社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

UMLをベースとしたJavaプログラムの視覚化における手続的処理部の表現について

堀田　吉彦†　大久保弘崇††　粕谷　英人††　山本晋一郎††　斎藤　邦彦†††

† 愛知県立大学大学院情報科学研究科 〒480-1198 愛知群長久手町大字熊張字茨ヶ廻間 1522-3
†† 愛知県立大学情報科学部 〒480-1198 愛知県長久手町大字熊張字茨ヶ廻間 1522-3
††† 滋賀大学経済学部 〒522-8522 滋賀県彦根市馬場 1 丁目 1-1

あらまし　UML は OOP のモデリング言語だけでなく，OOP を可視化するときにも用いられる．中でも UML のシーケンス図はオブジェクト相互のメッセージのやりとりを図示することを目的としている．この図では，自分自身あるいは自分と同じクラスへのメッセージ送信の表現が難しい．調査の結果，そのようなメッセージ送信はプログラム中に約 10%存在する．OOP の視覚化を目的として，この点に関する UML の拡張を提案する．
キーワード　UML，シーケンス図，視覚化，Java，手続的処理部

On a Representation of Procedural Processing Parts in UML Based Visualization of Java Programs

Yoshihiko HOTTA†, Hirotaka OHKUBO††, Hideto KASUYA††, Shinichiro YAMAMOTO††, and Kunihiko SAITO†††

† Graduate School of Information Science and Technology, Aichi Prefectural University 1522-3, Ibaragabasama, Kumabari, Nagakute, Aichi-gun, Aichi, 480-1198, Japan
†† Faculty of Information Science and Technology, Aichi Prefectural University 1522-3, Ibaragabasama, Kumabari, Nagakute, Aichi-gun, Aichi, 480-1198, Japan
††† Faculty of Economics, Shiga University 1-1-1, Baba, Hikone, Shiga, 522-8522, Japan

Abstract　UML is a modeling language for OOP, but there is a case to visualize OOP. The principal objective of UML sequence diagram is to illustrate exchanges of messages between objects. Some part of program sends messages to oneself or its class, and it is difficult to express such part in UML sequence diagram. Our investigation revealed that 10% of message sends are to itself. We propose an expansion of UML sequence diagram to illusutrate such part more effciently.
Key words　UML, Sequence Diagram, Visualization, Java, Procedural Processing Part

1. はじめに

ソフトウェアの保守を行う際，プログラムの理解に多くの時間が費やされる．ソースプログラムに存在する情報は膨大であり，それらを理解するのは非常に困難である．

プログラムにある膨大な情報の視覚化を行うことで，プログラムの直観的な理解を助けることができる．

UML [1], [2] は主にオブジェクト指向プログラム (OOP) の要求分析，設計のフェーズで用いられるグラフィカルなモデリング言語である．また，既存のソースプログラムを視覚化する際に用いられる場合もある．

UML は OOP の様々な側面を図示することが可能なので OOP の構造や振舞いの情報を視覚化するのに適している．

OOP の計算とはオブジェクト間のメッセージ交換である．UML の中でもシーケンス図は，OOP の振舞いとして，オブジェクト間のメッセージのやりとりを図示するものである．

シーケンス図は他のオブジェクトへのメッセージ送信を表現することを主眼としているので，自分自身あるいは自分のクラスへのメッセージ送信を適切に表現することが困難であるという問題が存在する．自分自身あるいは自分と同じクラスへのメッセージとは，自分自身に対するメソッドコールやクラスメソッドのコールであり，OOP でありながら手続的処理を行っ

ている部分のことである．以下これを**手続的処理部**と呼ぶ．

本論文では，UML のシーケンス図で適切に表現することが困難な手続的処理部を，クラスに属するメソッドに着目することでより適切に表現するための図示法を考案し，シーケンス図を拡張した**手続的メソッドシーケンス図**を提案する．手続的メソッドシーケンス図を Java ソースプログラムから生成するツールを作成し，その効果を確認する．

2 節では，OOP における手続的処理について説明する．3 節では，手続的処理部を多く含むプログラムに対する UML シーケンス図の表現上の問題を述べる．4 節では，3 節で示した問題を解決する手続的メソッドシーケンス図を提案する．その表記方法を述べる．また，手続的メソッドシーケンス図と UML シーケンス図を連携させることで，手続的メソッドシーケンス図の表現が得意としない OO 部を適切に表現できることを示す．5 節では，提案手法の実装システムをオープンソースプログラムに対して適用することで評価実験を行う．最後に 7 節で本研究のまとめと今後の課題にについて述べる．

1.1 関 連 研 究

Java プログラムからシーケンス図を作図する研究が多数行なわれている．文献 [3] [4] は Java プログラムの実行履歴を用いて，動的情報からシーケンス図を作成している．文献 [4] は，実行履歴から繰り返しなどの構造を検出しまとめることで，膨大となるシーケンス図をコンパクトにしている．文献 [5] は，Java プログラムのソースを用いて静的解析結果からシーケンス図を生成している．本論文では，文献 [5] の手法を参考にして静的なシーケンス図の生成を行なっている．また，実行履歴からシーケンス図を生成するツールも多数存在する [10] [11]．これらの研究やツールでは，本研究と異なり，シーケンス図の手続的処理部の表現力に対する問題は考慮されていない．また，従来の手続型プログラムの伝統的な表現図式であるフローチャートや PAD は，表現する情報の粒度が細かすぎ，OO 部を表現するシーケンス図と整合しない．

2. OOP における手続的処理部

OOP では，基本的にオブジェクト間のメッセージのやりとりによってプログラムが動作動作するが，自分自身や自分のクラスへのメッセージ送信も存在する．図 1 は，自分自身，自分のクラス，他のオブジェクトへのメッセージ送信の例を示している．

本論文では，Java で記述されたオープンソースソフトウェアに対し，静的にクラス内で発生するメッセージ送信としてのメソッドコールの数と，メッセージ送信の対象が「自分自身および自分の属するクラス」であるか「他のオブジェクト」であるかの割合の調査を行った．

調査の結果を表 1 に示す．結果から約 10%のメッセージ送信が自分自身および自分の属するクラスを対象としているということがわかる．

調査したオープンソースソフトウェアの多くが約 10%の値を示しており，この値より大幅に大きい値を示すソフトウェアは，クラスおよびメソッド構成を再検討し機能分割する余地があると考えられる．クラスやメソッドの再構成を行う際，プログラム中の手続的処理が多用されているクラスの理解支援のための視覚化が必要であると考えられる．

```
public class Hoge {

        public OtherObject obj;

        public Hoge() {
                obj = new OtherObject();
                this.m(); /*自分自身へ*/
        }

        private void m() {
                Hoge.m2(); /*自分のクラスへ*/
                obj.m3() /* 他のオブジェクトへ*/
        }

        private static void m2() {
                ...
        }
}
```

図 1　送信先の異なるメッセージ送信の例

3. UML シーケンス図における問題点

UML のシーケンス図は OOP の振舞いを表現する図である．図 2 は図 1 のプログラムのシーケンス図である．

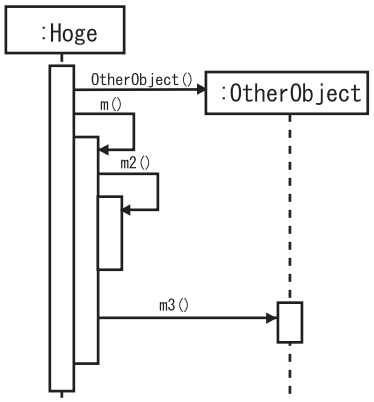

図 2　シーケンス図

シーケンス図は，ライフライン，活性区間，メッセージ送信から構成される．図 2 中の “:Hoge”, “:OtherObject” を囲っている矩形とその矩形の下部に伸びる破線の垂線の総称がライフラインである．ライフラインは，OOP におけるオブジェクトやクラスを表現する．ライフラインの破線の上の縦長の矩形は活性区間である．活性区間はオブジェクトが制御を保持している区間を表現する．活性区間はメッセージ送信によって開始し，制御をメッセージ送信元へ返すことで終了する．メッセージ送信は，活性区間から，活性区間またはライフラインへの矢印であり，矢印の元の活性区間のオブジェクトから先のオブジェクトへのメッセージ送信を表す．また，そのメッセージ送信がオブジェクトの生成を指示するものである場合は，ライフラインへの矢印となる．

シーケンス図はオブジェクト間のメッセージのやりとりを表わすための図である．しかし，図 2 中のメッセージ送信 m() や

表 1　自分自身へのメッセージ送信の割合

ソフトウェア名	クラス数	総メッセージ送信数	自分自身への送信数	割合 (%)
Apache Ant	929	28189	2838	10.1
一太郎 Ark	761	35836	3655	10.2
じゅん for Java	3007	107189	13624	12.7
eclipse	23192	663251	71287	10.7
XercesJ	565	19422	2217	11.4
JRuby	575	18753	1913	10.2

m2() のような手続的処理部では，自分自身もしくは自分自身のクラスにメッセージを送信している．これをシーケンス図で表すと，自分自身のオブジェクトやクラスに対するループになってしまう．複雑なコントロールフローや，多段の自分自身へのメッセージ送信が存在する手続的処理部を，OOP の図示のためのシーケンス図で表現することは問題があるといえる．

4. 手続的メソッドシーケンス図の提案

本節では，OOP における手続的処理部を適切に表現する，手続的メソッドシーケンス図の記法を提案する．また，UML シーケンス図とどのように連携するかを説明する．

4.1 手続的メソッドシーケンス図の記法

手続的メソッドシーケンス図は，特定のクラスに着目し，そのクラスの動作理解を支援するために，UML シーケンス図の一部を詳細化した表現である．

図中の要素として新たに以下の要素を導入する．

（1） メソッド

（2） フィールド (基本型および参照型)

（3） フィールドへのアクセス

UML シーケンス図では，オブジェクト間のメッセージ送信をメソッド名のラベルを付けた矢印で表現する．手続的メソッドシーケンス図では，メソッドからメソッドへの呼び出しを示すために，メソッドを UML シーケンス図におけるオブジェクトであるかのように表現する．

また，手続的処理部での大域変数に相当するフィールドとそのフィールドにアクセスするメソッドの関係は手続的処理部を理解するのに重要であるため，フィールドおよびフィールドへのアクセスの概念も導入する．

a） メソッドの表現の拡張

メソッドは，オブジェクトを表わす矩形の下部を，そのオブジェクトが持つメソッドで分割した矩形で表現する (図 3)．メソッド m がメソッド m1 を呼び出す時，図 4 のようにメソッド m の矩形の下部に伸びた垂線から，メソッド m1 の矩形の下部に伸びた垂線に対しての矢印で表現する．

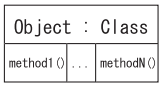

図 3　メソッドの表現

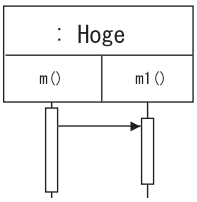

図 4　メソッドの呼び出しの表現

b） フィールドの拡張

Java 言語におけるフィールドは，クラス内全てをスコープとする変数であり，複数のメソッドから参照される．基本型のフィールドに対する操作は，読み書きのみで，メッセージのやり取りがない．一方，参照型である場合は，そのフィールドが参照しているオブジェクトに対するメッセージ送信が行なえる．フィールドは，メソッドと同様にオブジェクトの矩形の下部を分割した矩形で表現する．図 6 で例を示す．

フィールドがオブジェクトを参照している時，フィールドの矩形の下部に伸びた垂線にオブジェクトをあらわす横長の矩形を配置する．その矩形はオブジェクトのライフラインであり UML のシーケンス図のオブジェクトと同様の表現で記述する．

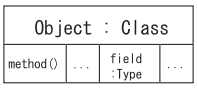

図 5　フィールドの表現

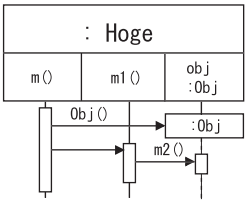

図 6　フィールドに参照されるオブジェクトの表現

4.2 UML シーケンス図との連携

手続的メソッドシーケンス図は手続的処理部の詳細を表現できるが，記述の粒度が細かくなるため，オブジェクト指向部の表現では従来の UML シーケンス図に劣る．そこで．UML シーケンス図と連携させることで，その欠点を補う．

UML シーケンス図には，UML2.0 で相互関連という記法が新しく追加されている．相互関連を用いると図 7 のようにシーケンス図中に，他のシーケンス図への参照を配置できる．

相互関連を用いることで，大規模なシーケンス図を階層的に表現することが可能である．手続型メソッドシーケンス図はシーケンス図を基にした記法を用いるため，相互関連で参照することで，図 8 のようにシーケンス図とのシームレスな連携が可能となる．

4.3 手続的メソッドシーケンス図の適用例

本節では，複雑な手続的処理を持つプログラムの例を用いて，提案手法である手続的シーケンス図と UML シーケンス図の比較を行う．

図 10 と図 11 は，それぞれ図 9 のプログラムを表現する，UML シーケンス図と手続的メソッドシーケンス図である．

図 10 の UML シーケンス図の手続的処理部は，自身へのメッ

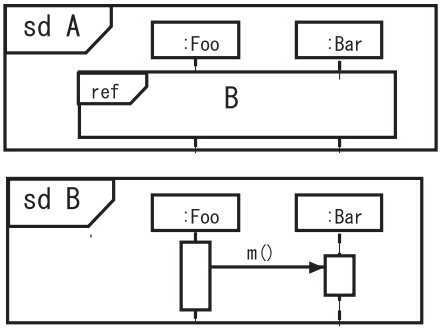

図 7　相 互 関 連

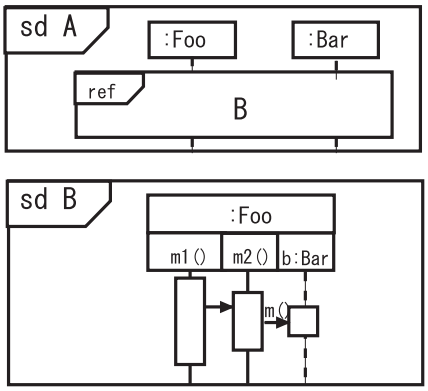

図 8　UML シーケンス図との連携

セージ送信が多数あり，一目でどのメソッドがどのメソッドを呼び出すかを把握することは困難である．一方，図 11 の手続的メソッドシーケンス図ではどのメソッドがどのメソッドを呼び出しているかは，上部のメソッド名を見ることで容易に把握することができる．

以上から提案手法がオブジェクト指向プログラム内に存在する手続的処理部の表現において，従来の UML シーケンス図よりも適切に表現することが可能となっている．

```
public class A {

        public static void main(String[] args) {
                A a = new A(args[0]);
        }

        public A(String s) {
                ma1(s);
        }

        public void  ma1(String s) {
                if (s.equals("hoge")) { ma2(); }
                else { ma3(); }
        }

        public void ma2() {
                for (int i = 0; i < 10; i++) { ma3(); }
        }

        public void m3a() {
                B b = new B();
                b.mb1();
        }
}
```

図 9　サンプルプログラム

5. 評　　価

本節では，提案手法を実装したシステムを用いて実験を行い，提案手法の有効性の評価を行う．

5.1 システムの概要

手続的メソッドシーケンス図描画システムは，Java プログラムの細粒度な静的解析情報を用いて手続的メソッドシーケンス図と UML シーケンス図を描画する機能を持つ CASE ツールである．システムの構成は，Java プログラムの静的解析部，シーケンスモデル生成部，シーケンス図描画ライブラリから成る．

5.1.1　静的解析部

Java プログラムの構文・意味解析には，CASE ツールプラットフォーム Sapid [6] 上の Java 用の解析器 Japid [7] を用いている．Japid は，Java プログラムの細粒度な解析情報を XML リポジトリ [8] として出力する．また Japid で解析した情報はファイル内の関係に関しての情報なので，そのファイル間の相互参照の解決に Jtool [9] を用いている．

5.1.2　シーケンスモデル生成部

シーケンスモデル生成部は，Java プログラムの解析情報から，シーケンス図に関わる情報を抽出したモデルを生成する部分である．Java プログラムのクラスから，宣言されたパッケージ，クラス，メソッド，コンストラクタ名および，オブジェクト生成文，メソッドコール文，for や while 等のループ文，if や switch などの条件文を抽出している．

5.1.3　シーケンス図描画ライブラリ

上記で生成されるシーケンスモデルを入力とし，手続的メソッドシーケンス図と UML シーケンス図を描画する部分である．ユーザインタフェース部は Swing と Java2D を用いて構成している．

5.2 評価実験の概要

Java 言語で記述されたオープンソースプログラムの中から，手続的処理部が多く含まれるクラスおよびメソッドを抽出し，上記システムを用いて手続的メソッドシーケンス図および UML シーケンス図を作成する．その両図の比較を行い，提案手法の有効性を示す．

5.3 評 価 実 験

本実験では，Java 言語で実装された XML パーサである XercesJ に含まれるクラス，TreeWalkerImpl.java 内のメソッド nextNode() を対象として，UML シーケンス図と手続的メソッドシーケンス図の生成を行った．TreeWalkerImpl.java は自分自身を対象としたメッセージ送信の割合が 71%である，手続的処理部が中心となっているクラスである．

図 12 と図 13 は，TreeWalkerImpl.java の nextNode() メソッドを対象として，本システムで生成した UML シーケンス図と手続的メソッドシーケンス図である．提案手法の記法と図 13 ではオブジェクトを示す矩形とメソッドを示す矩形が接していない点が異なるが，意味的な差異はない．

5.4 評価実験結果の考察

図 12 の UML シーケンス図では，自己クラスの関数呼び出しのネストが深くなると図が煩雑になり視認性が悪くなること

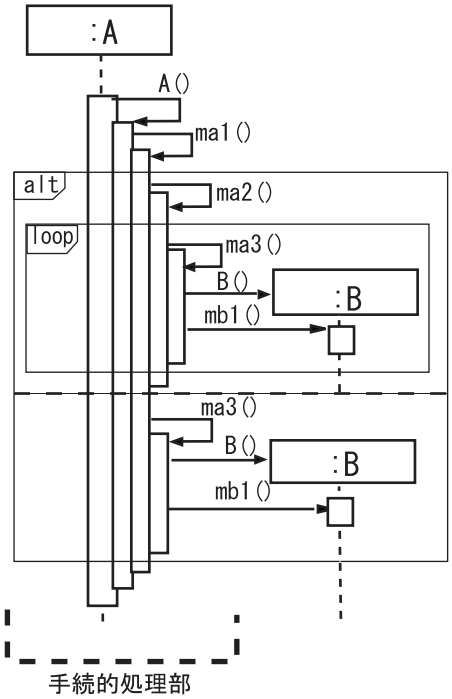

図 10　UML シーケンス図

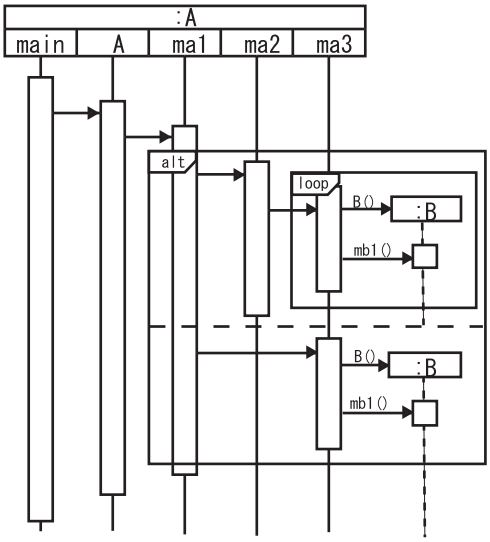

図 11　手続的メソッドシーケンス図

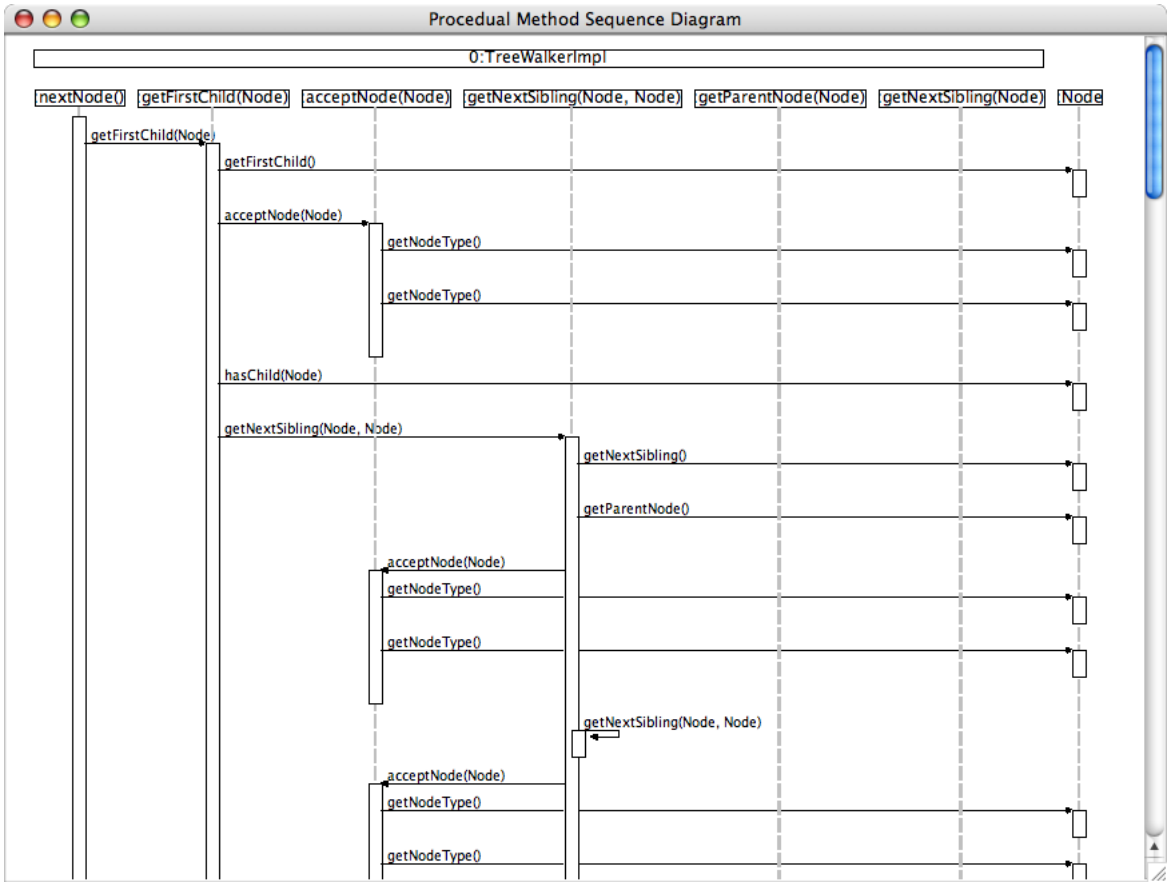

図 13　手続的メソッドシーケンス図

がわかる．一方，図 13 の手続的メソッドシーケンス図では，どのメソッドがどのメソッドから呼び出されているかということを容易に把握できる．よって提案手法は手続的処理部を表現するのに有効であるといえる．

しかし，提案手法は描画面積を必要とするため，大規模なシーケンス図を表示するには適さない．よって，提案手法は単独で利用するのではなく，UML シーケンス図に対して手続的処理部が多いクラスに関して補完的に用いると有効である．

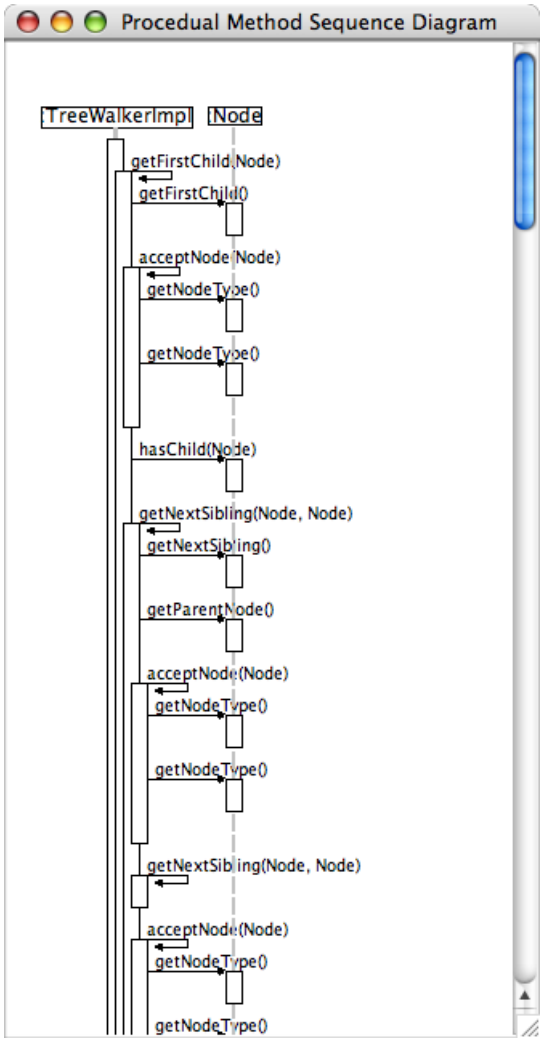

図 12　UML シーケンス図

6. おわりに

本研究では，UML シーケンス図で焦点が当てられていない手続的処理部の表現法である，手続的メソッドシーケンス図を提案した．また，提案手法を実装したシステムを用いて，提案手法の有効性を示した．この手法を用いることで，

6.1 今後の課題

提案手法を UML シーケンス図と効果的に使い分けてプログラムの可視化をすすめるためには，プログラムから手続的処理部の多いクラスを検出する方法が必要である．

文　献

[1] Unified Modeling Language (UML)1.5 specification. OMG, 2003.

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

[3] 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.

[4] 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.

[5] P. Tonella, A. Potrich: Reverse Engineering of Object Oriented Code. Springer, 2005.

[6] 福安 直樹, 山本 晋一郎, 阿草 清滋: 細粒度ソフトウェア・リポジトリに基づいた CASE ツール・プラットフォーム Sapid, 情報処理学会論文誌, Vol.39, No.6, pp.1990–1998, June 1998.

[7] Y. Hachisu, S. Yamamoto and K. Agusa: A CASE Tool Platform for an Object Oriented Language, IEICE Trans. on Information and Systems, Vol.E82-D, No.5, pp.977–984, May 1999.

[8] 吉田 一, 山本 晋一郎, 阿草 清滋: XML を用いた汎用的な細粒度ソフトウェアリポジトリの実装, 情報処理学会 OO2002 シンポジウム, pp.83–90, August 2002.

[9] K. Maruyama, S. Yamamoto: A CASE Tool Platform Using an XML Representation of Java Source Code, SCAM04, pp.158–167, Sep. 2004.

[10] Rational Rose: `http://www-306.ibm.com/software/rational/`.

[11] Eclipse UML: `http://www.eclipseuml.com/`.