B-017

リバースエンジニアリングによる UML をベースとした拡張シーケンス図の生成

Generation of Expanded Sequence Diagram based on UML by Reverse Engineering

堀田 吉彦† 大久保 弘崇‡ 粕谷 英人‡
山本 晋一郎‡ 斉藤 邦彦§
Yoshihiko HOTTA Hirotaka OHKUBO Hideto KASUYA
Shinichiro YAMAMOTO Kunihiko SAITO

1 リバースエンジニアリングによる拡張シーケンス図の生成

シーケンス図をリバースエンジニアリング生成するには，主に動的解析と静的解析の 2 つの手法がある．動的な解析手法では，プログラムの実行パスを網羅できないことや，プログラム断片を対象とできないという問題がある．一方，静的解析では，全てのパスを網羅するが，全てのパスが同じ重みなので，プログラムを理解したい人にとって，どのパスが重要であるかということがわかり難い．また，動的な計算を追跡することができないという欠点がある．そこで，本研究では関数呼び出しの実行パスを網羅して表現することを重視し，静的解析でベースとなるシーケンス図を作成する．そのシーケンス図上に動的解析から得られるパスを重ねあわせることで，双方の解析の欠点を補完し，プログラムの理解を促進することを目的としている．

1.1 静的シーケンス図作成の手順

本研究では以下の 3 段階でシーケンス図を作成する．

1. ソースプログラムの静的解析を行ない関数の呼び出し関係を取得する．
2. 呼び出す関数と呼び出される関数の情報を元にコールツリーを作成する．
3. 2 の結果を利用して静的シーケンス図を描画する．

1.1.1 静的解析

ソースプログラムから，そのソースプログラム内に定義された関数とブロック内で呼び出している関数の情報を取得する．例として図 1 のプログラムに静的解析を行うことで図 2 で示されている関数の呼び出し関係の情報の取得が可能である．図 2 の情報を用いてシーケンス図を描くことができる．

```
 1: void g() {
 2:     ...
 3: }
 4: void f() {
 5:     g();
 6: }
 7: void main( int argc, char *argv[] ) {
 8:     if (argc > 2) f();
 9:     else  g();
10: }
```

図 1 解析対象プログラム (test.c)

```
func:g:test.c:1
func:f:test.c:4              <ー関数 f の宣言
    call:g:test.c:5           <ー関数 g の呼び出し
func:main:test.c:7
    call:f:test.c:8
    call:g:test.c:9
```

図 2 静的解析結果

1.1.2 拡張シーケンス図

本研究では，非オブジェクト指向である C 言語の関数呼び出し関係を表現するのに，UML シーケンス図の形式に着目した．UML[1] シーケンス図の，主要な構成要素は生存線，実行指定，メッセージである．オブジェクト指向において，生存線はオブジェクトとその生存期間を表わす．実行指定は生存線上で発生し，メソッドが活動している期間を表わす．メッセージは生存線間でのコミュニケーション (通信) を表わす．

関数の呼び出し関係を表現するコールグラフでは，関

† 愛知県立大学大学院情報科学研究科
‡ 愛知県立大学情報科学部
§ 滋賀大学経済学部

数実行の回数や順番を適切に表現するのは困難であり，時間 (実行順) 軸のあるシーケンス図による表現は有用である．本研究では UML のオブジェクトを表わすライフラインには C 言語の関数を適用する．生存線の長さには意味を持たない．実行指定は，オブジェクト指向においてはメソッドの活動期間であり，本研究においては関数の活動期間を表現する．また活動期間を起動するメッセージには関数の呼び出しを適用した．図表現としては関数は横長の矩形と垂線で示し関数名は矩形の中に記述する．

図 3 では `main`, `f` , `g` の 3 つの関数が存在し，`main` からは `g`, `f` が，`f` からは `g` が呼ばれるというパスが存在することがわかる．

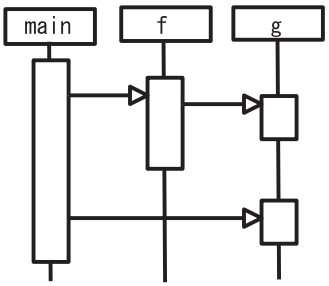

図 3　拡張シーケンス図

1.2　実行履歴

本研究における C プログラムの実行履歴とは，ユーザが定義した解析対象である関数の実行開始時と終了時，及びその関数が宣言されている C ファイルと行数を時系列順に列挙したものであると定義した．図 4 は，実際にプログラムを動作させ取得した実行履歴である．この情報を図 3 の情報に重ね合せると `main` → `f` → `g` というパスで呼び出しが起っており，`main` → `g` というパスは通過しないということがわかる．

```
1: ENTER: main:test.c:7
2: ENTER: f:test.c:4
3: ENTER: g:test.c:1
4: EXIT: g:test.c:1
5: EXIT: f:test.c:4
6: EXIT: main:test.c:7
```

図 4　実行履歴

2　実装

本研究では，C プログラムを対象としたシーケンス図描画ツールを実装した．ソースプログラムの静的解析部には Sapid[1] を利用している．また実行履歴には GCC の GNU 拡張である`-finstrument-functions` オプションとアドレスから関数名を取得するのに addr2line[2]を利用しソースプログラムに手を加えずに情報の取得を可能である．シーケンス図を表示する GUI 部分は Java Swing で実装している．動作確認は gzip(1 万行程度) に適用することで行った．

3　関連研究

[2, 3] では Java を対象に実行履歴のみからのシーケンス図生成を行なっている．実行履歴の圧縮手法 [3], 動的スライシング手法 [2] が特徴である．[4] では C++ を対象に静的解析によるシーケンス図，コラボレーション図の作成を行なっている．静的解析の手法では，動的束縛などの静的解析では決定できない情報をオブジェクトフロー解析を利用して近似させている．UML モデリングツール Rational Rose[3]は，C 言語の実行トレースをシーケンス図形式で記述している．

4　おわりに

本研究では，ソースプログラムの解析結果から拡張シーケンス図を生成し，実行プログラムから得る実行履歴を重ねあわせる手法を提案し，そのツールを実装した．今後の課題としては，(1) 拡張シーケンス図上における動的情報の効果的な見せ方を検討する，(2) ソースプログラムからコントロールフローに関する情報を抽出しシーケンス図の記法を UML2.0 に準拠させることが挙げられる．

参考文献

[1] Unified Modeling Language (UML)2.0 specification. OMG, 2005.

[2] 小林隆志, 堅田敦也, 鹿内将志, 佐伯元司: プログラムスライシングを用いた Java 実行系列からの部分シーケンス生成手法. ソフトウェア工学の基礎 XI, 日本ソフトウェア科学会 FOSE2004, pp.17–28, 2004.

[3] 谷口 考治, 石尾 隆, 神谷 年洋, 楠本 真二, 井上 克郎: Java プログラムの実行履歴に基づくシーケンス図の作成, ソフトウェア工学の基礎 XI，日本ソフトウェア科学会 FOSE2004，pp5–15, 2004.

[4] P. Tonella, A. Potrich: Reverse Engineering of the Interaction Diagrams from C++ Code. Proceedings of International Conference on Software Maintenance, pp.159–168, 2003.

[1] CASE ツールプラットフォーム Sapid, http://www.sapid.org/

[2] GNU Binary Utilites, http://www.gnu.org/software/binutils/

[3] Rose, http://www-306.ibm.com/software/rational/