UML とソースコードの間でトレーサビリティを リアルタイムに維持するツール RETUSS の現状と課題

森　敬介 a)・片山　徹郎 b)

Current Status and Issues of RETUSS to Ensure Traceability between UML and Java Source Code in Real Time

Keisuke MORI, Tetsuro KATAYAMA

Abstract

It's increasing the importance of software in society, and it's becoming more important to secure the quality of software. Ensuring of the traceability of deliverables is one of effective methods to secure the quality of software. It can verify that the requirements are reflected in the programs, and close the gap between the documents and the source code. But it has two problems: taking labor and time, and causing mistakes by human handling. This paper has implemented RETUSS (Real-time Ensure Traceability between UML and Source-code System) in order to solve the above two problems. RETUSS can ensure the traceability between Class diagram in UML and Java source code in real time.

Keywords:　Software Quality, Traceability, UML, Java

1.　はじめに

社会におけるソフトウェアの重要性はますます高まっており、システム、およびソフトウェアの品質確保がより重要視されてきている。ソフトウェアの品質確保のための方法の 1 つに、成果物のトレーサビリティの維持が存在する [1]。トレーサビリティの維持により、要求がプログラムへと反映されていることの検証、要求変更による影響範囲の特定、ドキュメントとソースコードのズレの解消などが可能になる。しかし、成果物のトレーサビリティの維持には、以下の 2 つの課題が存在する。

- 成果物の一部の変更によって、他の関連している成果物も同じように変更する必要があり、手間と時間がかかること
- 人手に起因するトレーサビリティの維持にはミスの入り込む余地があり、トレーサビリティを維持できなくなる恐れがあること

そこで本研究では、上記 2 つの課題の解決を目的として、UML とソースコード間のトレーサビリティをリアルタイムに維持するツール RETUSS (Real-time Ensure Traceability between UML and Source-code System) の試作を行う。RETUSS は、トレーサビリティの維持を自動化することで、手間と時間を削減でき、人手に起因するミスを除去できる。

なお、UML (Unified Modeling Language)[2, 3]とは、ソフトウェアの設計とパターンを表現するための視覚的な言語であり、要求仕様書やシステム設計書などに用いられている [4]。RETUSS では、複数存在する UML ダイアグラムのうち、システムの静的な構造を表す重要なダイアグラムである、クラス図を対象とする。また、ソースコードは Java 言語を対象とする。

2.　RETUSS の外観と機能

本研究で試作したツール RETUSS の外観を、図 1 に示す。RETUSS は、メニューバー、ファイル一覧エリア、描画アイテム選択エリア、UML 記述エリア、ソースコード記述エリアからなる。

また、本研究で試作した RETUSS は、以下の 3 つの機能を持つ。

- java ファイルの展開
- クラス図の記述
- Java ソースコードの記述

以降、各機能についてそれぞれ説明する。

a)工学専攻 機械・情報系コース大学院生
b)情報システム工学科准教授

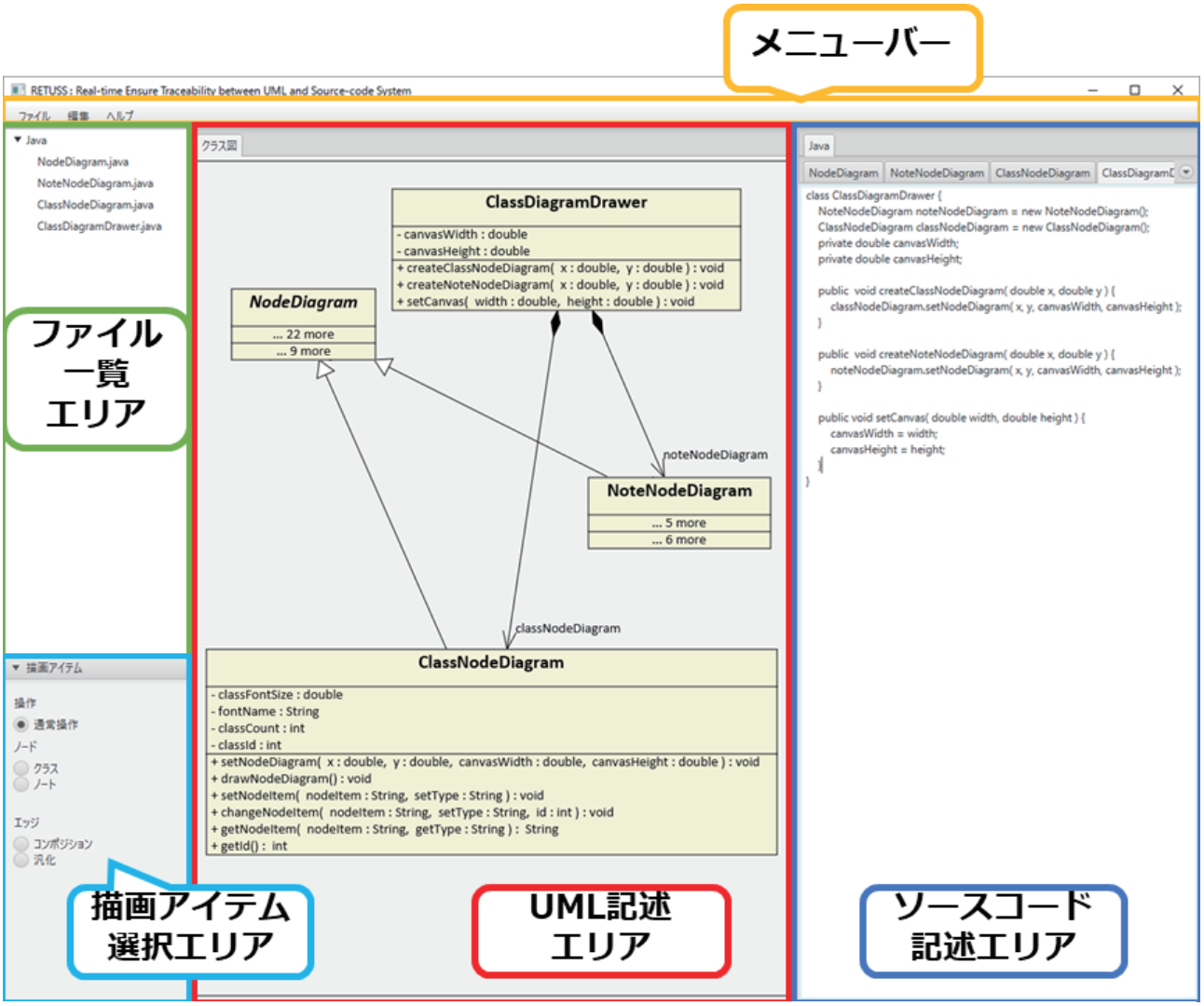

図 1. RETUSS の外観

2.1 java ファイルの展開

ユーザがメニューバーの「ファイル」メニューアイテムの「Java」メニューを選択し、「java ファイルを開く」メニューアイテムを選択すると、ファイルオープンダイアログを表示する。ファイルオープンダイアログから任意のファイルを指定すると、そのファイル内のテキストデータをソースコード記述エリアに表示する。

「java ファイルの展開」機能を用いると、ソースコード記述エリアにファイル内のテキストデータを表示する。そして、テキストデータを解析し、ファイル一覧エリアにファイル名として、クラス名に文字列「.java」を後ろに追加した文字列を表示し、UML 記述エリアにそのテキストデータに対応するクラスを描画する。更に、Java ソースコードにおける継承がテキストデータに含まれており、かつ継承クラス名に該当するクラスが UML 記述エリアに既に描画済みの場合、その 2 つのクラス間に汎化関係を描画する。

2.2 クラス図の記述

UML 記述エリアにおいて、ユーザはクラス図を記述できる。クラス図を記述すると、その内容に対応した Java ソースコードをソースコード記述エリアにリアルタイムで表示する。記述できる項目および機能を、以下に示す。

- クラスおよびノートの記述
- クラスおよびノートの移動
- クラスの名前およびノートの内容の変更
- クラスおよびノートの削除
- クラスの属性および操作の追加
- クラスの属性および操作の変更
- クラスの属性および操作の削除
- クラスの属性および操作の非表示
- クラス間のコンポジションおよび汎化の記述

2.3 Javaソースコードの記述

ソースコード記述エリアにおいて、ユーザはJavaソースコードを記述できる。Javaソースコードを記述すると、その内容に対応したクラス図のクラスをUML記述エリアにリアルタイムで描画する。

ソースコード記述エリアには、複数のクラスに対応したJavaソースコードを表示する。1つのタブが1つのクラスに対応し、各タブのテキストエリアに、対応するクラスのJavaソースコードを表示する。

3. RETUSSの実装

RETUSSの機能の一部である「クラス図の記述」機能の実装、およびクラス図とJavaソースコードの対応について説明する。

3.1 クラス図の記述

「クラス図の記述」機能の実装について説明する。

- 要素の描画
  UML記述エリアをクリックすることで、クラスおよびノートに相当するイメージを描画する。また、描画済みの要素をドラッグすることで、その要素を移動する。描画済みの要素を右クリックすると、変更または削除を選択するメニューを表示する。
- クラスの属性および操作の描画
  UML記述エリアに記述済みのクラスを右クリックすると、属性および操作の追加、変更、削除、表示を選択するメニューを表示する。追加メニューは、属性および操作を追加する。変更および削除メニューは、クラスに追加済みの属性および操作の一覧を表示し、選択した属性および操作を変更または削除する。表示メニューは、クラスに追加済みの属性および操作のチェックボックスメニューの一覧を表示し、チェックを外した属性および操作を非表示にする。
- クラス間のコンポジションおよび汎化の描画
  UML記述エリアのクラスを2つ選択することで、そのクラス間のコンポジションまたは汎化を描画する。

3.2 クラス図とJavaソースコードの対応

クラス図およびJavaソースコードにおける各項目の対応関係を、表1に示す。

表1における各項目の対応関係の多くは、UML2.0の仕様書[2)]で既に定義されている。定義されていない一部の対応関係については、今回独自に定義している。

4. 適用例

本研究で試作したRETUSSの機能が正しく動作することを検証するために、RETUSS自体の構造の一部を用いたクラス図を適用する。適用例を実行した際のRETUSSの画面を図1に、クラス図とJavaソースコードとの対応を図2に、それぞれ示す。

図2を見ると、UML記述エリアのクラス図とソースコード記述エリアのJavaソースコード間のトレーサビリティを維持していることが確認できる。よって、RETUSSの機能が正しく動作していることがわかる。

5. RETUSSの有用性の評価

本研究で試作したRETUSSの有用性を評価するため、被験者を用いた実験を行う。実験方法として、トレーサビリティを維持している状態のクラス図とソースコードそれぞれの修正を、被験者が行う。クラス図とJavaソースコードを修正する方法として、RETUSSを用いてクラス図とJavaソースコードを修正するケースA.と、RETUSSを用いずにクラス図とJavaソースコードを修正するケースB.の、2つのケースを用意する。それぞれのケースについて、被験者が修正後のクラス図とJavaソースコード間のトレーサビリティを維持した状態になるまでに要する時間を測定し、比較する。それぞれのケースにおける、クラス図とソースコード間のトレーサビリティ維持に、3人の被験者が要した時間を、表2に示す。

RETUSSを用いてクラス図およびJavaソースコードの修正に要した時間の平均は、106秒である。一方、RETUSSを用いずにクラス図およびJavaソースコードの修正に要した時間の平均は、223秒である。つまり、RETUSSを用いてクラス図およびJavaソースコードを修正することで、用いなかった場合と比べてトレーサビリティの維持にかかる時間を、52.46%短縮できた。

また、RETUSSを用いないケースB.では、トレーサビリティの維持を人手で行ったため、修正途中に、クラスの可視性とJavaソースコードのアクセス修飾子の不一致、Javaソースコードのメソッドボディの中括弧の削除、などのミスが発生した。一方、RETUSSを用いるケースA.では、トレーサビリティの維持を自動で行うため、上記のようなミスは発生しなかった。

以上のことから、RETUSSを利用することによって、手間と時間を削減でき、人手に起因するミスを無くすことができる。

6. RETUSSの課題

RETUSSの実装および考察を行った上で、明らかになった課題を次に示す。

表 1. クラス図および Java ソースコードにおける各項目の対応関係

クラス図		Java ソースコードとの対応関係
クラス	クラス名	クラス名として宣言する。Java ソースコードでは「class クラス名 {}」となる。
	属性	フィールドとして宣言する。クラスにおいて「可視性 名前 : 型」と記述し、Java ソースコードでは「アクセス修飾子 型 名前;」となる。
	操作	メソッドとして宣言する。クラスにおいて「可視性 名前 (パラメータ名 : パラメータの型) : 戻り値の型」と記述し、Java ソースコードでは「アクセス修飾子 戻り値の型 名前 (パラメータの型 パラメータ名) {}」となる。
可視性	+	アクセス修飾子として宣言する。Java ソースコードでは「public」となる。
	-	アクセス修飾子として宣言する。Java ソースコードでは「private」となる。
	#	アクセス修飾子として宣言する。Java ソースコードでは「protected」となる。
	~	アクセス修飾子を何も記述しない。
コンポジション		フィールドとして宣言する。Java ソースコードでは「コンポジション先のクラス名 コンポジション先の関連端名 = new コンポジション先のクラス名 ();」となる。
汎化		継承クラスとして宣言する。Java ソースコードでは「class クラス名 extends 継承クラス名 {}」となる。
抽象クラス		abstract 修飾子を付けたクラスとして宣言する。Java ソースコードでは「abstract class クラス名 {}」となる。
抽象メソッド		abstract 修飾子を付けたメソッドとして宣言する。Java ソースコードでは「abstract メソッド」となる。

- 同じ属性および同じ操作に未対応
  RETUSSは、UML記述エリアにおいて、同じ属性および同じ操作に対応できない。これにより、同じ属性および同じ操作が2つ表示されてしまうため、対応する必要があると考える。この課題を解決するためには、属性および操作を追加する際に、既に追加済みの属性および操作を調べ、同じものが存在する場合は入力できないように実装しなければならない。
- Javaソースコードにおける一部の構文に未対応
  RETUSSには、Javaソースコードにおける未対応の構文が存在する。これにより、未対応の構文をクラス図に反映できず、トレーサビリティの維持ができなくなるため、対応する必要があると考える。この課題を解決するためには、RETUSSの解析結果から未対応の構文内容を抽出し、クラス図との対応規則を実装し、クラス図に反映しなければならない。
- クラス図のクラスの属性および操作の一部の構文に未対応
  RETUSSには、クラス図のクラスの属性および操作における未対応の構文が存在する。これにより、未対応の構文をJavaソースコードに反映できず、トレーサビリティの維持ができなくなるため、対応する必要があると考える。この課題を解決するためには、クラス図のクラスの属性および操作の構文解析を行い、各項目を抽出してJavaソースコードに反映しなければならない。
- クラス図の記述機能が一部未対応
  RETUSSのUML記述エリアには、「関係を削除」「関連端名の変更」など、未実装の機能が存在する。これにより、RETUSSのUML記述エリアにおける描画能力は十分であるとはいえないため、対応する必要があると考える。

7. おわりに

本研究では、トレーサビリティの維持における2つの課題の解決を目的として、UMLとソースコード間のトレーサビリティをリアルタイムに維持するツールRETUSSの試作を行った。トレーサビリティの維持における2つの課題を以下に示す。

- 成果物の一部の変更によって、他の関連している成果物も同じように変更する必要があり、手間と時間がかかること
- 人手によるトレーサビリティの維持にはミスの入り込む余地があり、トレーサビリティを維持できなくなる恐れがあること

試作したRETUSSは、以下の3つの機能を持つ。

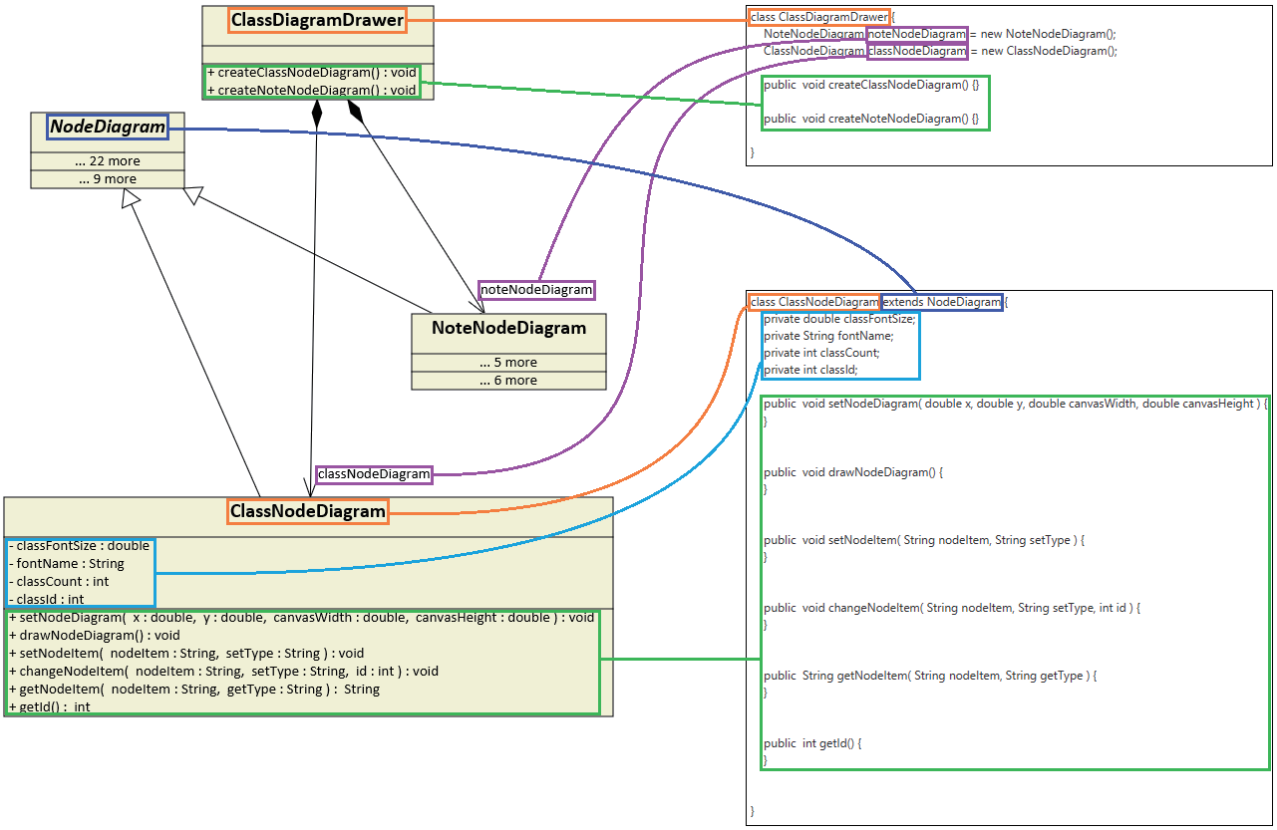

図 2.　適用例を実行した際のクラス図と Java ソースコードとの対応

表 2.　クラス図と Java ソースコード間のトレーサビリティ維持に要した時間（秒）

試行回数	ケース A.	ケース B.
1	113	292
2	76	148
3	129	229
平均	106	223

- javaファイルの展開
- クラス図の記述
- Javaソースコードの記述

適用例として、本研究で試作したRETUSSに、RETUSS自体の構造の一部を用いたクラス図を適用することで、RETUSSが正しく動作することを確認できた。

更に、被験者を用いた実験として、修正後のクラス図とソースコード間のトレーサビリティの維持について2種類のケースを用意し、それぞれのケースにおける、クラス図とソースコード間のトレーサビリティを維持するまでにかかる時間を計測し、比較した。クラス図とソースコード間のトレーサビリティを維持するまでにかかる時間は、RETUSSの機能を用いた場合、他の手段を用いた場合と比べて、52.46%短縮できた。また、RETUSSを用いることにより、トレーサビリティの維持を自動で行うため、人手に起因するミスを無くすことができた。

以上のことから、本研究で試作したRETUSSは、トレーサビリティの維持における2つの課題を解決でき、ひいては、ソフトウェアの品質確保の支援ができると考えられる。

今後の課題を以下に示す。

- 同じ属性および同じ操作への対応
- Javaソースコードにおける未対応の構文への対応
- クラス図のクラスの属性および操作の未対応の構文への対応
- クラス図の記述機能の拡張

参考文献

1) SQuBOK 策定部会, “ソフトウェア品質知識体系ガイド 第 2 版”, オーム社, 2014.
2) Object Management Group, 西原 裕善 監訳, “UML2.0 仕様書 2.1 対応”, オーム社, 2006.
3) OMG (Object Management Group, Inc.), “Welcome To UML Web Site!”, http://www.uml.org/ （最終アクセス 2018/02/14）.
4) Dan Pilone, Neil Pitman 著, 原 隆文 訳, “UML2.0 クイックリファレンス”, オライリー・ジャパン, 2008.