抽象解釈に基づく発展的プロトタイピングのための開発環境に関する研究

番 真悟 (110106)

北陸先端科学技術大学院大学 情報科学研究科

2003 年 2 月 14 日

キーワード: Java，段階的詳細化，抽象実行，プログラミング環境，プロトタイピング.

1 背景

近年, インターネット上で稼働する大規模なネットワークソフトウェアや, 携帯電話のような複雑な制御を必要とするソフトウェアを構築するための高度なソフトウェア開発技術が要求されている.

段階的詳細化は大規模で複雑なソフトウェアを開発するための有効な手法の一つである. この手法では, まず単純で重要な部分からソフトウェアを構築し, そのソフトウェアが完全なものになるまで, 必要に応じてくり返し詳細化を行う. この手法の問題点は, 開発の途中段階でプログラムを動かすことができないことである. そのため, 仕様の誤りを開発の初期段階で発見することが困難である.

発展的プロトタイピング技法は, 抽象値を用いてプログラムを行なうことで, この問題を解決している. この技法では, まず抽象的なデータを用いて原始的なソフトウェアを構築し, そのデータを具体化していくに従って, ソフトウェアの詳細を決定していく. この技法は, システムのある部分が抽象的であったり, 実装されていない場合でも, 抽象解釈を用いてシステム全体の実行を可能としている. このため, 開発の初期段階であっても, 実際にプログラムを実行することで, システムの誤りを見つけることが可能であると考えられている.

しかし, 発展的プロトタイピング技法は提案されてから間もないので, その技法を用いるための実行環境や開発環境はまだ存在しない. また, それらの環境の実現方法についても明らかになっていない. その技法の有効性を検証するためには, それらの環境の実現手法を明らかにした上で, それらの構築を行ない, その上で実際にシステム開発を行なうことが必要である.

抽象解釈をソフトウェア開発に利用する手法は新しい物ではない. 吉岡によって ISDR (Incremental Software development method based on Data Reification) 法が提案されてい

る. これは ML 上で関数の詳細化を扱っている. この研究の問題点は, 副作用やフロー制御などを扱えないことである. 発展的プロトタイピング技法では, 副作用のあるオブジェクト指向言語である Java を対象としている.

2 目的

本研究の目的は, 発展的プロトタイピング技法のための開発環境を構築し, その上で実際に開発を行なうことで, この技法の有効性を検証することである.

開発環境は以下の 2 つからなる.

- 抽象実行処理系
- 発展関係を視覚化する GUI アプリケーション

この技法では, 発展段階が異なるオブジェクト間でのメソッド呼出しを, 実行時にオブジェクト間の発展関係を揃えることで, 可能としている. 本研究ではこのような実行を抽象実行という. しかしこの機構の実現については, まだ十分に議論されていない. そこで本研究では, まず, Java において抽象実行を行なうための機構を明らかにし, 抽象実行のための処理系の設計と実装を行なう.

次に, ソフトウェア開発に有用な GUI アプリケーションの構築を行なう. この GUI アプリケーションは発展的に構成されたプログラムのドキュメントになりうるものを目指す. 一般的に, プロトタイピングにおいては, 正確なドキュメントが作成されない場合が多い. しかし, この技法では, 段階的にプログラムの開発を行なうため, 多くのプログラムを必要とし, それらの関係を表すドキュメントなしに, その関係を正確に把握することは困難である.

我々が提案する GUI アプリケーションは以下の 2 つである.

- 発展関係エディタ
- 抽象実行ビジュアライザ

発展関係エディタは, プログラムの発展関係を静的に表し, また, その関係の編集を可能とする. ビジュアライザは抽象実行時の振る舞いを視覚化する.

上記の開発環境を構築した上で, 実際に開発実験を行ない, 発展的プロトタイピング技法と開発環境を用いてどのようなアプリケーションを構築できるか示す. また, 本技法の特色である抽象実行の有効性を明らかにする.

3 アプローチ

抽象実行では, 異なる発展段階にあるオブジェクト間でメソッド呼出しを実現する必要があるため, オブジェクトは複数の発展段階にあるオブジェクトへの参照を実現できなけ

ればならない. 発展によってメソッドシグネチャが変更となった場合, そのオブジェクトの型も変更されてしまうが, Java は強く型付けられた言語なので, 型が異なるオブジェクトへの参照は単純には実現できない.

我々はこの問題を解決するために, 抽象実行のためのプロキシオブジェクトを提案した. プロキシオブジェクトは異なる発展段階にあるオブジェクト間を結び付けることを可能とする. 本研究では, Java の Reflection 機能と XML 技術を用いて, プロキシオブジェクトを実現した. リフレクションは, 実行中のプログラムが自分自身の情報を検査したり, プログラムの内部を変更することを可能とする. プロキシオブジェクトは, この技術を用いて, 異なる発展段階にある複数のオブジェクトを扱うことを可能としている. 発展関係の記述には XML を用いた. 本研究ではプログラムの発展関係は木構造を仮定しているので, XML は発展関係を記述するのに適している.

発展関係を視覚化するツールの構築では, Java の GUI ツールキットの Swing を使用した. 静的なプログラム発展関係は, Swing の木構造を表すコンポーネントである JTree によって視覚化した. 抽象実行の視覚化は, UML のダイアグアムの一つであるシーケンス図を用いた. シーケンス図は, 時間の流れに注目して, オブジェクト間の相互作用を表す. この特徴によって, 我々はある実行においていつ抽象化が発生したのか, 理解することが容易になると考えている.

4 開発実験

我々は, 構築した開発環境の上で, ブラックジャックゲームと在庫管理システムの 2 つの開発実験を行なった.

前者の開発実験では, まず, トランプカードを一つの抽象値に抽象化し, それを用いて最も抽象的なプログラムを構築する. 次に, その抽象値を徐々に実際のトランプカードに具体化していくことで, 段階的にプログラムの開発を行なう. この実験において, データをうまく抽象化することで, 抽象実行が効果的であることを 2 つの例を用いて示した. 1 つめの例では, スタブを用いた実行よりも, 抽象実行のほうが, 正しい結果を得ることができ, 実行のコストも低いことを示した. またもう 1 つの例として, 抽象実行を用いたテストによって, 具体的なプログラムに含まれるバグを発見することが可能であることを示した.

後者の実験では, 構築するアプリケーションによっては, うまく抽象値を定めることができないことを示した. 例えば, 在庫管理システムの開発では, 在庫数量をうまく抽象値として定めることができない. うまく抽象化されていないデータドメイン上では, 演算を厳密に定義することができない場合がある. そのため, そのデータドメインを使用するシステムは実行できない場合がある.

この問題を解決するために, その演算を実行時に決定する機構を提案した. この機構を用いることで, うまく定義されていないデータドメインを用いるプログラムでも実行することが可能となる.

5 結論

本研究では, 発展的プロトタイピング技法のための開発環境を構築した. この環境は, 抽象実行処理系と発展関係を視覚化する発展関係エディタと抽象実行ビジュアライザからなる. そして, その環境の上で 2 つの開発実験を行い, その結果をふまえ, データドメイン上の演算を実行時に定義するためにシステムの改善を行なった.

以上より, 本研究では次の 2 点を結論として得た.

- 抽象値をうまく定めることで, 抽象実行を効果的に行なうことができる.
- アプリケーションによっては, うまく抽象値を定めることができないことがあるが, その場合においても抽象実行を行なうことが可能である.