UMLとJava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

有馬 薫[1,a)]　片山 徹郎[1,b)]

概要：ソフトウェアの品質確保のための 1 つの方法として，成果物のトレーサビリティの維持が存在する．しかし，成果物のトレーサビリティの維持には，手間と時間がかかることと，人手に起因するミスが発生する可能性があることの 2 つの課題が存在する．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 Java 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 を試作する．RETUSS は，UML ダイアグラムのうち，システムの静的な構造を表すクラス図と，システムの振る舞いを表すシーケンス図を対象とする．RETUSS を使用してトレーサビリティを維持する場合と，人手によりトレーサビリティを維持する場合の時間を比較する実験を行った．実験結果より，RETUSS はクラス図と Java ソースコードの場合に約 62.6%，シーケンス図と Java ソースコードの場合に約 69.4%，トレーサビリティを維持する時間を削減できた．また，RETUSS は，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除去できた．これにより 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る．

キーワード：ソフトウェア品質，トレーサビリティ，UML，Java，ソースコード

RETUSS that Ensures Traceability in Real Time between UML and Java Source Code

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 1 つの方法として，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，他の関連した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
- 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

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 Java 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) [2], [3], [4] を開発する．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．

UML (Unified Modeling Language) [5] 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 [6]．本研究で開発する 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と，システムの振舞いを表すシーケンス図を対象とする．

2. RETUSS

RETUSS の外観を，**図 1** に示す．RETUSS は，UML

[1] 宮崎大学
University of Miyazaki
[a)] arima@earth.cs.miyazaki-u.ac.jp
[b)] kat@cs.miyazaki-u.ac.jp

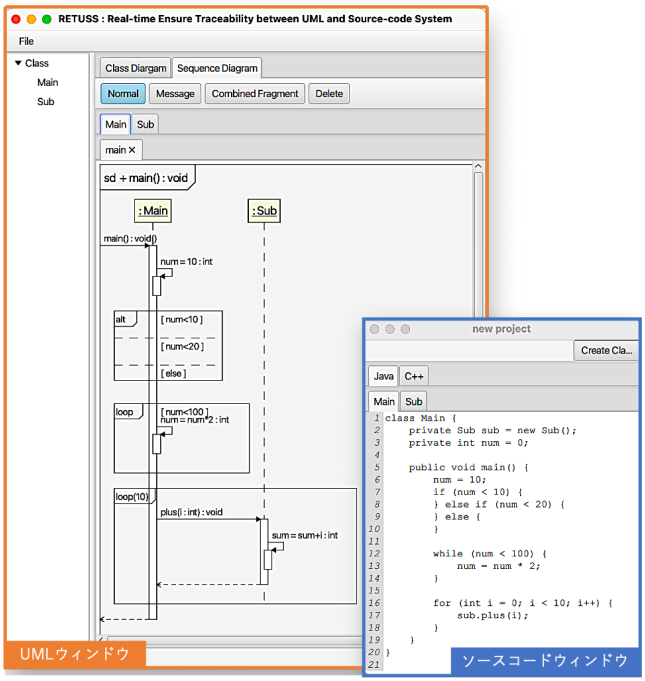

図 1　RETUSS の外観
Fig. 1　Interface of RETUSS.

ウィンドウとソースコードウィンドウを持つ．UML ウィンドウは，UML を記述するためのウィンドウである．ソースコードウィンドウは，ソースコードを記述するためのウィンドウである．

RETUSS が持つ主な機能を，以下に示す．

- UML 記述機能
- ソースコード記述機能
- リアルタイム同期機能

以降，各機能について説明する．

2.1　UML 記述機能

UML 記述機能は，UML ウィンドウにクラス図やシーケンス図を記述する機能である．UML ウィンドウには，クラス図タブとシーケンス図タブがある．クラス図タブ内には，クラス図描画ボタンとクラス図記述エリアが，シーケンス図タブ内には，シーケンス図描画ボタンとシーケンス図記述エリアがある．ユーザは，クラス図描画ボタン，またはシーケンス図描画ボタンを選択することで，GUI 操作によって UML を記述する．RETUSS は，ユーザが記述した UML を，リアルタイムに Java ソースコードへ変換する．さらに，ユーザは，UML 記述機能を用いて，Java ソースコードから変換した UML を編集することもできる．

ユーザは，クラス図タブ内のクラス図描画ボタンを選択することで，クラス図記述エリアのクラス図を編集できる．RETUSS が対応しているクラス図の記述項目を，以下に示す．

- クラスの追加・削除
- クラスの属性の追加・変更・削除
- クラスの操作の追加・変更・削除
- クラス間のコンポジション関係の追加・削除
- クラス間の汎化関係の追加・削除

ユーザは，UML ウィンドウのシーケンス図タブ内で，シーケンス図を記述できる．シーケンス図タブ内には，クラスタブがあり，各クラスタブ内には，各クラスが持つ操作ごとに操作タブがある．各操作タブ内には，1 つの操作を表すシーケンス図を記述する．そのため，ユーザは操作を選択して，選択した操作のシーケンス図を表示および編集できる．RETUSS が対応しているシーケンス図の記述項目を，以下に示す．

- メッセージの追加・削除
- 複合フラグメントの追加・削除

ただし，追加できるメッセージは，操作呼び出しを表すメッセージのみである．また，追加できる複合フラグメントは，opt，alt，loop の 3 種類のみである．

2.2　ソースコード記述機能

ソースコード記述機能は，ソースコードウィンドウに Java ソースコードを記述する機能である．ソースコードウィンドウには，言語タブとクラス作成ボタンがある．各言語タブ内には，クラスタブがある．各クラスタブ内には，テキストエディタがあり，ユーザは，クラスごとにソースコードを表示および編集できる．また，ユーザは，クラス作成ボタンを使用して，選択している言語タブ内に，クラスを追加できる．

2.3　リアルタイム同期機能

リアルタイム同期機能は，ユーザが RETUSS 上に記述した UML と Java 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機能である．ユーザが UML を変更した場合は，変更後の UML を Java ソースコードへリアルタイムに変換する．同様に，ユーザが Java ソースコードを変更した場合は，変更後の Java ソースコードを UML へリアルタイムに変換する．

リアルタイム同期機能の適用範囲は，RETUSS が対応している UML の要素と，Java ソースコードの構文のみである．なお，RETUSS が対応していない UML の要素や，Java ソースコードの構文は，変換する際に無視し，RETUSS 上でその情報を保持しない．

3.　リアルタイム同期機能のアプローチ

リアルタイム同期機能では，UML と Java ソースコードを対応関係に従ってリアルタイムに変換する．本章では，リアルタイム同期機能のために定義した，クラス図と Java ソースコードの対応関係，およびシーケンス図と Java ソースコードの対応関係を説明する．さらに，これらの対応関係を用いた，リアルタイム同期機能の流れを説明する．

3.1 クラス図と Java ソースコードの対応関係

クラス図と Java ソースコードの対応関係を，表 1 に示す．以降，クラス図と Java ソースコードの対応関係について詳細を述べる．

- クラス図のクラス名と Java ソースコードのクラス宣言
- クラス図の可視性と Java ソースコードのアクセス修飾子
  - 可視性+とアクセス修飾子 public
  - 可視性-とアクセス修飾子 private
  - 可視性# とアクセス修飾子 protected
    アクセス修飾子 protected は，可視性#と~を組み合わせたアクセス修飾子であり，厳密には異なるが，本研究では#と同等の可視性として扱う．
  - 可視性~とアクセス修飾子なし
- クラス図の属性と Java ソースコードのフィールド宣言
  - 属性の可視性とフィールド宣言のアクセス修飾子
  - 属性の属性名とフィールドのフィールド名
  - 属性の型名とフィールドの型名
- クラス図の操作と Java ソースコードのメソッド宣言
  - 操作の可視性とメソッドのアクセス修飾子
  - 操作の操作名とメソッド名
  - 操作のパラメータ名とメソッドの仮引数名
  - 操作のパラメータの型とメソッドの仮引数の型
  - 操作の戻り値の型とメソッドの戻り値の型
- クラス図のコンポジション関係と Java ソースコードのフィールド宣言
  - コンポジション先のクラス名とフィールド宣言のクラス名
  - コンポジション関係の関連端名とフィールド宣言のフィールド名
  - コンポジション先のクラス名とフィールド宣言のコンストラクタ名
- クラス図の汎化関係と Java ソースコードの継承クラス宣言
  - 汎化関係の親クラス名と継承クラス宣言の親クラス名
  - 汎化関係の子クラス名と継承クラス宣言のクラス名
- クラス図の抽象クラスと Java ソースコードの抽象クラス宣言
  対応する要素はクラス名と同様である．
- クラス図の抽象操作と Java ソースコードの抽象メソッド宣言
  対応する要素はメソッド宣言と同様である．

3.2 シーケンス図と Java ソースコードの対応関係

シーケンス図と Java ソースコードの対応関係を，表 2 に示す．以降，シーケンス図と Java ソースコードの対応関係について詳細を述べる．

- シーケンス図のライフラインと Java ソースコードのクラス宣言
  - ライフラインのクラス名とクラス宣言のクラス名
- シーケンス図のローカル変数定義を表すメッセージと Java ソースコードのローカル変数定義文
  - メッセージシグニチャの左辺値とローカル変数定義文の変数名
  - メッセージシグニチャの右辺値とローカル変数定義文の既定値
  - メッセージシグニチャの戻り値の型とローカル変数定義文の型名
- シーケンス図の代入文を表すメッセージと Java ソースコードの代入文
  - メッセージシグニチャの左辺値と代入文の変数名
  - メッセージシグニチャの右辺値と代入文の代入値
- シーケンス図の操作呼び出しを表すメッセージとメソッド呼び出し式およびメソッド宣言
  - メッセージシグニチャのメッセージ名とメソッド呼び出し式のメソッド名
  - メッセージシグニチャの引数とメソッド呼び出し式の実引数
  - メッセージシグニチャの引数の型とメソッド宣言の仮引数の型
  - メッセージシグニチャの戻り値の型とメソッド宣言の戻り値の型
- シーケンス図の複合フラグメント opt と Java ソースコードの if-then 文
  - 複合フラグメント opt のガード条件と if-then 文の条件式
- シーケンス図の複合フラグメント alt と Java ソースコードの if-then-else 文
  - 複合フラグメント alt のガード条件と if-then-else 文の条件式
- シーケンス図の複合フラグメント loop と Java ソースコードの while 文または for 文
  複合フラグメント loop がガード条件を持つ場合は，Java ソースコードの while 文と対応する．複合フラグメント loop が繰り返し回数を持つ場合は，Java ソースコードの for 文と対応する．ただし，現時点の RETUSS は，表 2 に示す形式以外の for 文には対応していない．
  - 複合フラグメント loop のガード条件と while 文の条件式
  - 複合フラグメント loop の反復回数と for 文の X および Y から計算した反復回数

ただし，現在の RETUSS は，複合フラグメントのネストには対応していない．

表 1　クラス図と Java ソースコードの対応関係
Table 1　Correspondence between class diagram and Java source code.

クラス図での名称	クラス図での表記	Java での名称	Java での構文
クラス名	クラス名	クラス宣言	class クラス名 { }
可視性	+ - # ~	アクセス修飾子	public private protected なし
属性	可視性 属性名 : 型名	フィールド宣言	アクセス修飾子 型名 フィールド名;
操作	可視性 操作名 (パラメータ名:パラメータの型):戻り値の型	メソッド宣言	アクセス修飾子 戻り値の型 メソッド名 (仮引数の型 仮引数名){}
コンポジション関係	コンポジション先のクラス名，関連端名	フィールドとして宣言	クラス名 フィールド名 = new コンストラクタ名 ();
汎化関係	親クラス名，子クラス名	継承クラス宣言	class クラス名 extends 親クラス名 {}
抽象クラス	抽象クラス名	抽象クラス宣言	abstract クラス宣言
抽象操作	可視性 操作名 (パラメータ名:パラメータの型):戻り値の型	抽象メソッド宣言	abstract アクセス修飾子 戻り値の型 名前 (仮引数の型 仮引数名)

表 2　シーケンス図と Java ソースコードの対応関係
Table 2　Correspondence between sequence diagram and Java source code.

シーケンス図での名称	シーケンス図での表記	Java での名称	Java での構文
ライフライン	: クラス名	クラス宣言	class クラス名 { }
ローカル変数定義文を表すメッセージ	左辺値 = 右辺値:戻り値の型	ローカル変数定義文	型名 変数名 = 既定値;
代入文を表すメッセージ	左辺値 = 右辺値:戻り値の型	代入文	変数名 = 代入値;
操作呼び出しを表すメッセージ	メッセージ名 (引数:引数の型, …):戻り値の型	メソッド呼び出し式 メソッド宣言	メソッド名 (実引数,…); アクセス修飾子 戻り値の型 メソッド名 (仮引数の型 仮引数) { … }

複合フラグメント opt	[ガード条件]	if-then 文	if(条件式) {
			　条件式が真の場合に実行する文
			}

複合フラグメント alt	[ガード条件 1] [ガード条件 2] …	if-then-else 文	if(条件式 1) {
			　条件式 1 が真の場合に実行する文
			} else if (条件式 2) {
			　条件式 2 が真の場合に実行する文
			} …
			} else {
			　すべての条件式が偽の場合に実行する文
			}

複合フラグメント loop	[ガード条件]	while 文	while(条件式) {
			　条件式が真の間，実行する文
			}

	loop(反復回数)	for 文	for(int i=X; i < Y; i++){ 繰り返し実行する文 }
			for(int i=X; i <= Y; i++){ 繰り返し実行する文 }
			for(int i=X; i > Y; i−−){ 繰り返し実行する文 }
			for(int i=X; i >= Y; i−−){ 繰り返し実行する文 }

3.3　UML の変更を Java ソースコードに同期する流れ

リアルタイム同期機能のデータの流れを，**図 2** に示す．図中の UML 情報とは，表 1 で示したクラス図の要素と，表 2 で示したシーケンス図の要素を保持するための，内部データ構造である．同様に，Java 情報とは，表 1 と表 2 で示した Java ソースコードの構文を保持するための，内部データ構造である．

ユーザが UML 記述機能を用いて UML を変更した際に，UML の変更を Java ソースコードに同期する流れを，以下に示す．

(1) UML の記述内容を識別する．
(2) UML の記述内容を基に，UML 情報を更新する．
(3) UML 情報を Java 情報に変換する．
(4) Java 情報から Java ソースコードを生成する．

以降，各処理の詳細を説明する．

3.3.1　UML の記述内容の識別

RETUSS は，ユーザの GUI 操作から，ユーザが行った記述項目と，変更対象クラスを識別する．記述項目は，2.1 節で述べたクラス図の記述項目と，シーケンス図の記述項目の中から識別する．ユーザがクラス図を変更する場合は，UML の記述時にユーザが選択したクラス図描画ボタンの種類に応じて，変更の種類を識別する．さらに，ユー

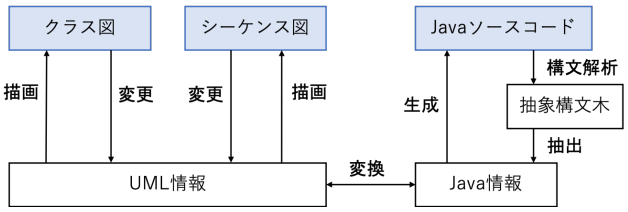

図 2　リアルタイム同期機能のデータの流れ
Fig. 2　The data flow of the real time synchronazation function.

ザがマウスで選択したクラスを，変更対象クラスとする．ユーザがシーケンス図を変更する場合は，UML の記述時にユーザが選択したシーケンス図描画ボタンの種類に応じて，変更の種類を識別する．さらに，ユーザが変更した操作が属するクラスを，変更対象クラスとする．

3.3.2　UML 情報の更新

RETUSS は，ユーザが行った記述項目と変更対象クラスを基に，UML 情報を更新する．UML 情報の更新は，各記述項目に対応したイベントハンドラが行う．まず，変更前の UML 情報が持つクラスリストから，クラス名が変更対象クラスと同一のクラスに関する情報（以降，これをクラス情報と呼ぶ）を探索する．変更対象クラスは，ユーザが GUI 操作で指定するため，変更前の UML 情報に必ず含まれる．

次に，記述項目に応じて，変更対象のクラス情報を変更する．例えば，記述項目が「クラスの属性の追加」である場合，変更対象のクラス情報が持つ属性リストに，新たな属性を追加する．

3.3.3　UML 情報から Java 情報への変換

RETUSS は，更新後の UML 情報を Java 情報に変換する．まず，UML 情報が持つ変更対象クラスのクラス情報を，Java 情報のクラス情報に変換する．このとき，表 1 および表 2 の対応関係に従って変換する．次に，Java 情報が持つクラスリストから，クラス名が変更対象クラスと同じクラス情報を探索する．クラス名が同じクラス情報を，UML 情報から変換したクラス情報に置き換える．

3.3.4　Java 情報から Java ソースコードの生成

RETUSS は，Java 情報から Java ソースコードを生成する．表 1 および表 2 で示した Java での構文に対して，Java 情報から抽出したクラス名やフィールド名などの情報を組み合わせることで，Java 情報から Java ソースコードを生成する．例えば，Java 情報に，クラス名が Person のクラスがあり，アクセス修飾子が private，フィールド名が name，型名が String のフィールドを持つ場合は，Java のクラス宣言文にクラス名を，フィールド宣言文にアクセス修飾子，フィールド名，型名をそれぞれ組み合わせて，Java ソースコード「class Person { private String name; }」を生成する．

3.4　Java ソースコードの変更を UML に同期する流れ

ユーザがソースコード記述機能を用いて Java ソースコードを変更した際に，Java ソースコードの変更を UML に同期する流れを，以下に示す．

(1) Java ソースコードの記述内容を識別する．
(2) Java ソースコードの記述内容を基に，Java 情報を更新する．
(3) Java 情報を UML 情報に変換する．
(4) UML 情報から，クラス図またはシーケンス図を描画する．

以降，各処理の詳細を説明する．

3.4.1　Java ソースコードの記述内容の識別

RETUSS は，ユーザが変更したクラスを識別し，変更対象クラスの Java ソースコードを構文解析する．ユーザがソースコードウィンドウのテキストエディタで変更したクラスを，変更対象クラスとする．また，Java ソースコードの構文解析には，ANTLR (ANother Tool for Language Recognition) [7] で生成した構文解析器を使用する．

3.4.2　Java 情報の更新

RETUSS は，変更対象クラスの抽象構文木から，RETUSS が対応している情報を抽出し，Java 情報を更新する．まず，変更対象クラスの抽象構文木から，RETUSS が対応している情報を抽出し，クラス情報を作成する．ユーザが新たなクラスを追加する場合は，Java 情報が持つクラスリストに，新たに作成したクラス情報を追加する．ユーザが既存のクラスを変更する場合は，Java 情報が持つクラスリストから，変更対象クラスと同一名のクラス情報を探索し，新たに作成したクラス情報で置き換える．ユーザが既存のクラスを削除する場合は，Java 情報が持つクラスリストから，変更対象クラスと同一名のクラス情報を探索し，削除する．

3.4.3　Java 情報から UML 情報への変換

RETUSS は，更新後の Java 情報を UML 情報に変換する．まず，Java 情報が持つ変更対象クラスのクラス情報を，UML 情報のクラス情報に変換する．このとき，表 1 および表 2 の対応関係に従って変換する．次に，UML 情報が持つクラスリストから，クラス名が変更対象のクラスと同じクラス情報を探索する．クラス名が同じクラス情報を，Java 情報から変換したクラス情報に置き換える．

3.4.4　UML 情報からクラス図またはシーケンス図の描画

RETUSS は，UML 情報から，クラス図またはシーケンス図を描画する．ユーザが UML ウィンドウでクラス図タブを選択している場合は，クラス図を描画し，シーケンス図タブを選択している場合は，シーケンス図を描画する．RETUSS は，クラス図およびシーケンス図に含まれる各要素の描画テンプレートを持つ．各要素の描画テンプレートに対して，UML 情報から抽出したクラス名や属性名などの情報を組み合わせることで，UML 情報からクラス図ま

たはシーケンス図を描画する．

4. 適用例

本研究で開発した RETUSS を使用して，UML と Java 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できることを示す．

クラス図と Java ソースコードの適用例を，**図 3** に示す．図 3 は，RETUSS に対して Java ソースコードを入力した直後の画面である．図 3 より，クラス図と Java ソースコードの対応関係（表 1）に基づき，Text クラスのトレーサビリティを維持できていることがわかる．同様に，Super クラス，Sub クラス，Point クラスもトレーサビリティを維持できることを確認した．よって，RETUSS は，クラス図と Java 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できる．

シーケンス図と Java ソースコードの適用例を，**図 4** に示す．図 4 は，RETUSS に対して Java ソースコードを入力した直後の画面である．図 4 より，シーケンス図と Java ソースコードの対応関係（表 2）に基づき，Main クラスのトレーサビリティを維持できていることがわかる．同様に，Sub クラスもトレーサビリティを維持できることを確認した．よって，RETUSS は，シーケンス図と Java 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できる．

5. 評価

本研究で開発した RETUSS の有用性を評価するために，UML と Java ソースコード間のトレーサビリティを維持する時間を計測する実験を行う．実験では，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する場合と，シーケンス図と Java ソースコード間のトレーサビリティを維持する場合について，それぞれ時間を計測する．被験者がトレーサビリティを維持する方法として，RETUSS を使用して自動でトレーサビリティを維持する Case A と，UML モデリングツールとテキストエディタの Visual Studio Code[8] を使用して手動でトレーサビリティを維持する Case B の 2 つを用意し，Case A と Case B の時間を比較する．Case B で使用する UML モデリングツールは，クラス図の場合は astah*[9] を使用し，シーケンス図の場合は Enterprise Architect[10] を使用する．

5.1 被験者

被験者は，宮崎大学で情報工学を専攻する 6 人の学生であり，普段からソースコードの読み書きを行い，基本的なプログラミングの知識がある．本実験で使用するクラス図とシーケンス図および Java に関する知識がない者も含まれるが，今回の実験に必要な知識は事前に説明する．

表 3 UML 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）

Table 3 Time of ensuring traceability between UML and Java source codes (sec).

実験回数	クラス図		シーケンス図	
	Case A	Case B	Case A	Case B
1	115	226	134	369
2	75	179	95	346
3	90	320	102	361
平均	93	242	110	359

5.2 実験方法

3 名の被験者は，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する実験を行い，残りの 3 名はシーケンス図と Java ソースコード間のトレーサビリティを維持する実験を行う．実験の手順を，以下に示す．

(1) 実験者は，トレーサビリティを維持した状態のクラス図またはシーケンス図と，Java ソースコード (以降，成果物と呼ぶ) を被験者に提示する．
(2) 実験者は，被験者に対して，成果物の変更を指示し，成果物の変更に要する時間の計測を開始する．
(3) 被験者は，成果物を変更する．被験者は，変更が完了した時点で実験者に報告する．
(4) 実験者は，変更時間の計測を一時停止し，変更後の成果物を確認する．変更後の成果物にミスがない場合は，変更時間の計測を終了する．変更後の成果物にミスがある場合は，被験者にミスがあることを伝え，変更時間の計測を再開する．このとき，ミスの内容は被験者に伝えない．被験者は手順 (3) に戻り，成果物を再度変更する．

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する実験では，変更前の成果物として，**図 5** を使用する．変更指示として，Sub クラスとコンポジション関係にある Text クラスの追加を指示する．変更後のクラス図と Java ソースコードを，図 3 とする．また，シーケンス図と Java ソースコード間のトレーサビリティを維持する実験では，変更前の成果物として，**図 6** を使用する．変更指示として，Main クラスの main メソッドに対して，3 つの複合フラグメントの追加と，2 つのメッセージの追加を指示する．変更後のシーケンス図と Java ソースコードを，図 4 とする．

なお，本実験で使用する成果物は，RETUSS が対応している要素および構文のみを含む，クラス図，シーケンス図，Java ソースコードを用いる．

5.3 実験結果と考察

実験結果を，表 3 に示す．表 3 より，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する時間は，Case A が Case B よりも約 62.6%短い．また，シーケンス

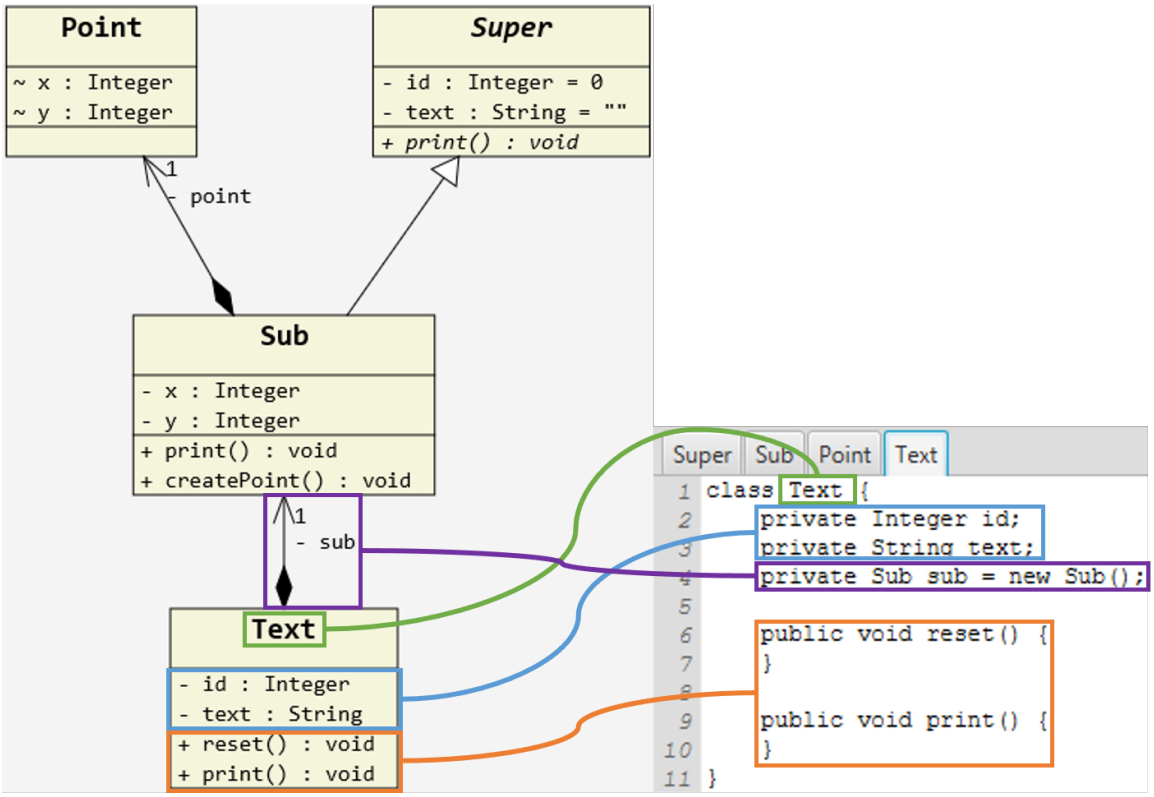

図 3　クラス図と Java ソースコードの適用例

Fig. 3　An application example of class diagram and Java source code.

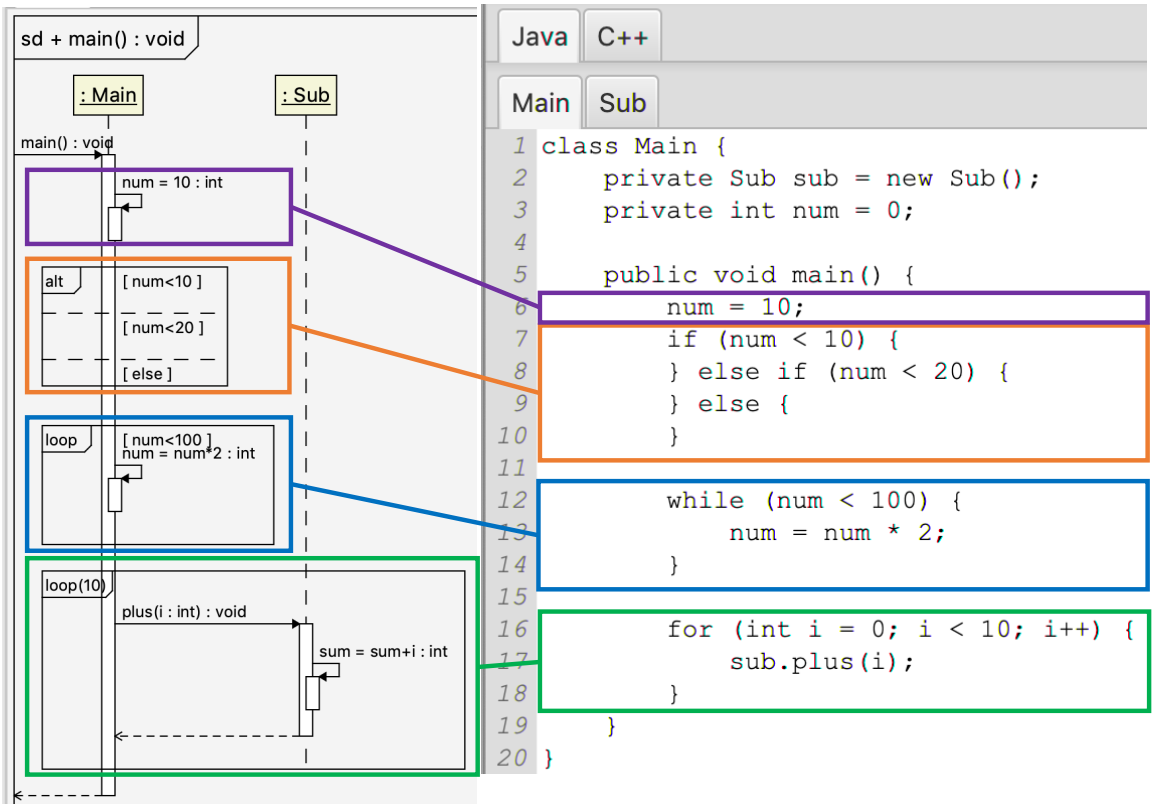

図 4　シーケンス図と Java ソースコードの適用例

Fig. 4　An application example of sequence diagram and Java source code.

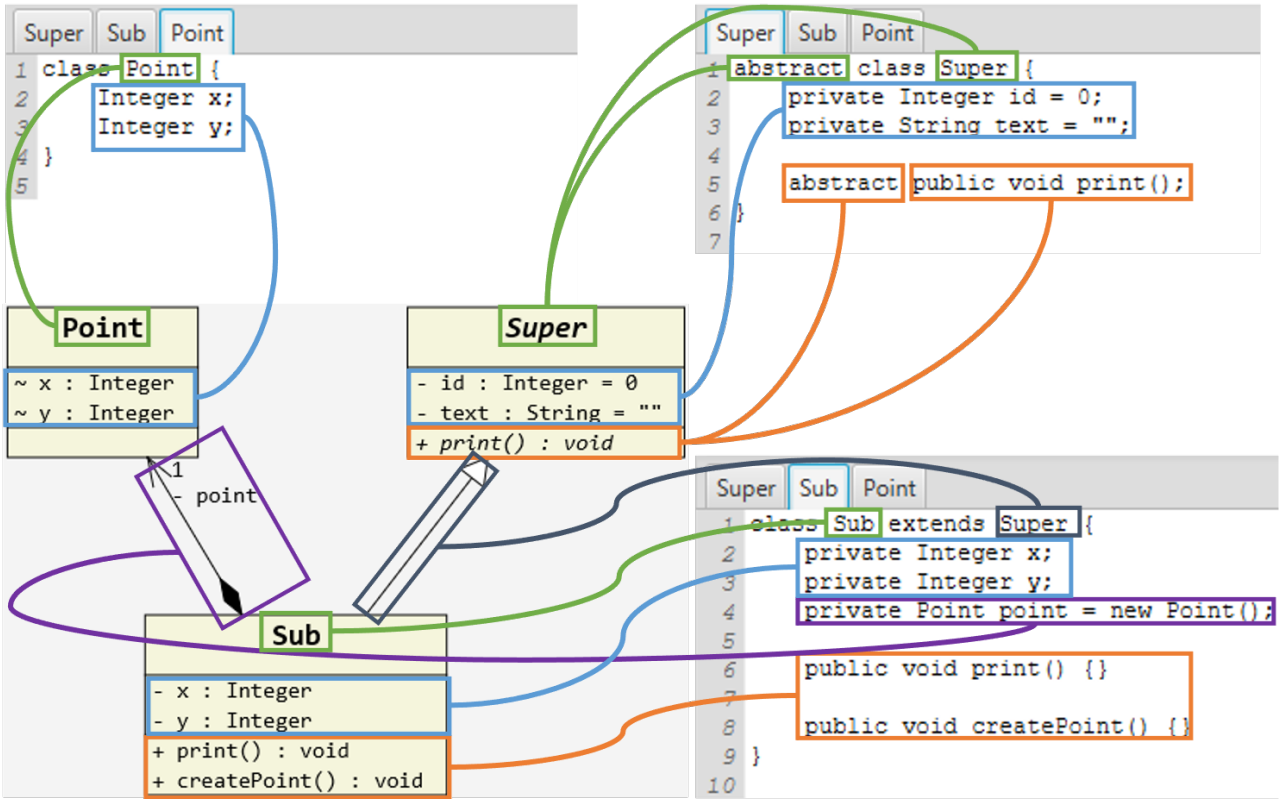

図 5　変更前のクラス図と Java ソースコード

Fig. 5　Class diagram and Java source codes before the change.

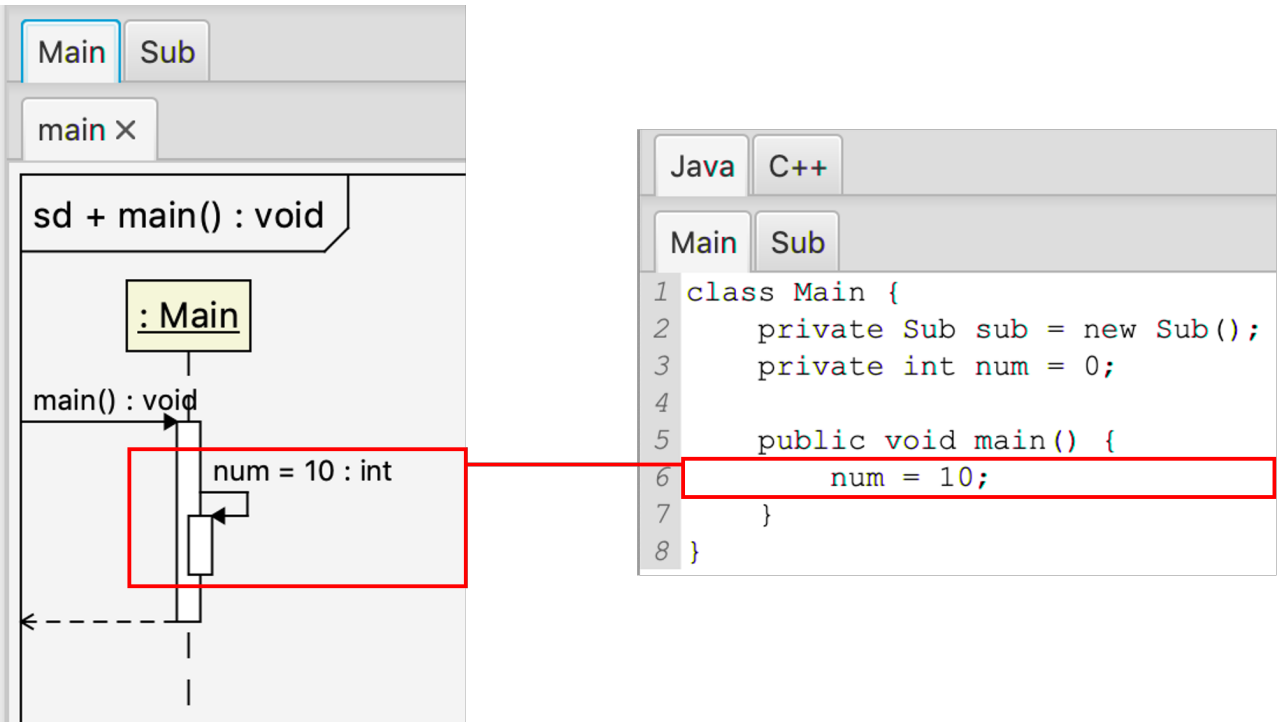

図 6　変更前のシーケンス図と Java ソースコード

Fig. 6　Sequence diagram and Java source code brefore the change.

図と Java ソースコード間のトレーサビリティを維持する時間は，Case A が Case B よりも約 69.4%短い．さらに，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する場合に，被験者が変更完了を報告した時点で，Case B では以下のミスが発生した．

- 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
- クラスの操作名と，Java ソースコードのメソッド名の不一致
- クラスの操作の戻り値の型と，Java ソースコードのメソッド宣言の戻り値の型の不一致

一方で，RETUSS を使用する Case 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．

以上の実験結果から，RETUSS を使用することによって，UML と Java ソースコード間のトレーサビリティを維持する際の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と考える．

6. 関連研究

UML とソースコードを変換する研究は，数多く行われている [11], [12], [13], [14]. また，UML からソースコードへの変換，またはソースコードから UML への変換が可能な既存ツールは数多く存在する [9], [10], [14], [15], [16]．しかし，これらの研究や既存ツールは，UML の構造図と振舞い図の両方を用いて，UML とソースコード間のトレーサビリティを維持することに焦点を当てていない．RETUSS の利点は，UML の構造図と振舞い図の両方に対して，リアルタイムに，ソースコードとのトレーサビリティを維持できることである．これによって，RETUSS は構造図には含まれない操作の内容も，UML とトレーサビリティを維持できる．以降，既存ツールの中から，Enterprise Architect[10] と，Maintenace Visualization and Synchronazation Tool[15] を取り上げ，RETUSS と比較する．

6.1 Enterprise Architect との比較

UML とソースコード間の変換が可能な既存ツールとして，Sparx Systems 社が開発した Enterprise Architect (EA) [10] がある．EA は，ソースコードをシーケンス図に変換する際は，対象とするプログラムを動的解析し，実行結果をシーケンス図に変換する．そのため，ソースコードの変更をシーケンス図に同期できず，メソッド内部の情報は UML とトレーサビリティを維持できない．また，動的解析で変換したシーケンス図には，条件分岐や繰り返し処理が表現されない．一方で RETUSS は，クラス図とシーケンス図の両方において，Java ソースコードとリアルタイムに同期できるため，メソッド内部の情報も UML とトレーサビリティを維持できる．また，条件分岐や繰り返し処理は複合フラグメントを用いて表現する．よって，RETUSS は特に，メソッド内部の情報も UML とトレーサビリティを維持する際に有用である．

6.2 Maintenace Visualization and Synchronazation Tool との比較

UML とソースコードを同期するツールとして，Ahid Yassen らが開発した Maintenace Visualization and Synchronazation Tool (MVST)[15] がある．MVST は，オブジェクト指向プログラムをクラス図とオブジェクト図，シーケンス図で可視化し，クラス図とソースコードを同期する．しかしながら，MVST はシーケンス図とソースコードの同期には対応せず，シーケンス図とソースコード間のトレーサビリティの維持には手間と時間がかかる．一方で RETUSS は，クラス図とシーケンス図の両方において，Java ソースコードとリアルタイムに同期できる．よって，RETUSS は特に，シーケンス図と Java ソースコード間のトレーサビリティの維持において有用である．

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 Java 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 を開発した．

RETUSS は，UML 記述機能，ソースコード記述機能，リアルタイム同期機能を持つ．リアルタイム同期機能は，ユーザが RETUSS 上に記述した UML と Java 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機能である．リアルタイム同期機能を実現するために，本研究ではクラス図と Java ソースコードの対応関係と，シーケンス図と Java ソースコードの対応関係を定義した．RETUSS は，これらの対応関係に従って，UML と Java ソースコードをリアルタイムに変換することによって，UML と Java 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．

RETUSS の有用性を評価するために，宮崎大学の学生 6 人を対象とした実験を行った．実験では，クラス図と Java ソースコードのトレーサビリティを維持する場合と，シーケンス図と Java ソースコードのトレーサビリティを維持する場合に，RETUSS を使用する場合と，人手の場合で，トレーサビリティの維持に要する時間を比較した．実験の結果，RETUSS は，クラス図と Java ソースコードのトレーサビリティを維持する場合に約 62.6%，シーケンス図と Java ソースコードのトレーサビリティを維持する場合に約 69.4%の時間を削減できた．また，RETUSS は，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除去できた．よって，本研究で開発した 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る．

以下に，RETUSS の今後の課題を示す．

- UML の未対応要素への対応
  現在の RETUSS は，未対応のクラス図要素とシーケンス図要素が存在する．例えば，クラス図の多重度やインターフェース，シーケンス図の複合フラグメント break や相互作用オカレンスなどに未対応である．このため，現在の RETUSS では，Java ソースコードと同期できるクラス図およびシーケンス図は制限されている．未対応要素に対応することによって，より大規模なクラス図やシーケンス図を，RETUSS に適用できるようになると考える．
- Java の未対応構文への対応
  現在の RETUSS は，未対応の Java の構文が存在する．例えば，クラスインスタンス作成式や return 文，switch-case 文などに未対応である．また，現在の RETUSS では，未対応構文を無視して UML に変換するため，RETUSS 上でその情報を保持しない．このため，現在の RETUSS では，UML と同期できる Java ソースコードは制限されている．未対応構文に対応することによって，より大規模な Java ソースコードを，RETUSS に適用できるようになると考える．
- クラス図とシーケンス図以外の UML ダイアグラムへの対応
  現在の RETUSS は，クラス図とシーケンス図以外の UML ダイアグラムに対応していない．例えば，パッケージ図やステートマシン図，アクティビティ図などに未対応である．UML は複数のダイアグラムを利用することによって，1 つのシステムを多面的に表現できる．そのため，より多くの UML ダイアグラムに対応することで，RETUSS の有用性は向上すると考える．
- Java 以外のプログラミング言語への対応
  現在の RETUSS は，Java 以外のプログラミング言語に対応していない．例えば，C++や Python，JavaScript などに未対応である．UML は特定のプログラミング言語に依存しない汎用モデリング言語である．そのため，より多くのプログラミング言語に対応することで，RETUSS の有用性は向上すると考える．

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] Katayama, T., Mori, K., Kita, Y., et al.: RETUSS: Ensuring Traceability System between Class Diagram in UML and Java Source Code in Real Time, *Journal of Robotics, Networking and Artificial Life*, Vol.5(2), pp.114-117 (2018).
[3] Mori, K., Katayama, T., Kita, Y., et al.: Implementation of RETUSS to Ensure Traceability between Class Diagram in UML and Java Source Code in Real Time, *Proc. The 2018 International Conference on Artificial Life and Robotics (ICAROB2018)*, pp.522-525 (2018).
[4] Morichan: RETUSS: Real-time Ensure Traceability between UML and Source-code System, GitHub (online), 入手先 ⟨https://github.com/Morichan/Retuss⟩ (2021.07.28).
[5] Object Management Group: About the Unified Modeling Language Specification Version 2.5.1 (online), 入手先 ⟨https://www.omg.org/spec/UML/⟩ (2021.07.28).
[6] Object Management Group: WHAT IS UML, UNIFIED MODELING LANGUAGE (online), 入手先 ⟨https://www.uml.org/what-is-uml.htm⟩ (2021.07.28).
[7] Parr, T.: ANTRL (Online), 入手先 ⟨https://www.antlr.org/⟩ (2021.07.28).
[8] Microsoft: Visual Studio Code (online), 入手先 ⟨https://code.visualstudio.com/⟩ (2021.07.28).
[9] チェンジビジョン: astah システム設計、ソフトウェア開発支援ツール, astah (online), 入手先 ⟨https://astah.change-vision.com/ja/⟩ (2021.07.28).
[10] スパークスシステムズジャパン: Enterprise Architect - UML/SysML/BPMN モデリングツール (online), 入手先 ⟨https://www.sparxsystems.jp/⟩ (2021.07.28).
[11] Decker, J.M., Swartz, K., Collard, L.K., et al.: A Tool for Efficiently Reverse Engineering Accurate UML Class Diagrams, *Proc. 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp.607-609 (2016).
[12] Thongmak, M. and Muenchaisri, P.: Design of rules for transforming UML sequence diagrams into Java code, *Proc. Ninth Asia-Pacific Software Engineering Conference*, pp.485-494 (2002).
[13] Cheers, H. and Lin, Y.: Reverse Engineering UML Sequence Diagrams for Program Comprehension Activities, *Proc. 2020 5th International Conference on Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA)*, pp.1-10 (2020).
[14] Kolling M., Quig, B., Patterson, A., et al. : The BlueJ system and its pedagogy, *Computer Science Education*, Vol.13, No.4, pp.208–213 (2014).
[15] Yaseen, A. and Fawareh, H.: Visualization and Synchronization of Object-oriented Programs using Re-engineering Approach, *International Journal of Engineering Research and Technology*, Vol. 12, pp.1239-1246, (2019).
[16] Visual Paradigm: Ideal Modeling & Diagramming Tool for Agile Team Collaboration (online), 入手先 ⟨https://www.visual-paradigm.com/⟩ (2021.07.28).