド操作が必要なことを挙げている。これらの知見は、次節で説明する Hi-De-Ho の設計指針を確立する上で、重要な前提となっている。

VDMPad は、ウェブブラウザ上で動作する簡易仕様記述環境であり、EpiLog[4] と呼ばれる履歴管理機能を提供している。VDMPad は探索的仕様記述支援環境 ViennaTalk の一部として開発され、主に教育などに利用されている [5]。EpiLog は、VDMPad 上での仕様の編集や表現式の評価実行を自動的に記録して、後に仕様記述者自身がその過程を整理したり、個別の評価実行記録をテストケースに変換して、その後のバージョンに対する回帰テストとして適用するための機能を提供している。本研究は、EpiLog で得られた知見を基に一般化して、履歴管理の設計および実装をフレームワークとして提供する。

3. 探索を支援するための履歴の利用

本研究は創造的作業での探索を直接的に支援する作業環境の実現を目指すものである。本節では、創造的作業のための履歴管理について、既存のバージョン管理ツールに欠ける部分を指摘し、あるべき機能の基本的な指針を示す。

3.1. スナップショットの自動保存

創造的作業を支援する環境には、Sterman らが指摘した探索リスクへのセーフティネットとしての役割から、過去の作業対象の状態が確実に記録され復元できることが求められる。本論文では以後、記録した作業対象をスナップショットと呼ぶ。

Git 等の既存のバージョン管理システムの多くは、ユーザによるコマンドによる明示的なコマンドによって、スナップショットを保存する。しかし、創造的作業では、

ある時点でのスナップショットが後で必要になるかどうかを予測することは、しばしば困難である。目指すべき具体的な成果は明確ではないために、一定の作業を継続した後で目指すべき方向から逸脱しているのが判明することがある。すなわち、探索の分岐点を過ぎた後で分岐点の存在に気付くことがあり、探索を支援するためには探索をしている当事者が把握している分岐点だけでは不十分である。したがって、スナップショットの保存は、ユーザによる明示的操作だけでなく、作業環境が自動的に実行する必要がある。

3.2. スナップショットのメタデータ

前述のように自動的にスナップショットを保存すると、大量のスナップショットが保存される。それらのスナップショットから、復元すべきスナップショットを発見して復元するためには、個々のスナップショットを説明するためのデータ、すなわちメタデータが必要である。

Git 等の既存のバージョン管理システムでは、バージョン管理機能が1つの独立したソフトウェアツールとして実現されている。そのため、スナップショットにどのようなメタデータが保存されるかは、保存するファイルの特性と関係なく、バージョン管理ツールごとに定義されている。具体的には、タイムスタンプやコメントや元バージョンを示すメタデータがバージョン管理システムごとに定義され、スナップショットごとに保存される。

タイムスタンプやコメントは創造的作業一般で有用と考えられるが、元バージョンへの参照は一般に創造的作業では必ずしも有用ではない。プログラミングは元のソースコードへの修正という形で作業を進めるが、創造的作業一般には元の作業状態への修正という形に限定されない。例えば、データ分析におけるデータベースへのクエリは、記述がプログラムソースと比較して非常に短く、基本的に個々のクエリは独立して扱われる。あるクエリとその直前のクエリとの差分には必ずしも意味はなく、複数のクエリ群全体で分析対象に対する知見を蓄積する形で分析作業が進行する。一方で、そのクエリによってデータベースから取得されたデータの概要を示すデータは、作業者にとってスナップショットを選択する上で重要な意味を持つ。したがって、メタデータは検索性を確保するために一定の共通性を持ちつつ、具体的な創造的作業環境ごとに特有の拡張を定義可能であることが望ましい。

3.3. API の共通化と実装の特化

創造的作業に求められる履歴の役割から、履歴管理システムに求められる機能項目とそれを利用するための API はある程度共通化することができる。しかし、具体的な作業環境ごとに作業対象が異なることから、具体的にスナップショットとして保存する対象が異なるとともに、作業フローが異なることから、ユーザインターフェイスが提供すべき対話性も異なる。

既存のバージョン管理システムの多くは、ファイルシステム上の特定のディレクトリ以下のファイルの更新状態を監視して、ファイル群をスナップショットとして保存する。作業環境がファイルシステム上に書き出した状態のみがスナップショットとして保存可能であり、かつ、作業環境がファイルシステム上にファイルとして書き出すタイミングよりも細かな時間粒度で作業状態を保存することはできない。バージョン管理システムが保存可能なスナップショットや自動保存の時間粒度は、作業環境に依存している。

本研究では、共通化された機能項目に対して異なる実装が必要になる点、および、ファイルのみを対象にしたバージョン管理の限界を克服するために、履歴管理をフレームワークとして提供することにした。フレームワークが履歴管理のインターフェイスを共通化した上で、具体的な作業環境ごとに個別の実装によって作業環境のスナップショットを保存するための実装を行うことにした。

4. 履歴管理フレームワーク Hi-De-Ho の設計と実装

本研究では、異なる応用ドメインで共通して使えるフレームワークとして Hi-De-Ho (Hypothetical Initiation and Dynamic Exploration using History of Operations) を試作した。本節では、Hi-De-Ho を試作する上で掲げた設計指針を説明する。

Hi-De-Ho の実装は、オブジェクト指向言語 Smalltalk の現代的な実装である Pharo 上で行われた。Pharo は統合開発環境と実行環境が統一された形で提供されており、アプリケーションソフトウェアとしての作業環境のベース環境に適している。また、Pharo はスロットやトレイトなどの、先進的な言語機能を搭載している。

スロットは変数モデルをユーザ定義可能にするための言語機能であり、変数への read および write アクセスに

対してコンパイラが生成するバイトコードをユーザプログラムが定義することができる。コンパイラが出力するバイトコードを定義することは、従来のプログラミング言語であればコンパイラに変更を加えることで実現する必要があるメタな定義だが、スロットを利用することで通常のプログラミングと同様の容易さで行うことができる。read および write アクセスをユーザプログラムが定義する手段としては、従来のオブジェクト指向言語ではアクセサメソッドを定義することが広く実践されているが、スロットによって変数モデルを定義することで確実かつ簡潔にユーザ定義による変数アクセスを適用することができる。

トレイトは既存のオブジェクト指向言語ではインターフェイス型に近い機能を提供する。インターフェイス型同様に、どのメソッドを実装する必要があるか、また、そのデフォルト実装を変数宣言とともに定義することができる。トレイトの特徴として、クラス階層とは独立して、共通機能および変数の宣言および実装を定義することができることが挙げられる。既存のクラスに対して必要な追加機能を後付けする場合に、特に有効性を発揮する言語機能である。スロットおよびトレイトは Hi-De-Ho の実装にも利用されており、詳細を後述する。

4.1. メタデータ

Hi-De-Ho では、メタデータを定義するクラスとして HiDeMetadata クラスを定義した。HiDeMetadata クラスは、必須のメタデータとして ID、タイムスタンプ、およびタグを格納し、それらを JSON 形式でファイルシステム上に書き出しおよび読み込みを行う。具体的な作業環境に応じて、HiDeMetadata クラスのサブクラスを定義することで、その他のメタデータを保持することができる。

4.2. ストレージ

履歴管理システムの中心になるコンポーネントは、スナップショットを保管・管理するストレージである。Hi-De-Ho では、ファイルシステム上にスナップショットを保管し、ユーザがメタ情報を編集し、スナップショットを分類し検索するための機能を提供するクラスの基底として HiDeStorage クラスを定義した。HiDeStorage クラスは、個々のスナップショットについてユニークな

ID を割り振り、また、ファイルシステム上のどのファイルパスに割り当てるのかを管理する。

HiDeStorage クラスはスナップショットをクロニクル（年代記）およびナラティブ（物語）と呼ばれる時系列に保管する。クロニクルは作業環境上で行われた作業全体を収め、ナラティブは特定の目的のために行なった一連の作業のみを切り出して保存することを想定している。また、HiDeStorage クラスは、クロニクルに対する検索機能を提供する。メタデータとして格納されたタイムスタンプやタグやコメントなどを対象に検索を行うとともに、ナラティブ機能を使ってユーザが独自の分類をすることで、ユーザは復元すべき過去のスナップショットを探ることができる。

HiDeStorage クラスは、スナップショットの復元を、メタデータのみを読み込みとスナップショット本体の生成の2段階に分けて行う。まずはメタデータを読み込み、そのメタデータの内容に応じて、具体的な作業状態をあらわすスナップショットオブジェクトを選択して、そのインスタンスを生成する。スナップショットオブジェクトの選択は `payloadClassFor`: メソッドによって行うが、具体的な作業環境によって具体的な作業状態を表すオブジェクトが異なるため、HiDeStorage クラスでは未定義のままである。HiDeStorage クラスは抽象クラスであり、`payloadClassFor`: メソッドを定義するサブクラスを具象クラスとして定義することで、その機能を利用することができる。

4.3. スナップショット

スナップショットは、作業環境上の作業対象をファイルシステム上に永続化させ、また、復元させるためのオブジェクトである。スナップショットの対象は作業環境ごとに異なるため、Hi-De-Ho ではスナップショットを具象クラスではなくトレイトとして定義した。

HiDePersistency トレイトの最も重要な機能は、`hideSave`: および `hideLoad`: メソッドである。これらのメソッドはそれぞれ、引数で渡されたファイルパスにスナップショット対象の書き出しおよび読み込みを行う。

作業環境によっては、保存すべき作業対象は単一オブジェクトではない場合がある。例えばデータ分析におけるクエリはその結果取得されたデータと共に保存することがユーザの利便性につながる場合がある。そのような複合的なスナップショット内の作業対象を一斉に復元す

ることはパフォーマンス上または処理効率上望ましくない。そのため、Hi-De-Ho では、個別の作業対象ごとに必要に応じてファイルシステムからの復元を行うための仕組みとして HiDePersistencySlot スロットを実装した。Pharo コンパイラは、HiDePersistencySlot スロットとして宣言された変数への `read` アクセスに対して、変数が保持する内容がファイルシステムから復元されているか検査して、復元されていない場合のみ復元処理を実行するためのバイトコードを出力する。これによって、クライアントコードはあたかも通常の変数への参照であるかのように、ファイルシステムからの復元処理を遅延したアクセスを行うことができる。

また、HiDePersistency トレイトにはメタデータおよびストレージオブジェクトへの参照を保持するための変数を定義した。Hi-De-Ho は作業環境に後付けすることを念頭に置いたフレームワークであり、Hi-De-Ho を組み込むために作業環境側のプログラムにも修正が必要である。Hi-De-Ho を組み込むための作業を容易にするために、作業環境側が Hi-De-Ho の実装への依存が抑制されることが望ましい。そのため、作業環境がメタデータやストレージオブジェクトを意識することなく、スナップショットオブジェクトのみを使ってスナップショットの保存および復元を行うことができよう、メタデータおよびストレージオブジェクトへの参照を保持して、処理がスナップショットオブジェクトに閉じるように実装した。

既存のバージョン管理システムの多くがテキストの差分を取得する機能を提供しているように、創造的作業を支援する環境でもスナップショットと他のスナップショットとの差分が有用なことが多い。また、作業対象にはパーサに生成される AST (Abstract Syntax Tree, 抽象構文木) をはじめとする木構造として表現されるものがある。そこで Hi-De-Ho では木構造の差分を取得する機能を HiDeSyntaxTree トレイトとして提供し、汎用の差分取得可能木構造として HiDeSyntaxDictionary クラスを実装した。

5. 作業環境への適用

本研究では、履歴管理フレームワーク Hi-De-Ho を ViennaTalk, gOLAP, re:mobidyc の3つの作業環境に組み込んだ。Pharo 上に実装されており、試行錯誤を伴う創造的作業を支援するためのソフトウェア環境である。い

表 1. Hi-De-Ho を適用した作業環境の概要

	ViennaTalk	gOLAP	re:mobidyc
対象ドメイン	ソフトウェア開発	データ分析	シミュレーション
想定ユーザ	ソフトウェア開発者	マーケティング分析者	生物学研究者
スナップショット対象	ソースコード群	データベースクエリ	モデル定義ファイル群
対象サイズ	数千行	数行	数百行
記述方法	テキスト編集	GUI またはテキストの書き下し	GUI での修正

ずれの作業環境も、Hi-De-Ho 開発開始時点よりも前からオープンソースとして継続的に開発されてきた。Hi-De-Ho を組み込んだ3つの作業環境は、それぞれ対象ドメイン、ユーザに想定している知識や技能、履歴として保存される個別データのサイズや作業期間が異なる。それぞれの作業環境の特徴を表1にまとめ、それぞれにおける履歴管理の概要を説明する。

ViennaTalk は、ソフトウェア開発における仕様定義の初期段階に焦点を当てた形式仕様記述環境であり、プログラミング環境と同様にテキスト形式のソースを編集対象にしている。ユーザはソフトウェア開発に精通していることを想定している。仕様記述の試行錯誤では、エディタ上での編集によって生じた差分やその影響範囲をユーザが把握することが重要であることから、Hi-De-Ho を組み込む以前からソースに対する短期的な履歴管理が実装されていた。しかし、Hi-De-Ho 組み込み以前は、履歴はエディタを開いている間のみ保管され、エディタ終了後は残らなかった。Hi-De-Ho を組み込むことで、履歴をファイルシステム上に永続化して、数日から数週間程度の履歴を保存するように拡張した。

gOLAP はサーバ・クライアント形式のマーケティング分析ツールであり、サーバ側ではデータの集計処理を行い、クライアント側ではサーバに送信するクエリの編集と、サーバが集計した結果をグラフ構造にしてインタラクティブな可視化を行う。本研究ではクライアントからサーバに送信するクエリを履歴管理の対象にした。クエリはいくつかの条件節の組み合わせであり、JSON形式でサーバに送信される。各クエリの記述は全体で10行前後であり数分程度で記述するため、ソフトウェア開

発での履歴とは異なる利用法や役割が想定される。

re:mobidyc は生物学・環境学向けのマルチエージェントモデリング環境であり、ユーザはテキストエディタではなく GUI を中心にしてモデルを構築する。構築されたデータは複数の定義ファイルにそれぞれテキスト形式で保存される。本研究では、それらの定義ファイルを履歴管理の対象にした。個々の履歴データは ViennaTalk の仕様記述と同様のサイズと作業期間が想定されているが、利用者がバージョン管理に習熟していることを想定している ViennaTalk とは異なり、re:mobidyc ではユーザはバージョン管理に関する知識や技能を前提にはしていない。以下、それぞれの作業環境への Hi-De-Ho の適用を説明する。

5.1. ViennaTalk への適用

ViennaTalk は仕様記述の初期段階での試行錯誤を支援するための仕様記述環境として開発された。形式仕様記述言語 VDM-SL による仕様記述に特化している。Hi-De-Ho による履歴管理を実現するにあたって、単にスナップショットを保存した時点の仕様を復元するだけでなく、スナップショット間の差分の取得を特に重視した。図1にスナップショットのリストと差分を表示している画面例を示す。画面の下半分に Hi-De-Ho タブが選択されている。画面左下のリストにスナップショットのタイムスタンプ、編集の種類、コメントが時系列で表示されており、選択されたスナップショットの差分が画面右下に表示されている。本節では、ViennaTalk での Hi-De-Ho の適用の特徴として、スナップショットのリストを作成するための時系列とフィルタについて説明する。

仕様記述の初期段階では、システムに関係する概念の名称が揺れたり、概念の粒度が変わったり、あるいは、概念を仕様記述言語のどのような言語要素として定義するかが変わることがあり、そのために仕様の表現が大きく変化することがある。ViennaTalk ではそのような仕様の修正を効率的に行うための支援として、識別子を一齐に変更する rename 操作や、繰り返し出現する式に識別子をつけて定義する extract 操作など、豊富な自動リファクタリング機能を提供している。

これらの強力な自動リファクタリング機能は、広範囲にわたる編集を漏れなく実行できる点で有効だが、ユーザにとって編集範囲を把握し難いことがある。自動リファクタリング機能を実行する操作によって変更されるソー

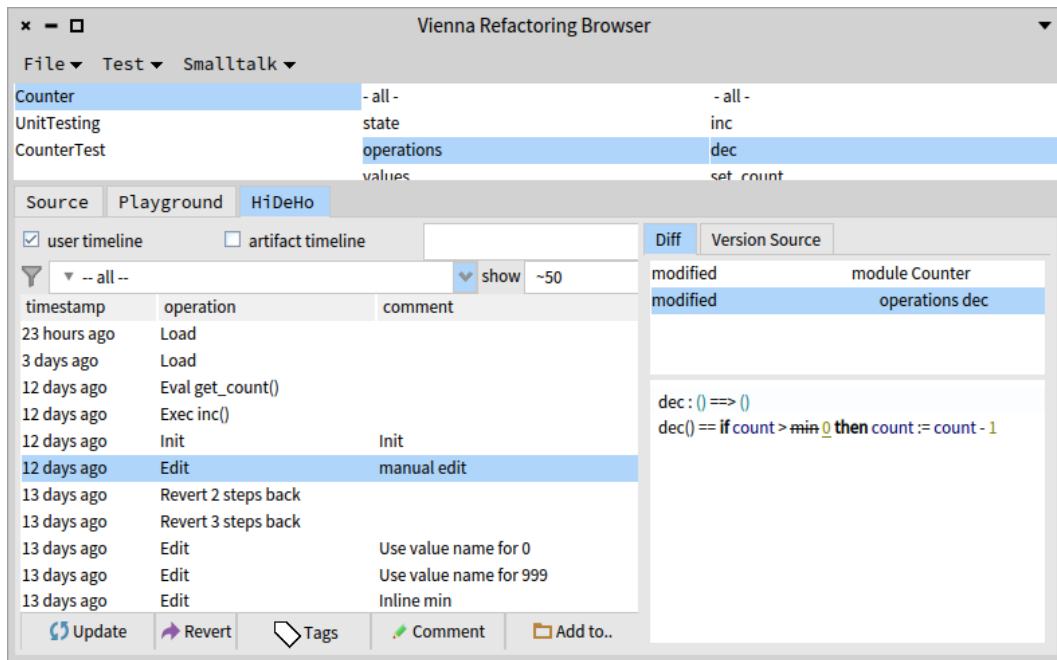


図 1. ViennaTalk 上の履歴管理 UI の画面例

スコード上の箇所について、ユーザが把握していない箇所があった場合には、ユーザが認知する仕様の定義と実際のソースコード上の定義に差が生じることになり、誤り混入につながる危険がある。そこで、Hi-De-Ho で保存したスナップショットを使って、自動リファクタリングによって生じた差分をユーザに見せることで、ユーザが仕様の修正状況を俯瞰する支援を行うことにした。スナップショットおよびその差分を見せることで仕様の修正状況を俯瞰する支援として、編集作業での自動スナップショット作成と、VDM-SL の言語仕様に基づいた影響範囲の分析による差分表示を実装した。

スナップショットの自動作成は、自動リファクタリング機能を実行するごとに、また、ユーザによるテキスト編集での仕様への修正を行うごとに、Hi-De-Ho によって仕様のスナップショットを自動的に保存するようにした。これは、ViennaTalk が自動リファクタリング機能またはユーザによる編集を受け付けるメソッドに、hideSave メソッド呼び出しを追加することで簡単に組み込むことができた。

第 4.2 節でストレージが管理するスナップショットの時系列として、履歴全体の時系列としてクロニクルと特定の目的を持った作業をあらわす時系列としてナラティブの 2 種類があることを説明した。ViennaTalk への適用

では、クロニクルから一部を切り出した時系列として、ナラティブ以外に 2 つの動的な時系列を定義し、さらに、時系列に対するフィルタを提供した。ViennaTalk の Hi-De-Ho 用インターフェイスでは、これら 2 種類のタイムラインとナラティブをチェックボックスで選択する。

ViennaTalk は 1 つのシステムの仕様を 1 つの作業フォルダ内にモジュールごとのソースファイルに分割して保存する。作業フォルダごとに定義される 2 つのタイムラインの概要を図 2 に示す。図 2 は、ユーザが仕様をテキスト入力で修正（編集 1）した後、リファクタリングをしようとした時に 2 種類のリファクタリングのどちらを実行するか迷い、まずはリファクタリング 1 を実行し、一旦元に戻した（編集 1 を復元）後でリファクタリング 2 を実行し、比較した結果リファクタリング 1 のほうが適切だと判断して、リファクタリング 1 を実行した状態を復元し、テキスト入力で修正（編集 2）を継続した状況を示している。

1 つ目のタイムラインは、最新スナップショットのチェックアウトや過去のスナップショットの復元を含むユーザ・タイムラインである。図 2 では、初期状態から編集 2 までの黒い矢印を辿った全てのスナップショットが、ユーザタイムライン上で整列される。ユーザ・タイムラインは、ユーザが試行錯誤をしてきた過程であり、その作業

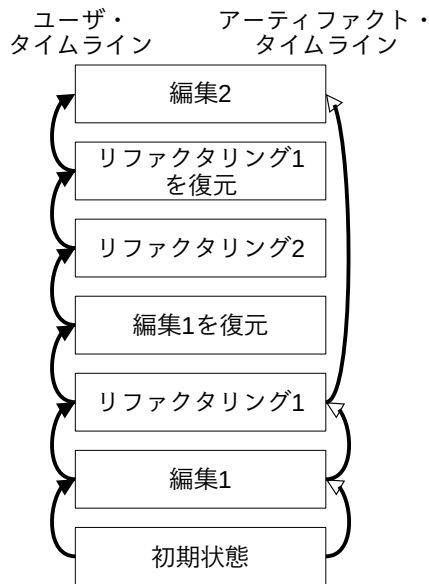


図 2. ViennaTalk 上のタイムラインの例

フォルダ中での ViennaTalk 上での操作履歴に相当する。もう 1 つのタイムラインは、復元するスナップショット以後のスナップショットをスキップするアーティファクト・タイムラインである。あるスナップショットから編集が行われても、編集前のスナップショットが復元されたら、復元と編集はユーザ・タイムラインに含まれない。図 2 での編集 2 は、リファクタリング 1 を復元したものに対する編集であるため、アーティファクト・タイムラインではリファクタリング 1 の次の操作として編集 2 が続く。アーティファクト・タイムラインは、現在の仕様定義が構成された過程を追跡した時系列であり、既存のバージョン管理システムのブランチに相当する。ユーザは、試行錯誤の過程を追跡している時にはユーザ・タイムライン、現在の仕様定義の変遷を追跡している時にはアーティファクト・タイムライン、あるいはユーザがスナップショットを選択して作成したナラティブを選択して、スナップショット・リストから適切なスナップショットを発見することができる。

5.2. gOLAP への適用

gOLAP はマーケティングのための共起分析を中心にしたデータ分析ツールである。gOLAP はサーバ・クライアント構成を取っていて、サーバは大量のレシートデータを蓄積して、そこから共起関係を抽出して、一緒に購

```
{
  "query" : {
    "selCond" : {
      "minSup" : 0.0,
      "minConf" : 0.0,
      "minLift" : 0.0,
      "minJac" : 0.0,
      "minPMI" : 0.0},
    "sortKey" : "sup",
    "sendMax" : 1000,
    "isolatedNodes" : "true",
    "traFilter" : "isin(支払方法,'現金')"},
  "deadlineTimer" : 600}
}
```

図 3. gOLAP クエリの例

入されることが多い商品ペアやその頻度を集計する。クライアントはサーバに送信するクエリを構成し、サーバから返信された大量の集計データをグラフ構造を使ってインタラクティブな可視化を行う。本研究では、クエリを構成する作業に着目して、gOLAP クライアントのクエリ編集 UI に Hi-De-Ho フレームワークを組み込みクエリ履歴を蓄積して、ユーザの探索的な分析を支援する。

gOLAP サーバへのクエリは JSON フォーマットのテキストとして構成される。図 3 にクエリの例を挙げる。gOLAP によるデータ分析作業は、クエリ単体でデータ分析が完結することは多くない。典型的には、分析したい特徴が現れるように、さまざまな条件節の組み合わせを試みて、それぞれの集計結果についてグラフ構造の可視化を使って特徴的なパターンを探索する。マーケティングで分析する特性には長期的な傾向に関するものもあり、同種のクエリで長期的に追跡するケースもありうる。Hi-De-Ho フレームワークによって、分析目的に応じて分類されたクエリを蓄積して、過去のクエリからそれぞれ一部を取り出して組み合わせるクエリを構成したり、1 セットのクエリ群で条件節の必要な組み合わせが網羅されたかを確認する支援を提供することを開発の中心に設定した。ただし、クエリはサーバごとに格納されているデータや集計項目が異なるため、スナップショットのメタデータにホスト情報を追加して、現在クエリを作成している対象のサーバのクエリのみをクロニクルおよびナラティブとして取得できるようにした。

図 4 に履歴を使ってクエリを作成するための GUI のスクリーンダンプを示す。最上段にサーバ URL を入力することで、そのサーバに対応した Hi-De-Ho のストレ



図 4. gOLAP 上の履歴を使ったクエリ作成 UI の画面例

ジを利用することができる。サーバ URL 以下の画面は、大きく横方向に 3 つの領域に分けられ、履歴を表示する左半分、クエリを編集する中央、サーバのパラメータを表示する右端から構成されている。ユーザは現在行っている分析作業に関わるクエリを格納したナラティブ（画面上ではフィールドとして示されている）を選択する。gOLAP はナラティブ内の全クエリに共通する部分を抽出し、左上の**共通部分**と示されたテキスト領域に表示する。共通部分は現在行っている分析作業に共通した構成要素であるから、現在作成中のクエリにもそのまま流用できる可能性が高い。共通部分の下に配置された**クエリに上書き**ボタンを押すことで、現在作成中のクエリにそのまま展開される。

一方、ナラティブ中のクエリで共通していない部分、すなわち各クエリと共通部分の差分も、クエリを構成する上で重要な役割を持つ。各クエリに含まれる非共通部分は、図 4 の画面左側下段に**変化部分**として表示される。変化部分はテーブル形式になっていて、各行がクエリに対応し、最左カラムにタイムスタンプ、以降のカラムには変化部分になっている条件節が割り当てられる。図 4 の画面例では、query-trafilter-日付すなわち売り上げられた日付に関する条件と、query-trafilter-支払い方法すなわち支払い方法に関する条件が、クエリごとに異なっている。ユーザはこの表から、どの条件節がどのようなパラメータでクエリに入れられたのか、その組み合わせが網羅的になっているのかを確認すること

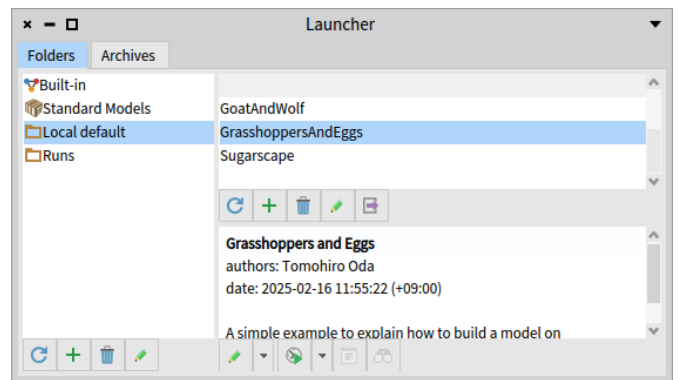


図 5. Launcher の画面例

ができる。また、このテーブルからクエリとカラムを選択して、現在編集中のクエリに追加挿入することができる。

5.3. re:mobidyc への適用

re:mobidyc は生物学研究者向けのマルチエージェントモデリング環境である。想定ユーザは生物学研究者であり、できるかぎりプログラミングスキルを要求しないことを設計指針の一つに掲げて、エージェントモデルやユーザーインターフェイスが設計および実装された。

モデルの一覧やモデルの編集、実行、実行結果の管理は、図 5 に示した Launcher と呼ばれる GUI 上で行う。

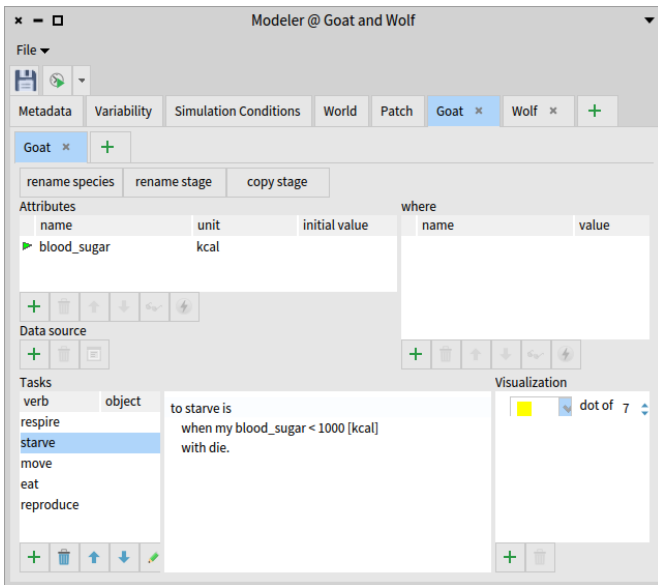


図 6. re:mobidyc 上のモデリング画面例

re:mobidyc ベースディレクトリ下にモデルごとにディレクトリを作成して、ディレクトリ内にモデル定義や時系列データなどの関連ファイルを保存する。ユーザはディレクトリ内の具体的ファイルを直接操作する必要はなく、個別のモデルの操作は Modeler と呼ばれる GUI 上で行う。

Modeler の画面例を図 6 に示す。Modeler 上では各エージェント定義がタブページ上で行われる。タブページ上では、エージェント定義として Attribute と呼ばれる変数の定義や Task と呼ばれる動作定義がリスト上に列挙される。Attribute や Task の追加はボタンなどの GUI を通して対話的に行われる。モデルの保存時には複数のエージェント定義がまとめられて、Attribute 定義ファイル、Task 定義ファイルなど定義の種類ごとに対応するテキストファイルに保存されるが、通常の使用ではユーザがこれらの定義ファイルを直接テキスト編集することはない。

ユーザの利用シーンでは、モデルを格納するディレクトリ名の末尾に日付をつけて管理している様子が観察された。Modeler でモデル定義を修正して保存するとそのモデルを格納するディレクトリ内のファイルが上書きされるため、意図せず上書きして元のモデル定義を失うリスクを回避するための手段と考えられる。

Hi-De-Ho を適用するにあたっては、モデルをディレクトリに保存する時点で自動的にスナップショットを保存

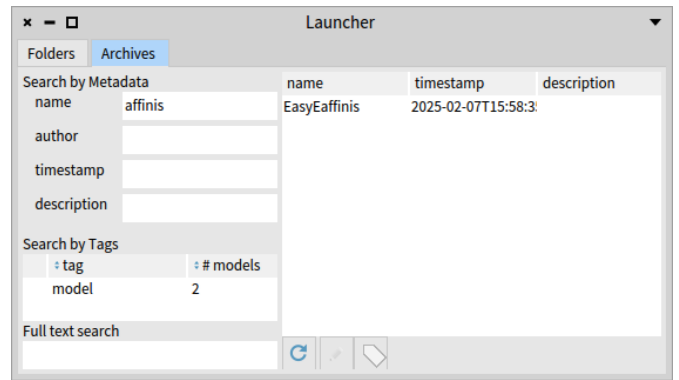


図 7. re:mobidyc 上の履歴管理 UI の画面例

して、ユーザによる明示的な操作は保存されたスナップショットの選択と復元とタグ編集に限定することにした。スナップショットの自動保存と復元は、ユーザがモデル定義を試行錯誤的に変更していく上での安心を提供することができる。ViennaTalk のような高度なスナップショット管理や、gOLAP のような複数スナップショットを絵の具のパレットのように使った合成の導入は、re:mobidyc のユーザがスナップショットの扱いに習熟してから改めて検討することにした。本論文では、初期導入としての、ユーザによるスナップショットの復元操作について説明する。

Launcher にスナップショット管理のユーザインターフェイスを組み込んだ画面の例を図 7 に示す。Launcher には Folders タブページと Archives タブページがあり、Folders タブページには従来のモデルディレクトリに対するユーザインターフェイス、Archives タブページにスナップショットに対するユーザインターフェイスが提供される。Archives タブページは大きく左右 2 つの部分から成っている。左側にはメタデータやタグによる検索およびモデル定義内の文字列検索を行うユーザインターフェイスが提供され、右側には検索にマッチしたスナップショットのリストが表示され、選択したスナップショットに対してタグを編集したりスナップショットを復元することができる。スナップショットを復元すると Modeler が開き、Modeler でモデル定義を確認した上で、新規または既存のモデルディレクトリに保存することができる。ユーザは、バージョン管理システムに関する知識やスキルを要求されることなく、自動的に保存されたスナップショットをバックアップとして利用することができる。

6. 評価と議論

本研究では探索作業を支援するための履歴管理フレームワーク Hi-De-Ho を仕様記述環境 ViennaTalk, データ分析ツール gOLAP, マルチエージェントモデリング環境 re:mobidyc の3つの作業環境に適用した。3つの作業環境は作業対象のファイルサイズ, 1つの作業対象への作業期間, 作業者の専門性, バージョン管理への習熟度などが大きく異なる。

Hi-De-Ho の設計指針として, スナップショットの自動保存, 共通項目に加えて拡張可能なメタデータ, API の共通化と実装の特化を挙げた。Hi-De-Ho の実装では, 異なる特性を持つ3つの作業環境に対して, 既存の実装に大きな変更をすることなく必要な機能を追加することができた。

3つの作業環境への適用で判明した困難としては, ユーザーインターフェイスの実装コストが挙げられる。Hi-De-Ho フレームワークはタグエディタなどの共通 GUI 部品を提供するが, 具体的なユーザーインターフェイスの構成は適用対象となる作業環境によって異なる。ユーザーインターフェイス上で使う用語は, 作業環境ごとの適用ドメインに適合した言葉を選択することが望ましい。例えば, HI-De-Ho のユーザー定義時系列であるナラティブは, ViennaTalk のユーザーインターフェイス上ではタイムライン, gOLAP のユーザーインターフェイス上ではフィールドと呼ばれ, re:mobidyc ではナラティブは利用されていない。また, 作業環境によって想定するユーザの知識やスキル, 必要なユースケースが異なり, 作業環境ごとにほとんどスクラッチからの実装になった。ユーザーインターフェイスの追加実装は作業環境の既存のユーザーインターフェイスの実装にも変更が必要になる場合があり, Hi-De-Ho フレームワークを適用する上での追加コストになり得る。

別の問題点として, ナラティブの構成やタグ付けなどの, スナップショットのメタデータ編集に関するユーザの負荷が挙げられる。ユーザの作業目的はあくまで創作物であり, 探索作業のための作業負荷や認知負荷は最小限に留めることが望ましい。

7. まとめと今後の展望

創造的な作業環境は, ソフトウェア開発にとって古くからあるアプリケーション領域である。本研究では, 創造的な作業に行われる探索作業を支援するための作業環

境を目指して, 履歴管理フレームワーク Hi-De-Ho を設計および実装し, 3つの異なる応用領域の創造的な作業環境に適用した。問題点としてユーザーインターフェイスの実装と, 履歴管理のためのユーザの負荷が挙げられた。履歴管理のためのユーザの負荷については, ベースとなる作業環境上での操作に連動する形でのメタデータの自動生成が今後の課題として挙げられる。例えば, 仕様記述環境 ViennaTalk では構文検査, 型検査, ユニットテストなどの検査が実行されるが, 各検査の結果をメタデータに自動的に付加することが今後の拡張として考えられる。また, イシュー管理システムや既存のバージョン管理システムとの連携など, 作業環境以外のツールとの連携も今後の課題として挙げられる。

謝辞

本研究の一部は JSPS 科研費 (23K01632) および JSPS 科研費 (24K09052) の助成を受けた。gOLAP は (株)NYSOL の丸橋弘明氏、岡山フードサービス (株) の中元政一氏、そして中原輝昭氏と共同開発を行っている。ここに感謝の意を表す。

参考文献

- [1] Donald A. Schön, “The reflective practitioner: how professionals think in action.”, 1983.
- [2] Sarah Sterman, et al. “Towards Creative Version Control.”, Proc. of the ACM on Human-Computer Interaction 6.CSCW2 (2022): 1-25.
- [3] 小田朋宏, Gaël Dur. “個体群動態論に特化したマルチエージェントシミュレーション基盤 Re:Mobidyc”, 第41回ソフトウェアシンポジウム論文集, ソフトウェア技術者協会, pp. 35-44, 2021.
- [4] 小田朋宏, 他. “探索的仕様記述のための履歴ツールの提案と実装”, 第40回ソフトウェアシンポジウム論文集, ソフトウェア技術者協会, pp. 95-103, 2020.
- [5] 小田朋宏, 荒木啓二郎. “形式仕様工程の初期段階に着目した統合仕様記述環境 ViennaTalk”, コンピュータソフトウェア, 日本ソフトウェア科学会, 34 巻, 4 号, pp. 129-143, 2017.