ソフトウェアの黙示的生成と明示的編集のための環境の開発
ー「MINAKA for Java」の開発ー

1． 背景

ソフトウェアの生産性や品質を向上させるためには **DRY**(Don't Repeat Yourself.)原則を実践することが近道である。この原則は「手作業の繰り返し」をしてはいけないことを意味する。この原則は当然のように思えるが、実践するとなると難しい。

ここでは、開発者が日常的に行っているプログラム変換について考える。例えば、デバッグ文やパラメータの追加などの単純なプログラム変換が考えられる。他にもリファクタリングなどの高度なプログラム変換もよく行われている。このようなプログラム変換を行うとき、変換対象が少ない状況では手作業で変換しても間違えないが、多い状況ではそうはいかない。多い状況であっても手作業を根性で繰り返し乗り切ってしまう凄い人がいるが、そういう人は「デバッグ後にデバッグ文を削除して欲しい」、「本当はパラメータを2つ追加して欲しかった」と言われたときに愕然とするだろう。そして、このような状況を「手作業の繰り返し」で乗り切っている限りソフトウェアの生産性や品質は向上しない。

開発者は「手作業の繰り返し」で乗り切るべきではないプログラム変換によく遭遇している。しかし、まともなツールや API が存在しない現状では「手作業の繰り返し」を自動化するために多くの工数がかかる。そのため、多くの開発者は今を乗り切ることを優先し「手作業の繰り返し」に陥る。これがソフトウェア開発における最も深刻な問題である。そして、この問題の根本的な原因は、**DRY** 原則を実践するための現実的手段が存在しないことである。

2． 目的

GUIアプリケーションでは、**DRY** 原則を実践するための現実的手段として操作履歴を記録・再生する機能が使われているが、プログラムの編集では使われていない（テキストの編集では使われている）。この機能は、操作を手軽かつ効率的に自動化できる点で優れている。それだけでなく、この機能は雛型プログラムの自動生成や、雛型プログラムを編集（一般化）してモジュールとして再利用するためにも使うことができるため、プログラムの編集との相性が非常によい。そこで、われわれは操作履歴の記録・再生をプログラムの編集に応用したメタプログラミング環境「**MINAKA for Java**」を開発することで、開発者に **DRY** 原則を実践させソフトウェアの生産性と品質を向上させる。

3． 開発の内容

われわれは「**3．1．枠組**」に従い「**3．2．実行環境**」で使うことができるソフトウェア「**MINAKA for Java**」を開発することで「**目的**」を達成した。

3．1．枠組

図 1「**MINAKA for Java**」**の枠組**が利用者と「**MINAKA for Java**」の関係を示している。

- **操作履歴の記録**
  - ① 操作（プログラムの編集など）という直感的な行為から、操作履歴（その操作を

再現するプログラム）を自動的に生成できる。
② 操作履歴の編集操作を記録することで、どんなメタ的な操作にも対応できる。

- **操作履歴の再現**
  ① 現在や過去の状況を再現できるため安心できる。
  ② 状況を確認しながら対話的にプログラミングできるため分かりやすい。
- **操作履歴の編集**
  ① 過去の操作の誤りを正すことができる。
  ② 一般的な操作（リファクタリングなど）を抽出することで記録したときとは異なる状況へ適用できる。

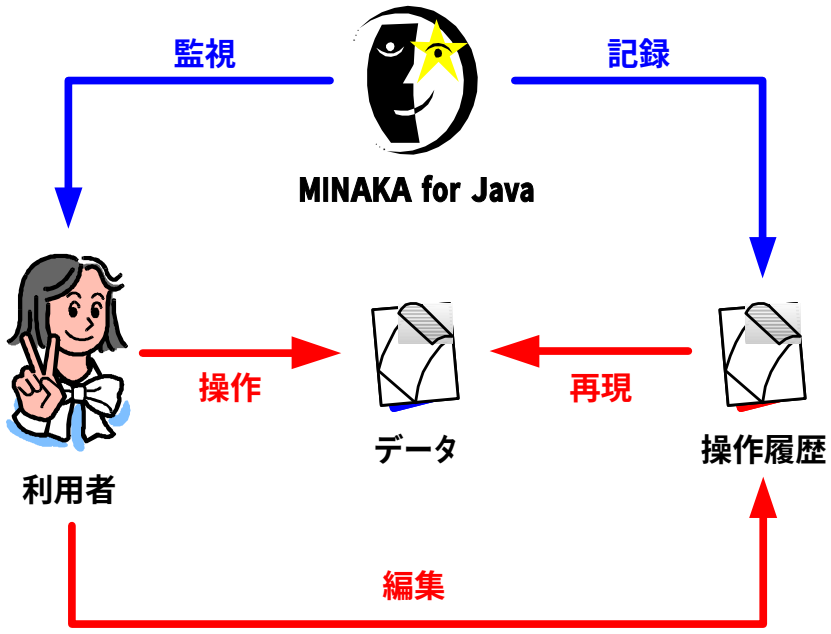

図 1 「MINAKA for Java」の枠組

3．2．実行環境

「MINAKA for Java」は図 2 「MINAKA for Java」の実行環境の 2 つの水色から構成される。

- **MINAKA for Java(Eclipse Plugin)**
  ① 「3．1．枠組」の利用者とのインターフェイスに関する機能を実現するモジュールである。
  ② ソフトウェア開発環境「Eclipse」の上で実行できる。
  ③ このモジュールと「MINAKA for Java(Core)」を組み込めば「Eclipse」から「MINAKA for Java」の機能を使うことができる。
- **MINAKA for Java(Core Module)**
  ① 「3．1．枠組」の概念的な機能を実現するモジュールである。
  ② Java 実行環境「Java Runtime Environment」の上で実行できる。
  ③ このモジュールを組み込めば Java プログラムから「MINAKA for Java」の機能を使うことができる。

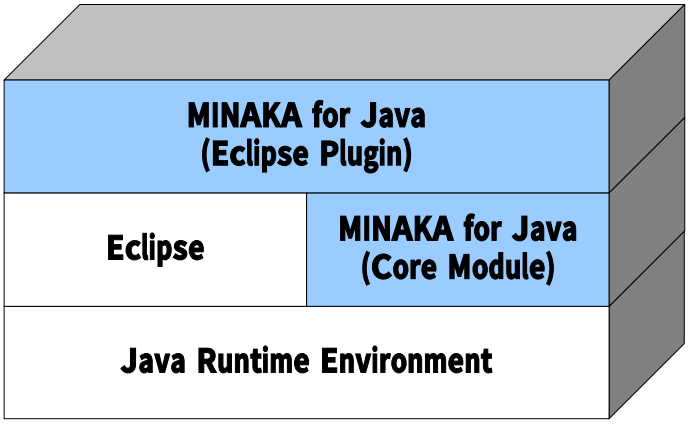

図 2 「MINAKA for Java」の実行環境

図 3 「MINAKA for Java」を「Eclipse」に組み込んだ画面は、操作履歴の記録・再現・編集を使いながら Java プログラムの開発を行っているところである。現在の「**MINAKA for Java**」はそれを使ってそれ自身を開発できるほど実用的である。

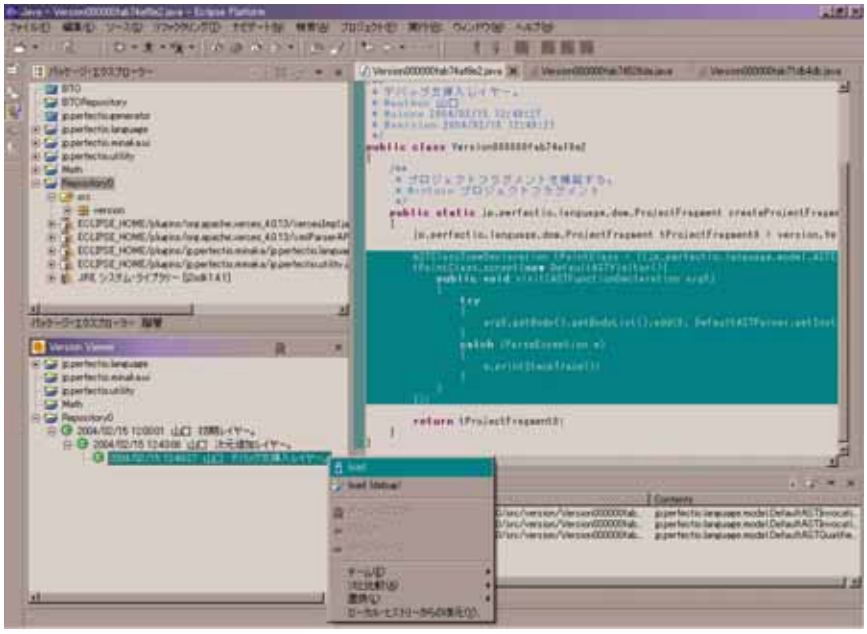

図 3 「MINAKA for Java」を「Eclipse」に組み込んだ画面

4. 従来の技術（または機能）との相違

「**MINAKA for Java**」はソフトウェア開発環境であるが様々なシステムと関係が深い。

- バージョニングシステム（CVS など）
  - ① CVS ではテキスト差分を行単位の差分としてしか得られないが、「**MINAKA for Java**」では高い抽象度の差分がプログラムの形式で得られるため、差分が理解しやすく再利用する価値が高い。
- ビルドシステム（ANT など）
  - ① ANT では記述できる処理が独自言語の表現力に制限されている。それに対し

て「MINAKA for Java」では汎用言語とライブラリを組み合わせることで、汎用性と適度な記述しやすさを実現できる。

- テンプレートシステム（Velocity など）
  ① Velocity はプログラム分析を伴わないファイル生成のみを許すことでメタプログラムの記述しやすさを実現している。それに対して「MINAKA for Java」は高度なプログラム変換を許しているにも関わらず、高度な API によって適度な記述しやすさを実現している。
- アスペクト指向システム（Aspect J）
  ① Aspect J はインターフェイスを変えない程度のプログラム変換と基本的なプログラム検索のみを許すことでメタプログラミングの記述しやすさと安全性を実現している。それに対して「MINAKA for Java」は高度なプログラム変換を許しているにも関わらず、高度な API と対話的な開発方法によって適度な記述しやすさと適度な安全性を実現している。

5． 期待される効果

「MINAKA for Java」を導入することで開発者は DRY 原則を実践できる。これにより、ソフトウェアの生産性や品質が向上する。また、操作履歴を上手く使うことで従来は困難だった高度なプログラムの再利用や分業が可能になる。
意外なことに「MINAKA for Java」は組込系に向いている。例えば、携帯電話用 Java の機種依存の解決や最適化に使える。この他にもさまざまな応用の可能性がある。

6． 普及（または活用）の見通し

これから「MINAKA for Java」を使って開発者向けツールを作り公開することで、「MINAKA for Java」が如何に役に立つかをアピールする。また、われわれが実際に使い成功している開発スタイル（「MINAKA for Java」を作るきっかけになったもの）を Web や雑誌やセミナーなどで紹介する。既に「MINAKA for Java」を使いたいというパートナーが現れているので、そのような人たちと協力して適用事例を増やしたい。最後に、「MINAKA for Java」を使いたい・開発したい、「MINAKA for Java」に投資したいというパートナーを増やし（いつでも募集中）、さらに発展させてゆきたい。

7． 開発者名（所属、e－mailアドレス）

開発者名	所属	e－mailアドレス
山口陽平	名古屋工業大学世木研究室 有限会社来栖川電算	mel@perfectio.design.co.jp
佐藤太亮	有限会社来栖川電算	sato@perfectio.design.co.jp
是安正博	有限会社来栖川電算	mau@cside.com

（参考）開発者ＵＲＬ

http://www.perfectio.jp/project/minaka/