東海大学紀要情報通信学部
vol.6,No2,2013,pp.37-44

論文

統合開発環境で生成された ユーザインターフェースイベントハンドラの統合手法

谷川 郁太[*1]，渡辺 晴美[*2]

A Unification Method of User Interface Event Handlers to Enhance Maintainability on IDE

by

Ikuta TANIGAWA[*1] and Harumi WATANABE[*2]

(received on September 30, 2013 & accepted on February 7, 2014)

Abstract

Recently, evolution and complexity of user interface cause maintenance problems of the event driven behavior. In developing system by using *Form* or *UserControl* on IDE, we often create many similar event handlers, which may give rise to the problem of redundancy. To overcome this problem, the paper proposes a method for unifying event handlers. By detecting and unifying similar event handlers, the method can solve the problem of the redundant source code and we can acquire higher maintainability. Moreover, our method can keep the behavior after the unifying. Finally, to evaluate the method, we will apply it to a cloud and embedded system “Buta-twii” which is a cooperating charity pot with SNS.

Keywords: *Refactoring, Event Handler, C#, IDE*
キーワード:リファクタリング、イベントハンドラ、C#、統合開発環境

1. はじめに

ソフトウェアの保守は，一般的に冗長な箇所が増えるほど困難になっていくことが知られている．イベント駆動型の振る舞いは，組込みシステムをはじめ多くのソフトウェアに古くから存在しているが，近年ユーザインタフェースの発展・複雑化に伴い，イベント駆動型の振る舞いに関する保守について問題視されるようになってきた[1]．統合開発環境によってFormやUserControlにイベントハンドラを追加していくうちに，類似したイベントハンドラが多くできてしまうことで，ソースコードが冗長となってしまう問題がある．これは，統合開発環境を用いた画面設計において，本来なら既存のイベントハンドラを用いることが可能な場面で新たなイベントハンドラを追加してしまうためである．類似したイベントハンドラが多く作られることは，コードクローンの増加であると言える．コードクローンが保守に弊害をもたらすことは知られている[2][3]．類似した箇所を取り除く方法として，Extract MethodやPull Up Methodといったリファクタリング技術がある[4][5]．しかし，類似したイベントハンドラは，Fig.1に示すとおりExtract Methodにより類似した箇所をまとめても，イベントハンドラ自体は残る．これは，従来のExtract Methodではイベントハンドラ受け渡し箇所について考慮されておらず，イベントハンドラをそのまま一つにまとめることができないためである．従って，イベントハンドラ受け渡しについても考慮した新たなリファクタリング手法が必要となる．

さらに，画面設計においてコントロールごとに使用するリテラル，参照型変数のみが異なるイベントハンドラができるケースが多いことも考慮する必要がある．例として，Fig.2に示す電卓を上げる．ここではボタンごとに異なる数値を入力するイベントハンドラがあり，それぞれ部分的なリテラル以外は同じである．このような場合にも対応することで，より多くの類似したイベントハンドラを除去することができる．

我々は以前に，上記の問題を解決するための類似したイベントハンドラを統合するリファクタリング手法・ツールを開発し展示した[6]．本手法ではソースコード中の類似したイベントハンドラを抽出し，それらを統合することでソースコードの冗長化を解消する．

本論文では，提案手法の適用前後の変化と，それにより外部の振る舞いが変わらないことを示す．

以下，2章では類似したイベントハンドラ統合のために，コードクローンについての関連研究を紹介する．3章では統合対象となるイベントハンドラの条件について述べる．4章では類似したイベントハンドラを統合するための手法を記す．5章では本手法を適用後に外部の振る舞いが変わらないことを示す．6章では適用例となる募金箱システムについて，7章で適用結果を表す．8章で考察と今後の課題を述べる．

*1 情報通信学研究科情報通信学専攻 修士課程
Graduate School of Information and Telecommunication Engineering, Course of Information and Telecommunication Engineering, Master’s Program
*2 情報通信学部組込みソフトウェア工学科 教授
School of Information and Telecommunication Engineering, Department of Embedded Technology, Professor

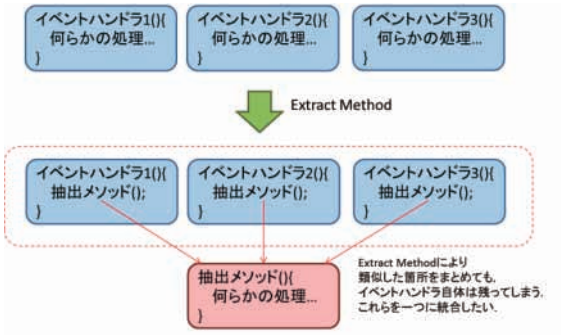

Fig.1 A problem of applying “Extract method”

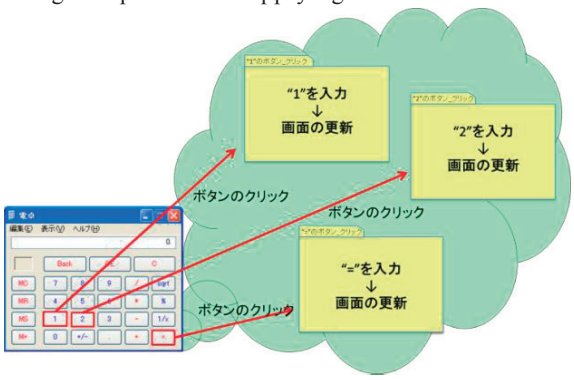

Fig.2 An example of buttons on a pocket calculator

2. 関連研究

類似したイベントハンドラによる冗長化を解消する方法と関連した研究に，コードクローンおよびプログラム依存グラフがある．コードクローンとは，ソースコード中に存在する同一，または類似したコード片のことである．これまで，コードクローンを自動検出するために様々な手法が提案されてきた[7-11]．さらに，検出されたコードクローンに対して，Extract Method や Pull Up Method といったリファクタリングを用いることによりコードクローンを除去する手法もいくつか提案されている[12-16]．

類似したイベントハンドラの統合においても，類似したイベントハンドラの検出が必要となるため，コードクローンの検出手法が参考となる．我々が以前に展示した類似したイベントハンドラ統合ツール[4]では字句単位のコードクローン検出法[8]を参考にして解析を行っている．字句単位のコードクローン検出の利点は，字句解析を行うことで改行や空行，コメントを取り除くことができる点，識別子や定数を特定の種類の字句を特殊な一つの字句に固定化することで，変数名や関数名が変更されたプログラム断片もコードクローンとして認識できる点である．また，プログラム依存グラフを用いる方法[15-16]では意味的に同じ箇所をクローンとして検出できるが，本論文では実装のコストが少なく済むように，字句解析のみで検出している．

類似したイベントハンドラの統合では，単に一つにまとめるだけでなく，イベントハンドラの受け渡しを行っている箇所も統合後のものを渡すように書き換えなければならない．似たケースとして，コードクローン間で呼び出すメソッドが異なり，それらメソッド同士もコードクローンなので集約可能なことがある．プログラム依存グラフにより，このような関係を検出し，解消するリファクタリング手法が提案されている[15]．また，後述する部分的に異なるイベントハンドラ統合手法は，コードクローンの差分の扱いにおいて，Template Method によるリファクタリング手法[16]に似ている．

上記を実現することを考えると，類似したイベントハンドラの検出や統合の手順は複雑になりがちである．我々は外部の振る舞いが変わらないことを示しやすくするために簡便な手法を提案した．

3. 統合対象となるイベントハンドラ

提案手法では，関数ポインタやインターフェース等を介して他のオブジェクトに渡された関数，メソッドをイベントハンドラとして扱い，統合対象となるイベントハンドラの検出，統合を行う．

対象とする言語は，間接的に関数，メソッドを他のオブジェクトから呼び出すことが可能な，関数ポインタやインターフェース等の仕組みを持った命令型の言語である．大抵の命令型言語では，以下の字句を用いてイベントハンドラを表現できる．

- 識別子
- キーワード
- リテラル
- 演算子、区切り記号

これらの字句は，字句を表現している文字列を属性として持つ．さらに，リテラルは属性として型と値を持ち，識別子は属性として型，スコープレベル，位置の情報を持つ．字句が一致しているかどうかの判定は，上記の属性に基づき行う．リテラル以外の字句では，字句を表現している文字列が同じであれば一致していると見なし，リテラルでは，型と値の属性が一致すれば，別の文字列により表現されていても一致していると見なす．

具体例として，C#におけるイベントハンドラを挙げる．C#では，関数ポインタの一種であるデリゲートを介して，他のオブジェクトにメソッドをイベントハンドラとして受け渡すことができる．イベントハンドラの受け渡しは Fig.3 の拡張 BNF によって表すことができる．<受け渡し文>の<メンバ>がイベントハンドラとして渡されているメソッド名にあたり，受け渡しが行われているクラス内で定義された同名のメソッドがイベントハンドラである．イベントハンドラは Fig.4 の拡張 BNF で定義することができる．

統合対象となるイベントハンドラの判定は，上記で表される各イベントハンドラを対象言語の字句列，すなわち識別子，キーワード，リテラル，演算子，区切り記号の列に分解し，字句ごとに比較することで行う．比較を行った結果，「A) イベントハンドラ名を表す識別子以外の字句が一致している」「B) 上記からリテラル，参照型変数に関する字句が部分的に異なる」のいずれかの条件を満たしたものを統合対象のイベントハンドラとする．本章各節でそれぞれの詳細と例を示す．

```
<受け渡し文> = <イベント>, "+=", ( <メンバ> |
    "new", <デリゲート名>, "(", <メンバ>, ")" ), ";" ;
<イベント> = <メンバ>, ".", <識別子>, { ".", <識別子> } ;
<メンバ> = [ "this", "." ], <識別子> ;
<デリゲート名> = <識別子>, { ".", <識別子> } ;
<識別子> = ("_" | <英字>), {"_" | <英字> | <数字>};
```

Fig.3 Expression of registering event handler

```
<メソッド> = <修飾子>, <型名>, <識別子>,
            <パラメータ>, <ブロック> ;
<修飾子> = { public | private | static | … } ;
<パラメータ> = "(", { <型名>, <識別子> }, ")" ;
<ブロック> = "{", { <ステートメント> }, "}" ;
<ステートメント> = <ブロック> | <変数宣言> | … ;
```

Fig.4 Expression of event handler

3.1 全ての字句が一致するケース

複数のイベントハンドラを比較した結果，イベントハンドラ名を表す識別子以外の全ての字句が一致する場合，比較対象は統合可能な類似したイベントハンドラと判断できる．

上記の条件は字句列が完全に一致するもののみを対象としているため，制御文後の中括弧の有無やローカル変数名の違いなど，表現の方法が異なるものは振る舞いが同じでも統合できない．このような問題は字句解析段階で字句列を操作することにより解決することができる．以下に解消可能な違いと具体的な解決方法を示す．

- 制御文後の中括弧の有無による違いを解消するために，中括弧が省略されていれば制御文直後に”{”字句を挿入し，”;”字句の後に”}”字句を挿入する
- ローカル変数名が違いを解消するために，識別子の属性を調べ，識別子がローカル変数名を表す場合には，識別子の文字列属性を宣言された順番に応じた名前に変更する

全ての字句が一致するイベントハンドラの例として，設定ウィンドウの適用ボタン有効化を行うイベントハンドラを挙げる．これは，設定ウィンドウにおいて，コントロールごとに適用ボタンを有効化するイベントハンドラを用意してしまった状況を想定している．Fig.5 にその様子を示す．このように，全ての字句が一致するイベントハンドラが複数あるのは明らかに冗長であり問題である．

3.2 リテラル，参照型変数が部分的に異なるケース

基本的には 3.1 節の条件を満たす文で構成されており，リテラル，参照型変数が部分的に異なっているようなイベントハンドラも統合対象とする．これらは具体的には次のようなケースである．

- リテラルの属性で型は同じだが，値が異なる
- 識別子の属性より，互いに同じ型の参照型変数であり，イベントハンドラを定義しているクラスのメンバである

Fig.2 で示した電卓の例は部分的にリテラルが異なるケースに当たる．例では，ボタンごとに異なる数値を入力するイベントハンドラができ，入力以外の処理内容は全て同じである．参照型変数が異なる例としては，Fig.6 に示すようなものが挙げられる．この例では，ダイアログから指定したファイルのパスをテキストボックスに表示するボタンが複数あり，ボタンごとにイベントハンドラができている．

このケースのイベントハンドラを統合するためには統合手法の関係上，イベントハンドラの引数にイベント発生元のコントロールを示す参照型変数が必要となる．C#では sender という名前の変数がこれにあたる．上記参照型変数がないものは統合の対象外とする．

リテラル，参照型変数が部分的に異なるイベントハンドラは，本来ならば 1 つにまとめることが可能である．

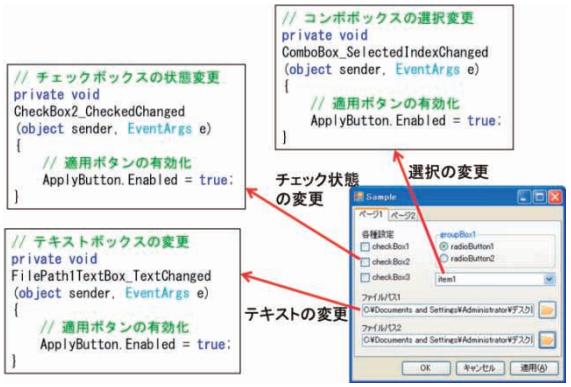

Fig.5 An example of activating buttons

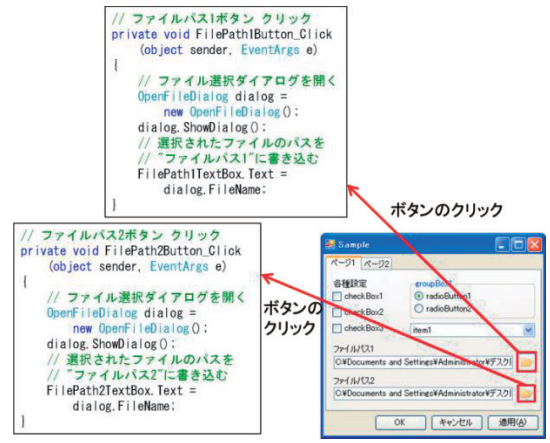

Fig.6 An example of specifying file path

4. 類似したイベントハンドラ統合手法

統合開発環境を用いた開発では，ボタンやテキストボックスなどのウィンドウを構成する部品ごとにイベントハンドラを追加していくことで，類似したイベントハンドラが発生する問題がある．

本論文では，ボタンやテキストボックスなどのウィンドウを構成する部品をコントロールと呼ぶ．また，イベントハンドラを呼び出すオブジェクトにおいて，イベントハンドラを受け取るための変数をイベントと呼ぶ．

上記によって生じた，類似したイベントハンドラを統合し 1 つにまとめることで冗長性を解消できる．本章では，3 章で示した類似したイベントハンドラを統合するための手順とそれをツールにより自動化するために必要な処理を示す．

4.1 統合の手順

4.1.1 字句列が一致するイベントハンドラ統合手順

イベントハンドラを統合するプロセスを Fig.7 に示し，その説明を以下に記す．尚，以下の番号は Fig.7 の番号と対応している．

(1) 各コントロールのイベントにイベントハンドラを受け渡している箇所を確認する．
(2) 手順(1)で渡されていたイベントハンドラの中身を全て比較し，3.1 節で示された条件に合う統合可能なイベントハンドラを調べる．
(3) 手順(2)で見つけた統合可能なイベントハンドラのうち，適当なものを 1 つコピーして新たなイベントハンドラを作る．
(4) 手順(2)の統合対象であるイベントハンドラを受け渡している箇所を手順(3)のイベントハンドラを渡すよう書き換える．
(5) テストし動作が変わってないことを確認する．
(6) 古いイベントハンドラを全て削除する．

以上のプロセスを C#に適用した例を Fig.8 に示す．図中の番号は，上記プロセスおよび Fig.7 の番号と同じである．

4.1.2 部分的に異なるイベントハンドラ統合手順

電卓の例では，リテラルがイベントの発生元によって異なる．このようなリテラル，参照型変数が部分的に異なるイベントハンドラを統合するには異なっている箇所をイベント発生元に応じた値を返す処理に置き換える必要がある．これはハッシュマップを用いることにより解決する．Fig.9 にリテラル，参照型変数が部分的に異なるイベントハンドラ統合前後のイベントハンドラの振る舞いを示す．

内容が全て同じイベントハンドラの統合手順と異なる点は(2)(3)(4)である．これらが次の(a)～(h)の手順に変わる．Fig.10 に下記プロセスを示す．

(a) 手順(1)で渡されていたイベントハンドラの中身を全て比較し，3.2 節の条件に合うイベントハンドラを対象のイベントハンドラとする．
(b) 手順(1)の内容を基に統合対象となるイベントハンドラがどのコントロールに渡されているか調べる．
(c) イベント発生元に応じた値を返すためのハッシュマップや各コントロールに対応するリテラル，参照先を登録するためのメソッドがあるか調べ，無い場合はそれらを追加する．
(d) ハッシュマップの値にあたる内部クラスにイベントハンドラごとに異なるリテラル，参照先を確保するためのメンバ変数を追加する．
(e) 各コントロールに対応するリテラル，参照先を登録するメソッドを実装する．ここでは，手順 ii で調べたイベント発生元コントロールの参照型変数をキーとし，手順(d)のクラスを値として，ハッシュマップに登録する．
(f) (e)のメソッドがコンストラクタから呼ばれていない場合は呼び出す処理を追加する．追加箇所はコントロールのインスタンス化直後とする．
(g) 統合対象のイベントハンドラで，適当なものを 1 つコピーし，リテラル，参照型変数が異なる部分をハッシュマップから取得する処理に書き換え，新たなイベントハンドラを作成する．
(h) 統合対象のイベントハンドラを渡す箇所を手順(g)のイベントハンドラを渡すよう書き換える．

4.1.1 と同様に，上記のプロセスを C#の電卓アプリケーションに適用した例を Fig.11 に示す．

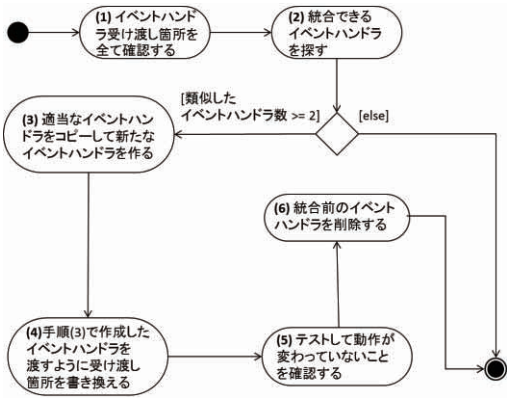

Fig.7 The process of unifying redundant event handlers

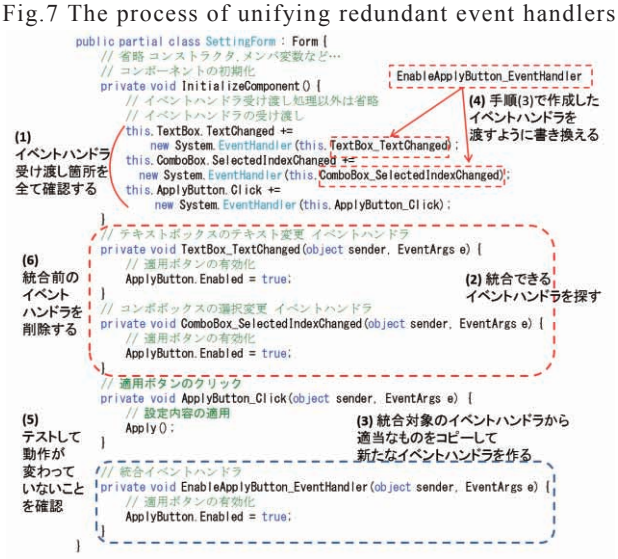

Fig.8 An example of the unifying method(1)

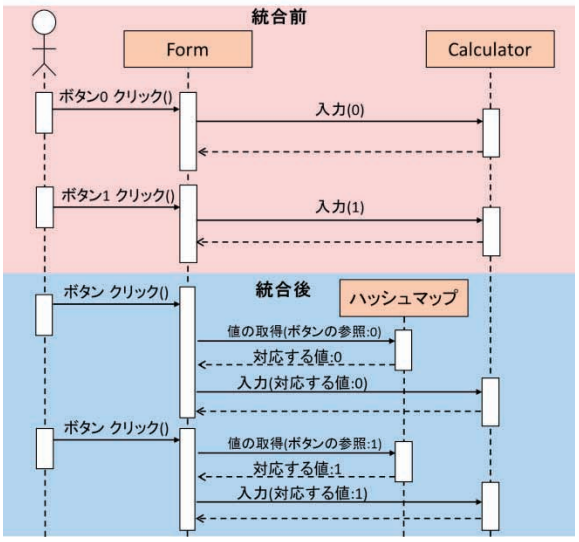

Fig.9 A comparison of behaviors of handlers on the unifying method for partly different symbols

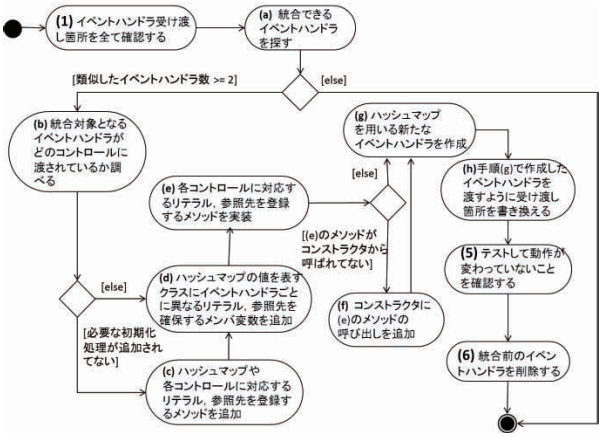

Fig.10 The process of unifying redundant event handlers for partly different symbols and referring types

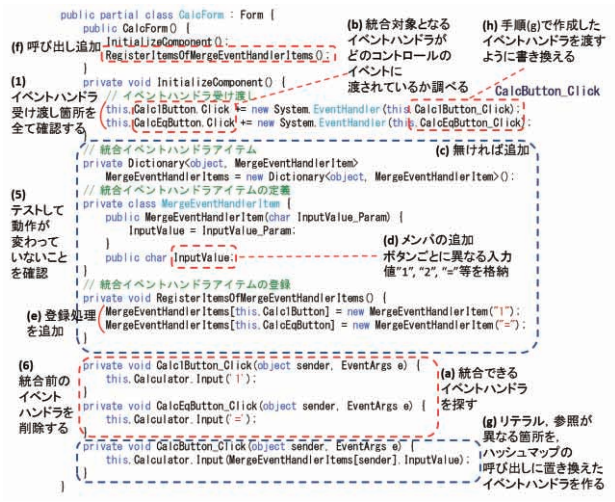

Fig.11 An example of the unifying method(2)

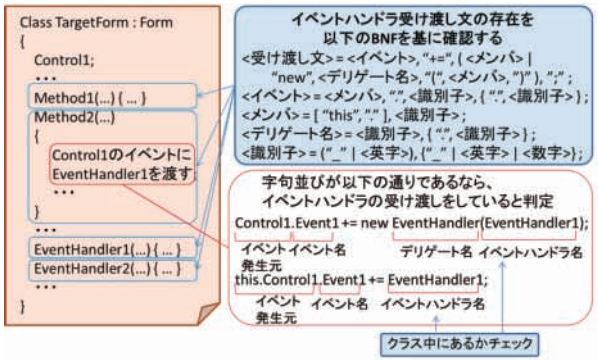

Fig.12 Detecting event handlers

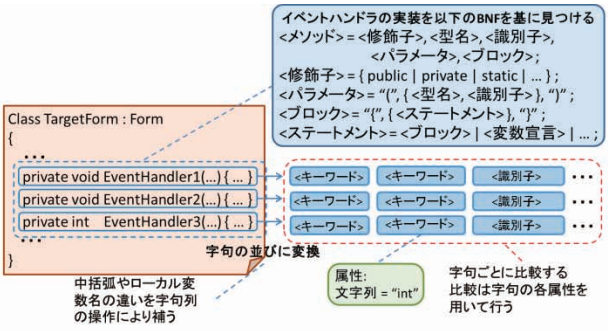

Fig.13 Lexical analysis for event handlers

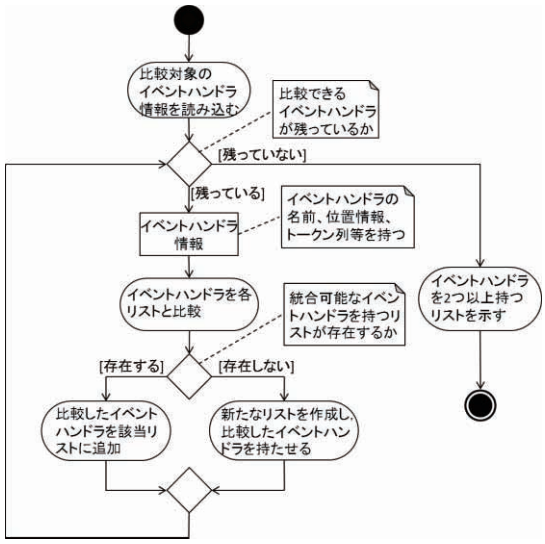

Fig.14 Constructing a list of event handlers for unifying

4.2 C#におけるイベントハンドラ統合の自動化

本節では，C#における統合の自動化を実現する方法について述べる．比較対象となるイベントハンドラは，Fig.3 で示した EBNF の受け渡し文で渡されているメソッドである．イベントハンドラを探す手順を Fig.12 に示す．図では，TargetForm クラス内の全てのメソッドをチェックし，イベントハンドラの受け渡しと思われる記述を見つければ，渡されているイベントハンドラが TargetForm クラス中で定義されているかを確認する．定義されていれば比較対象のイベントハンドラとする．

類似したイベントハンドラの発見は，比較対象であるイベントハンドラをそれぞれ字句並びに変換し，それらを比較することで行っている．統合可能かどうかは 3 章で示した条件に基づき判断している．3 章の条件では，中括弧の有無やローカル変数名などが違っても統合対象としている．このために字句並びに変換する際に中括弧や変数名の違いを補っている．Fig.13 に字句並びへの変換の様子を示す．

ツールの実用性を考えると，統合可能なイベントハンドラの確認や実際に統合するかの選択をユーザが行える必要がある．上記を実現するために，統合可能なイベントハンドラ同士をまとめたリストを作成しユーザに示す．ユーザはリストから実際に統合するイベントハンドラを選択する．Fig.14 にユーザに示す統合可能なイベントハンドラリスト作成の様子を示す．比較対象となるイベントハンドラごとに統合可能なイベントハンドラがあるリストを探し出し，見つけた場合はそのリストに追加，見つからなかった場合は新たなリストを作成する．最終的に，2 つ以上のイベントハンドラがまとめられているリストをユーザに示す．

5. 外部的振る舞いの保存

リファクタリングは単なる修正ではなく，外部の振る舞いが変わらないことを保障した修正方法であることから，本章では，4 章で示した類似したイベントハンドラの統合により，外部の振る舞いを保つこ

とを示す．ここでは，以下の条件を満たすことで振る舞いを保つこととする．

(1) イベントハンドラが渡されたオブジェクトにおいて，イベントハンドラが呼び出されるステップと，イベントハンドラ終了後に戻るステップが統合前と変わらない．以降，前者を開始地点，後者を終了地点と呼ぶこととする．

(2) イベントハンドラが渡されたオブジェクトによってイベントハンドラが呼び出された際に，渡されてから今までに呼び出された回数が統合前と同じであれば，元からある変数の変化が統合前と変わらない．

(3) リファクタリングにより追加，変更したコードによって，イベントハンドラ外の処理における変数の変化に影響を与えない．

以降，5.1，5.2 節でそれぞれ 4.1.1，4.1.2 の統合において外部の振る舞いを保つことを示す．

5.1 字句列一致の統合に関する振る舞いの保存

統合前後のイベントハンドラ呼び出しの様子を Fig.15 に示す．あるイベント E1～3 において，対応するイベントハンドラを H1～3，開始地点を S1～3，終了地点を F1～F3 とする．統合前後では，呼び出すイベントハンドラが変わるだけで開始地点は変わらない．終了地点は開始地点の次のステップなので開始地点が変わらなければ変化しない．よって(1)は満たされる．

条件(2)を満たすのは，イベントハンドラに同一の値で初期化される静的なローカル変数が更新されない場合に限る．静的なローカル変数はメソッドごとに固有の変数となっており，それらを統合して共有してしまうと，統合前後の変数の変化が変わってしまう．上記以外でイベントハンドラから参照される変数はローカル変数，メンバ変数，静的なメンバ変数のみである．これらの変数は処理内容が同じならば，元のイベントハンドラと同じように変化するため，条件(2)は満たされる．

イベントハンドラ外の処理における変更点はイベントハンドラの受け渡しで指定するイベントハンドラのみなので，イベントハンドラ外での処理には影響を与えない．よって条件(3)も満たされ，外部の振る舞いは変わらない．

5.2 部分的に異なる統合に関する振る舞いの保存

リテラル，参照型変数が部分的に異なるイベントハンドラの統合では，異なっていた箇所がイベント発生元に応じた値を返す処理に置き換えられる．返される値が統合前のリテラル，参照先と変わっていなければ，5.1 と同様に条件(1)(2)が満たされる．条件(3)については，インスタンス化の際にコントロールごとのリテラル，参照先を登録するメソッドを追加するだけなので，イベントハンドラ外に影響を与えることはない．

統合前と異なる値が返る，または値の取得で失敗してしまうケースを以下に示す．返される値が統合前のリテラル，参照先と変わらないことを保証するためには，以下のケースに注意しなければならない．

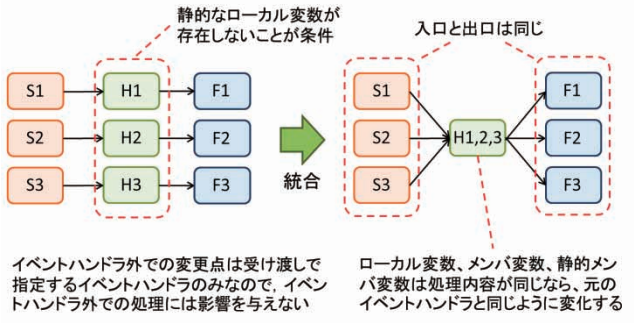

Fig.15 A change of calling way by the unified method

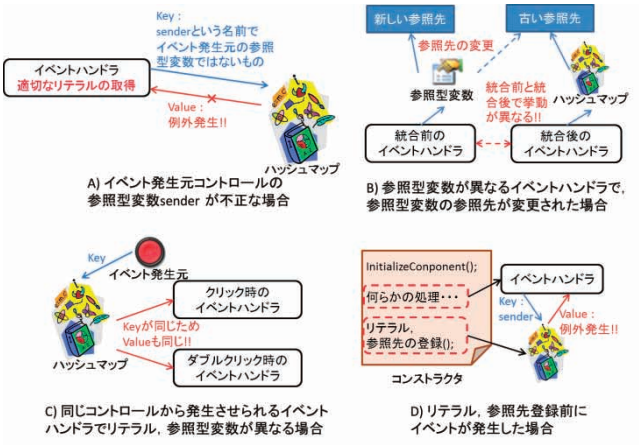

Fig.16 Cases with incorrect return values

A) イベント発生元のコントロールを示す参照型変数の参照先が不正である場合

B) イベントハンドラ中で使われる参照型変数の参照先がアプリケーション実行中に変更される場合

C) 同じコントロールから発生されるイベントハンドラでリテラル，参照先が異なる場合

D) リテラル，参照先を登録する前にイベントが発生した場合

次に C#における具体例を述べる．また，Fig.16 にそれぞれの例を示す．

A)の例は，イベントハンドラが sender という名前の仮引数を持つが，実引数がイベント発生元のコントロールを示す参照型変数でない場合である．このようなケースを避けるためには，イベントハンドラを呼び出している箇所を確認する必要がある．

B)は参照型変数に代入が行われる際に起こる．コントロールによって異なるリテラル，参照先を登録する処理は初期化の段階でしか呼ばれないため，途中で参照先が変わっても変更を反映できない．途中で参照先が変わる参照型変数を使うイベントハンドラは統合しないようにする必要がある．

C)の例はクリックした時とダブルクリックした時のイベントハンドラで，一部リテラルが異なる場合が挙げられる．リテラル，参照型変数の違いはハッシュマップにイベント発生元のコントロールをキーとして渡すことで対応しているので，このような場合は同じ値を受け取ってしまう．このようなイベントハンドラの統合は避けなければならない．

D)はリテラル，参照先を登録するメソッドが呼び出される前にイベントが発生してしまった場合に起こる．これを回避するには，できるだけ早い段階で登録メソッドを呼び出す必要がある．最短では各コントロールのインスタンス化が済んだ段階で呼び出すことができる．

上記より，部分的に異なるイベントハンドラの統合では，A)～D)を回避することで，外部の振る舞いを保つことができる．

6. 適用例：募金箱システム

4 章で提案した類似したイベントハンドラ統合手法を募金箱と SNS の連動システム「ぶたツイ」で行った．「ぶたツイ」は入金を感知するための通過センサを搭載しており，入金が確認されたら，SNS に入金されたことをつぶやく．さらに，木が成長するアニメーションの表示と募金額推移を表すグラフの更新を行う．Fig.17 にぶたツイを示す．

評価はぶたツイの各スクリーンクラスに対して適用することで行う．スクリーンクラスは，システム中の各画面を表しており，それぞれベーススクリーンクラスを継承して定義されている．ベーススクリーンクラスは UserControl クラスを継承しており，スクリーン遷移のためのメソッドが追加されている．ベーススクリーンクラスは基本的には UserControl と同じなので，統合開発環境を用いて画面設計を行うことができる．1 章で述べたとおり，統合開発環境から画面設計を行うことで，類似したイベントハンドラができてしまう可能性がある．そのため，本手法を適用することでソースコードの冗長さを解消できる見込みがある．

具体的に適用によって効果が見込めるのは，Fig.17 に示すようなスクリーン遷移メソッドを呼び出すイベントハンドラと，設定スクリーンの適用ボタン有効化のイベントハンドラである．前者は遷移先スクリーンを指定する文字列リテラルが異なるだけ，後者は 3.1 節の適用ボタン有効化の例と同じケースである．

7. 評価・考察

本章では 6 章で紹介したシステムを例に考察する．提案方法の適用前後のプログラムについて Fig.18, 19 にそれぞれ記す．この例ではホームスクリーンクラス中のスクリーン遷移を行うイベントハンドラに対して提案手法を適用している．また，6 章で紹介した適用例により評価した結果を Table.1 に記す．上記の図表より，プログラムの行数への削減効果は少ないが，イベントハンドラ数は半分近く削減された．

Fig.18, 19 で示された，適用前と適用後のソースコードを比べると，適用前に 5 つあったスクリーン遷移を行うイベントハンドラが適用後は 1 つに統合されている．また，イベントハンドラの受け渡しも統合されたイベントハンドラを受け渡すように変わっている．一方，Fig.19 の上の点線で示した遷移先のスクリーン名の違いを解消するためのコードがあるため，行数自体はほとんど変わっていない．

Table.1 Result of applying the method to “Buta-twii”

		ホームスクリーン		設定スクリーン		合計
		初期化部	EH部	初期化部	EH部	
適用前	行数	93	24	442	48	607
	メソッド数	2	6	3	12	23
適用後	行数	111	8	442	20	581
	メソッド数	3	2	3	5	13
差分	行数	18	−16	0	−28	−26
	メソッド数	1	−4	0	−7	−10

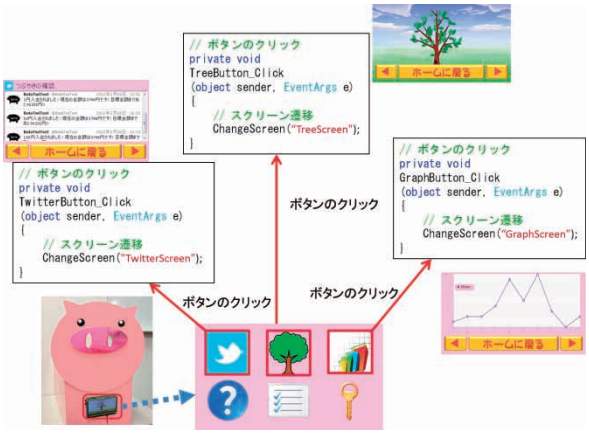

Fig.17 Redundant event handlers for screen transitions

```
// ホームスクリーン
public partial class HomeScreen : BaseScreen {
    // 省略 コンストラクタ,メンバ変数など…

    // コンポーネントの初期化（このメソッドは基本的に統合開発環境によって書き換えられる）
    private void InitializeComponent() {
        // イベントハンドラ受け渡し処理以外は省略
        //   各コントロールのインスタンス化やそれらの配置処理
        //   サイズ，位置などのプロパティの設定など

        // 各ボタンに対応するイベントハンドラの受け渡し
        this.m_ExitButton.Click += new System.EventHandler(this.m_ExitButton_Click);
        this.m_SettingsButton.Click += new System.EventHandler(this.m_SettingsButton_Click);
        this.m_GraphButton.Click += new System.EventHandler(this.m_GraphButton_Click);
        this.m_HelpButton.Click += new System.EventHandler(this.m_HelpButton_Click);
        this.m_TreeButton.Click += new System.EventHandler(this.m_TreeButton_Click);
        this.m_TwitterButton.Click += new System.EventHandler(this.m_TwitterButton_Click);
    }

    // ツイッターボタンが押された際に実行されるイベントハンドラ
    private void m_TwitterButton_Click(object sender, EventArgs e) {
        // "TwitterScreen"に遷移
        ChangeScreen("TwitterScreen");
    }
    // ツリーボタンが押された際に実行されるイベントハンドラ
    private void m_TreeButton_Click(object sender, EventArgs e) {
        // "TreeScreen"に遷移
        ChangeScreen("TreeScreen");
    }
    // 省略 同様にグラフボタン，ヘルプボタン，設定ボタンと続く…
}
```

類似したイベントハンドラ

Fig.18 Source code before unifying event handlers

```
// ホームスクリーン
public partial class HomeScreen : BaseScreen {
    // 省略 コンストラクタ,メンバ変数など…

    // コンポーネントの初期化（このメソッドは基本的に統合開発環境によって書き換えられる）
    private void InitializeComponent() {
        // イベントハンドラ受け渡し処理以外は省略
        //   各コントロールのインスタンス化やそれらの配置処理
        //   サイズ，位置などのプロパティの設定など

        // 各ボタンに対応するイベントハンドラの受け渡し
        //   終了ボタン以外は統合後のイベントハンドラを受け渡している
        this.m_ExitButton.Click += new System.EventHandler(this.m_ExitButton_Click);
        this.m_SettingsButton.Click += new System.EventHandler(this.HomeButtons_Click);
        this.m_GraphButton.Click += new System.EventHandler(this.HomeButtons_Click);
        this.m_HelpButton.Click += new System.EventHandler(this.HomeButtons_Click);
        this.m_TreeButton.Click += new System.EventHandler(this.HomeButtons_Click);
        this.m_TwitterButton.Click += new System.EventHandler(this.HomeButtons_Click);
    }

    // 統合イベントハンドラアイテム
    private Dictionary<object, MergeEventHandlerItem> MergeEventHandlerItems =
        new Dictionary<object, MergeEventHandlerItem>();
    // 統合イベントハンドラアイテムの定義
    private class MergeEventHandlerItem {
        // コンストラクタ
        public MergeEventHandlerItem(string ScreenName_Param) {
            ScreenName = MergeEventHandlerItem0_Param;
        }
        // 遷移先のスクリーン名
        public string ScreenName;
    }
    // 統合イベントハンドラアイテムの登録
    private void RegisterItemsOfMergeEventHandlerItems() {
        MergeEventHandlerItems[this.m_TwitterButton] =
            new MergeEventHandlerItem("TwitterScreen");
        MergeEventHandlerItems[this.m_TreeButton] =
            new MergeEventHandlerItem("TreeScreen");
        // 省略 同様にグラフボタン，ヘルプボタン，設定ボタンと続く…
    }
    // 統合後のイベントハンドラ
    private void HomeButtons_Click(object sender, EventArgs e) {
        ChangeScreen(MergeEventHandlerItems[sender].ScreenName);
    }
}
```

遷移先スクリーン名の違いを解消するためのコード

統合後のイベントハンドラ

Fig.19 Source code after unifying event handlers

類似したイベントハンドラを統合することによる行数，メソッド数の削減効果の大きさは下記の式に応じた程度となる．n は統合するイベントハンドラの数を，m は 1 つのイベントハンドラの行数示す．

$$\text{削減メソッド数} = n - 1$$

$$\text{削減行数} = m \times \text{削減メソッド数}$$

一部リテラル，参照型変数が異なるイベントハンドラでは初期化のためにメソッドの数が 1 つ増える．また，それぞれのイベントハンドラ中に含まれる異なるリテラル，参照型変数の数をlとし，n，m，lの値に影響されない固定の増加値をaとすると，違いを吸収する初期化コードの行数は以下の式で表せる．

$$\text{増加行数} = a + 2l + n$$

上記の式より，全ての字句が一致するイベントハンドラでは，統合するイベントハンドラの数だけ行数，メソッド数を削減できる．一部リテラル，参照型変数が異なるイベントハンドラでは，n，m，lの値によって行数削減の効果が異なる．ホームスクリーンでは，$m = 4$，$l = 1$，$a = 10$のため，$n \leq 5$では統合前より行数が多くなる．上記より，n，m，lの値によっては効果が薄く統合しない方が良い場合もある．また，一時的に重複しているだけのイベントハンドラを統合してしまうと，後で変更しづらくなるため，そのようなイベントハンドラも統合すべきではない．

一部リテラル，参照型変数が異なるイベントハンドラでは，違いを吸収するための初期化処理が追加される．追加されるのは統合するイベントハンドラの数n個分のハッシュマップへの登録処理であり，その分初期化にかかる時間が長くなる．

8. おわりに

本論文では，統合開発環境によってイベントハンドラを追加していくうちに，類似したイベントハンドラが多くできてしまうことで，ソースコードが冗長となってしまう問題について、イベントハンドラ統合手法を提案した。イベントハンドラはソースコードの一部であることから、類似したソースコードの増加を扱う研究と関連し、コードクローンの検出、リファクタリング手法が提案されていることを紹介した。既存方法ではイベントハンドラ受け渡し箇所について考慮されておらず，イベントハンドラをそのまま一つにまとめることができないため，類似したイベントハンドラは解消するには十分でないことを述べた．

提案手法では，類似したイベントハンドラを見つけ出し，一つに統合することで，類似したイベントハンドラを除去できる．さらに，提案手法の適用前後の変化と，それにより外部の振る舞いが変わらないことを示した．最後に，募金箱システムに本手法を適用し，その評価を行った．

今後，本手法の適用による可読性・保守性への効果，Form，UserControl 以外の類似したイベントハンドラやオープンソースソフトウェアに対する適用の有効性の調査・評価を行う．また，統合を手動で行う場合と自動で行う場合の工数やバグ混入数についての比較を行う予定である．

参考文献

1) G.Salvaneschi, M.Mezini : Reactive Behavior in Object-oriented Applications: An Analysis and a Research Roadmap, AOSD2013, ACM 978-1-4503-1766-5/13/03, (2013).

2) C.Kapser, M.W.Godfrey : "Cloning Considered Harmful" Considered Harmful, WCRE2006, pp.19-28, (2006).

3) C.J.Kapser, M.W.Godfrey : "Cloning Considered Harmful" Considered Harmful: Patterns of Cloning in Software, Empirical Software Engineering, Volume 13, Issue 6, pp.645-692, (2008).

4) M.Fowler, K.Beck, J.Brant, W.Opdyke, D.Roberts : Refactoring: Improving the Design of Existing Code, Addison-Wesley Professional, (1999).

5)W.F.Opdyke : Refactoring Object-Oriented Frameworks, PhD Thesis, University of Illinois at Urbana-Champaign, (1992).

6) 谷川郁太，渡辺晴美 : C#における類似したイベントハンドラ統合手法の提案，ソフトウェアエンジニアリングシンポジウム 2013 論文集，pp.1-6，(2013).

7) 井上克郎，神谷年洋，楠本真二 : コードクローン検出法，コンピュータソフトウェア，vol.18，no.5，pp.47-54，(2001).

8) T.Kamiya, S.Kusumoto, K.Inoue : A Code Clone Detection Technique for Object-Oriented Programming Languages and Its Emprical Evaluation, Proc. of the 62nd National Convention of IPSJ, pp. 23-28, (2001).

9)B.S Baker : A Progtam for Identifying Duplicated Code, Proc. of Computing Science and Statistics: 24th Symposium on the Interface, 24, pp. 49-57, (1992).

10) L.Prechet, G.Malpohl, M.Phlippsen : JPlad: Finding plagiarisms among a set of programs, Technical Report, Fakultat fur Informatik Universitat Karlsruhe, (2001).

11) M.Balazinska, E.Merlo, M.Dagenais, B.Lague, K.A. Kontogiannis : Measuring Clone Based Reengineering Opportunities, Proc. of the 6th IEEE Int'l Symposium on Software Metrics (METRICS'99), pp. 292-303, Boca Raton, Florida, (1999).

12)肥後芳樹，神谷年洋，楠本真二，井上克郎 : コードクローンを対象としたリファクタリング支援環境，電子情報通信学会論文誌 Vol.J88-D-I，No.2，pp.186-195，(2005).

13)肥後芳樹，神谷年洋，楠本真二，井上克郎 : コードクローン解析に基づくリファクタリングの試み，情報処理学会論文誌，Vol.45，No.5,pp1357-1366，(2004).

14) 吉岡一樹，吉田則裕，徳永将之，松下誠，井上克郎 : コードクローンの特徴に基づくメソッド引き上げリファクタリングパターンの提案，情報処理学会研究報告，Vol.2011-SE-173，No.7，pp.1-8，(2011).

15) 吉田則裕，肥後芳樹，神谷年洋，楠本真二，井上克郎 : コードクローン間の依存関係に基づくリファクタリング支援，情報処理学会論文，Vol.48，No.3，pp.1431-1442，(2007).

16)堀田圭佑，肥後芳樹，楠本真二 : プログラム依存グラフを用いたコードクローンに対するテンプレートメソッドパターン適用支援手法，電子情報処理学会論文誌 D，Vol.J95-D，No.7，pp.1439-1453(2012).