C#における類似したイベントハンドラ統合手法の提案

谷川郁太† 渡辺晴美†

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．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．本論文では，上記問題を解決するために類似した C#のイベントハンドラ統合手法を提案する．本手法は，ソースコード中の類似したイベントハンドラを見つけ出し，それらを一つに統合することで，ソースコードの冗長化を解消し，ソフトウェアの保守性を向上させる．さらに，本手法の適用により外部の振る舞いが変わらないことと，本手法を適用した評価を示す．評価は，募金箱と SNS との連動システム「ぶたツイ」に適用することで行う．

A Method for Unifying Redundant C# Event Handlers

IKUTA TANIGAWA† HARUMI WATANABE†

Recently, the 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 becoming redundancy. Since we tend to add new event handlers, when we use screen design function on IDE, the number of event handler will increase. To overcome this problem, the paper proposes a method for unifying event handlers. By detecting and unifying similar event handlers, the method can be solved the problem of the redundant source code and the maintainability makes progress. Furthermore, the method can keep the behavior after the unifying. Finally, to evaluate the method, we will apply it to “Buta-twii” which is a cooperating charity pot with SNS, that is, Cloud and Embedded system.

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[16][17]. 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[2][3]．しかし, 類似したイベントハンドラは, 図 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，図 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[4][5]．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，それらを一つに統合することで, 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．

† 東海大学大学院情報通信学研究科
Tokai University Graduate School of Information and Telecommunication Engineering

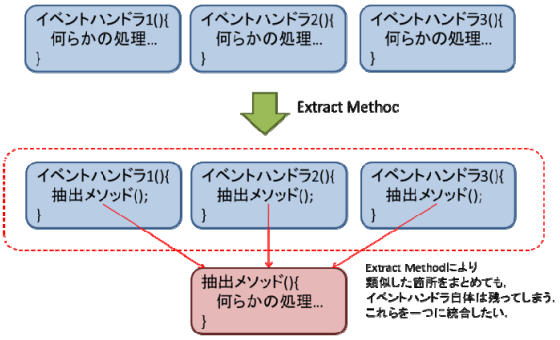

図 1　Extract Method 適用の問題
Figure 1　A problem of applying “Extract method”

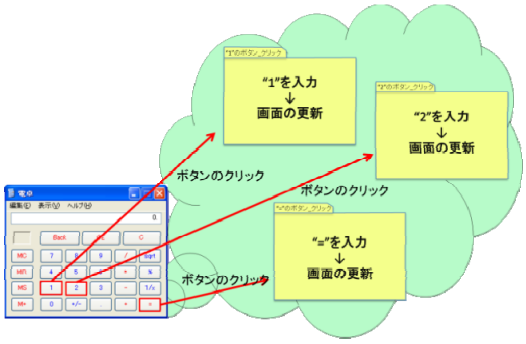

図 2　電卓ボタンの例
Figure 2　An example of buttons on pocket calculator

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[6][7][8][9][10]．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[11][12][13][14][15]．

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[7]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[14][15]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，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[14]．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[15]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. 統合対象となる類似したイベントハンドラ

本論文で対象とするイベントハンドラは次のいずれかの条件を満たすものである．「A) 戻り値の型，引数の型が同じであり，各文の順番，内容が全て同じ」「B) 上記からリテラル，参照型変数が部分的に異なる」ただし，対象となるイベントハンドラは統合を行うクラス内で定義されたイベントハンドラのみとする．本章各節でそれぞれの詳細と例を示す．

3.1 各文の実行順序，内容が全て同じケース

複数のイベントハンドラを比較した結果，戻り値の型，引数の型が同じであり，ブロック中の各文の実行順序，内容が全て同じであるなら，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．内容が同じというのは，それぞれの文の字句が同じということである．また，以下のような場合も字句解析の段階で補完することにより，内容が同じであると判断できる．

- int a = 0;と int b = 0;ローカル変数名以外は同じる場合
- sbyte a = 15;と sbyte a = 0x0f;リテラルの表記方法が異なっても，内容が同じである場合
- if や for，while 文などにおいて，中括弧の有無で結果が変わらない場合

各文の順番，内容が全て同じであ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．図 3 にその様子を示す．このように，全てが同じ内容であ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．例えば，a = 0;と a = 1;といった文や this.textBox1.Text = “a”と this.textBox2.Text = “b”といった文である．図 2 で示した電卓の例もこれに当たる．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．

イベントハンドラの引数には統合方法の関係上，イベント発生元のコントロールを示す参照型変数の sender が必要となる．sender が無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．

4. 類似したイベントハンドラ統合手法

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 同じ内容のイベントハンドラ統合手順

図 4 に，類似したイベントハンドラの統合手順を示す．(1)～(6)の順序で統合する．

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 C#における HashMap の発展系である Dictionary 型メンバを用いることにより解決する．図 5 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例を示す．

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．さらに，図 6 に電卓の例で下記手順を行う時の流れを示す．i～viii の順に統合を行う．

4.2 自動化

我々が以前に展示した，自動でイベントハンドラ統合を行うツール[4]の構造を図 7 に示す．本ツールでは，字句解析を C#用の lex を用いて行い，その結果を用いて比較対象となるイベントハンドラの判定や各イベントハンドラが類似しているかどうかの比較を行っている．上記の処理はそれぞれ EventHandlerChecker.cs と TokenListCreater.cs から字句解析時のアクションを追加することで実装している．

比較対象となるイベントハンドラは 3 章で述べたとおり，統合を行うクラス内で定義されたイベントハンドラのみである．このようなイベントハンドラを探す手順を図 8 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探索は，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．図 9 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．図 10 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．

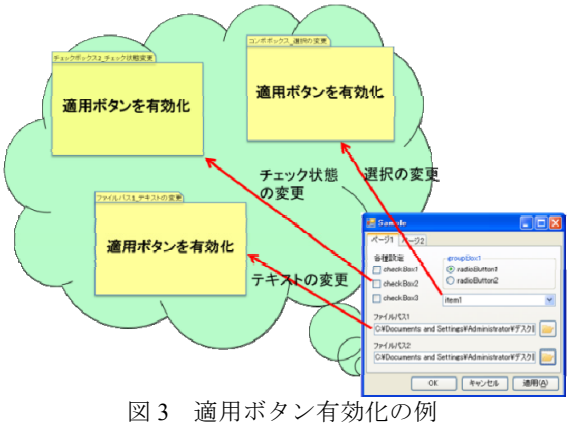

図 3　適用ボタン有効化の例
Figure 3　An example of activating buttons

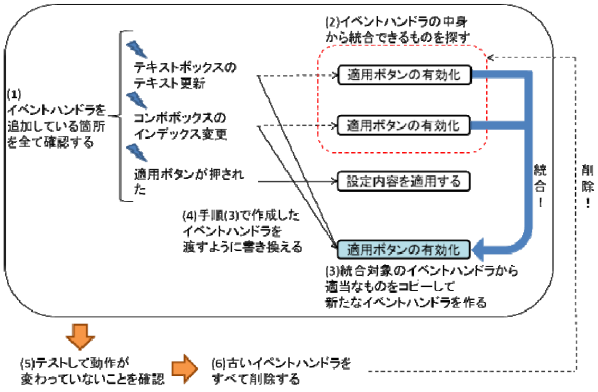

図 4　類似したイベントハンドラの統合手順
Figure 4　The process of unifying redundant event handlers

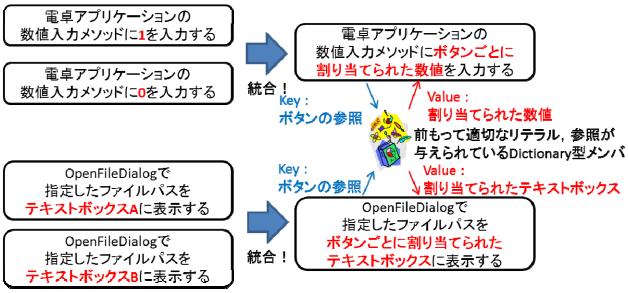

図 5　リテラル，参照型変数が部分的に異なるイベントハンドラの統合方法
Figure 5　A unifying method for partly different symbols and referring type

5. 外部的振る舞いの保存

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手順により，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) イベントの開始，終了地点が統合前と変わらない．

(2) イベントハンドラを実行した際に実行回数とタイミングが同じであれば，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における変数や出力の変化に影響を与えない．

5.1 同じ内容のイベントハンドラ統合手順

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，終了時に到達するところを F とすると，イベントハンドラ統合前と統合後を図 11 のように表せる．図より，統合前後でイベントの開始，終了地点は変わらないので，(1)は満たされる．

C#ではメソッド内で static 変数を宣言することができないので，メソッド固有の変数ができることは無い．イベントハンドラで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)も満たされる．

イベントハンドラ外での変更点はイベントハンドラの追加，削除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 部分的に異なるイベントハンドラ統合手順

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手順では，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図 12 にまとめる．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，図 12 の問題が起こらないように注意しなければならない．

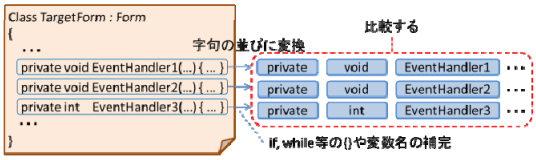

図 9 イベントハンドラの字句解析
Figure 9 Lexical analysis for event handlers

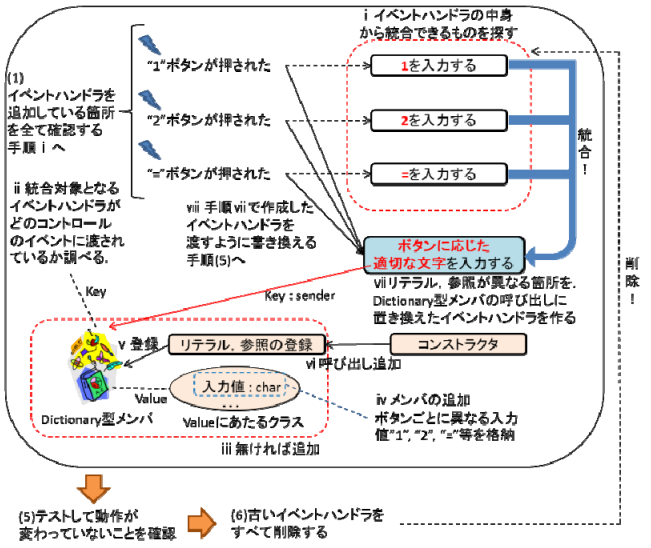

図 6 提案方法の電卓への適用
Figure 6 An applying our method into pocket calculator

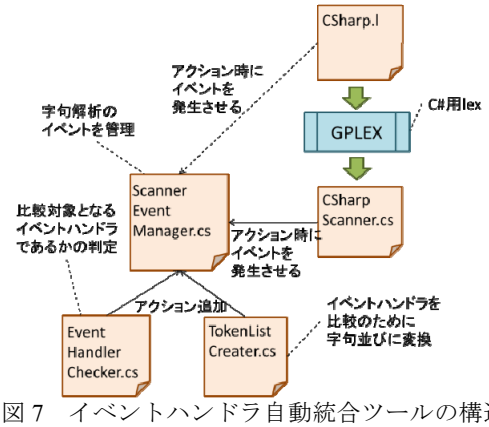

図 7 イベントハンドラ自動統合ツールの構造
Figure 7 The structure of the tool for unifying event handlers

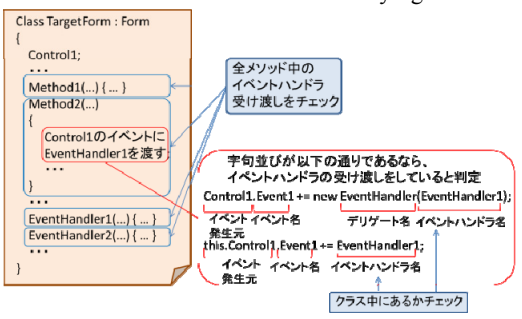

図 8 統合対象となるイベントハンドラの判定
Figure 8 Detecting event handlers

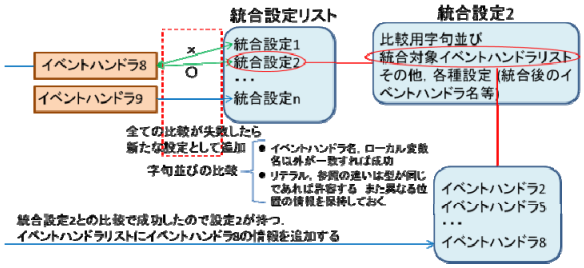

図 10 統合可能なイベントハンドラリスト作成
Figure 10 Constructing a list of event handlers for unifying

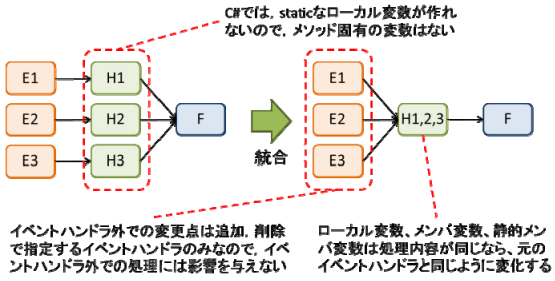

図 11　統合前後の構造
Figure 11　Structure of before and after integration

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，D2C コンテストの作品である募金箱と SNS の連動システムである「ぶたツイ」に適用した[18].「ぶたツイ」は，図 13 に示すとおり，複数のスクリーンを有す．評価はぶたツイの各スクリーンクラスに対し行う．スクリーンクラスは，システム中の各画面を表しており，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場合，類似したイベントハンドラができる．図 14 にぶたツイのクラス図と適用する箇所を示す．具体的に適用によって効果が見込めるのは，図 15 に示す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図 16 に示す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場合である．

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図 17, 18 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表 1 に記す．表 1 に記す通り，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 30%削減された．

図 17，18 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，図 18 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．

適用ボタン例の内容が同じイベントハンドラや電卓の部分的に異なるイベントハンドラが多数ある場合，それらを統合することで類似したイベントハンドラを除去することができる．一方で，統合しない方が良いケースもある．例えば，部分的に異なる類似したイベントハンドラが 2,3 個と少ない場合，それらの違いを吸収するためのコードの方が大きくなり，かえって可読性が落ちることもある．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．これらの判断を自動で行うのは難しいので，ツールではユーザに任せている．

表 1　統合手法の適用結果
Table 1　Result of applying the method to “Buta-twii”

	全体行数	適用範囲行数	イベントハンドラ数
適用前	7006	2437	40
適用後	6982	2413	28

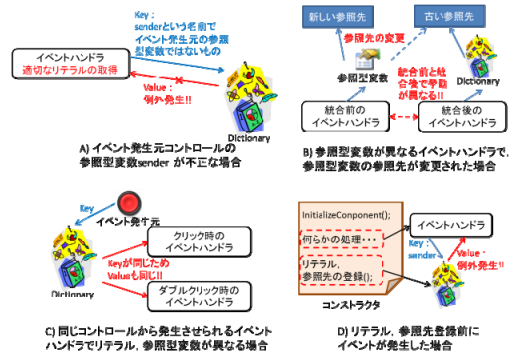

図 12　イベント発生元に応じた値が正常でないケース
Figure 12　Cases where the return value is incorrect

8. おわりに

本論文では，C#における類似したイベントハンドラの統合手法を提案し，募金箱システムへ適用することでその評価を行った．さらに，適用によって外部の振る舞いが変わらないことも示した．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・評価を行う予定である．

図 13　ぶたツイの機能
Figure 13　The functions of “Buta-twii”

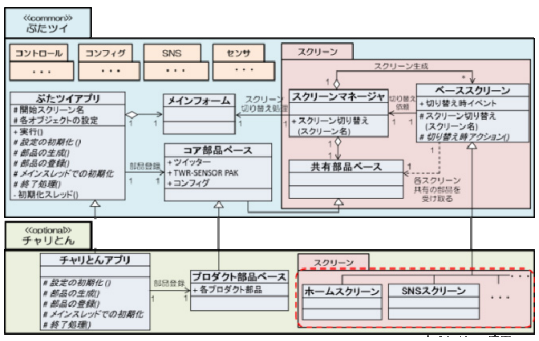

図 14　提案方法の適用箇所
Figure 14　The area of applying our method

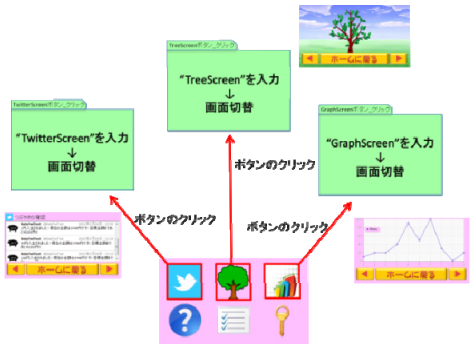

図 15　スクリーン遷移の類似したイベントハンドラ
Figure 15　Redundant event handlers for screen transitions

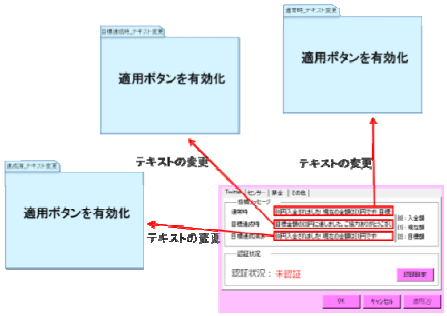

図 16　適用ボタン有効化の類似したイベントハンドラ
Figure 16　Redundant event handlers for activating buttons

```
// 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

図 17　イベントハンドラ統合前のソースコード
Figure 17　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

図 18　イベントハンドラ統合後のソースコード
Figure 18　Source code after unifying event handlers

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)　M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts : Refactoring: Improving the Design of Existing Code, Addison-Wesley Professional, (1999). 児玉公信，友野晶夫，平澤章，梅沢真史(訳)：リファクタリング プログラムの体質改善テクニック, (2000).

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

4)　谷川郁太，石井貴大，鈴木智明，渡辺晴美 : C#における冗長なイベントハンドラの合成ツール，組込みシステムシンポジウム 2012 論文集, pp.214-215, (2012).

5)　谷川郁太，石井貴大，鈴木智明，渡辺晴美 : C#における冗長なイベントハンドラ統合手法の提案，情報処理学会研究報告, Vol.2012-EMB-27, No5, pp.1-7, (2012).

6)　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).

7)　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).

8)　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).

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

10)　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).

11)　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).

12)　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).

13)　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).

14)　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).

15)　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).

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

17)　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).

18)　D2C コンテスト(Device2Cloud Contest) http://www.d2c-con.com/