Vol.2012-OS-123 No.5
Vol.2012-EMB-27 No.5
2012/12/5

C#における冗長なイベントハンドラ統合手法の提案

谷川郁太† 石井貴大† 鈴木智明† 渡辺晴美†

近年，組込み Windows 向けのシステムを開発においても C#が利用されるケースが増えてきている．C#の統合開発環境では，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問題がある．本論文では，上記問題を解決するための類似したイベントハンドラ統合手法を提案する．また，募金箱と Twitter との連動システム「ぶたツイ」に適用し，その効果を評価する．

A Refactoring technique for Reducing C# Event Handlers

IKUTA TANIGAWA† TAKAHIRO ISHII†
TOMOAKI SUZUKI† HARUMI WATANABE†

Recently we have come to use C# for developing embedded systems. In developing system by using “Form” or “UserControl” on IDE, we often create many similar event handlers which cause the problem of becoming redundancy. For the sake of overcoming this problem, the paper proposes a refactoring technique for C# event handlers. To evaluate the technique, we will apply to ButaTwi, a cooperating system with collection box and Twitter, that is, Cloud and Embedded Devices system.

1. はじめに

C#の統合開発環境では，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問題がある．これは，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．組込み Windows 向けのシステムを開発する際も，統合環境から画面設計を行うとこのような問題に陥る可能性がある．本論文では，プログラムを冗長にする原因となる類似したイベントハンドラのことを 冗長なイベントハンドラと呼ぶ．

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．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はコードクローンと呼ばれ，自動検出するために様々な手法が提案されてきた[3][4][5][6][7]．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[8][9][10]．

初めに述べた冗長なイベントハンドラは，図 1 に示すとおり“Extract Method”により冗長な箇所をまとめても，イベントハンドラ自体は残ってしまうため，新たなリファクタリング手法が必要である．我々は以前に，上記の問題を解決するためのイベントハンドラを統合するリファクタリングツールを提案した[13]．図 2 に以前提案したリファクタリングツールを示す．このツールは統合可能なイベントハンドラをユーザに示し，統合するように指示されたイベントハンドラを自動的に統合するツールである．本論文では，冗長なイベントハンドラ除去のためのイベントハンドラ統合手法を提案し，その効果についての評価と外部の振る舞いの不変性についての考察を行う．

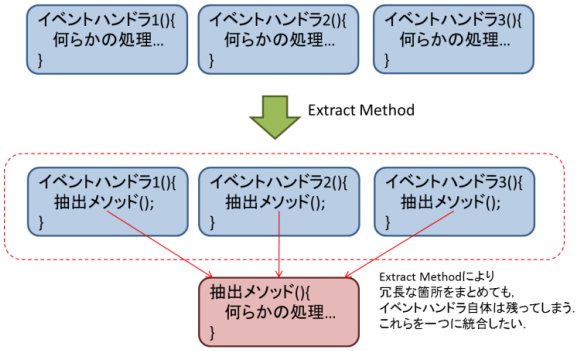

図 1 冗長なイベントハンドラ統合の必要性

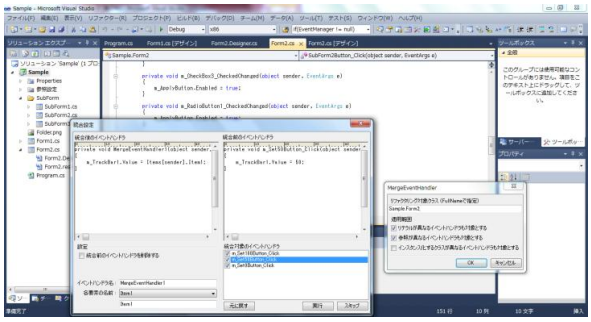

図 2 冗長なイベントハンドラ統合ツール(アドイン形式)

2. 統合対象となる冗長なイベントハンドラ

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，内容が全く同じである

B) 上記から一部のリテラル，参照が異なる

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

†東海大学情報通信学部
Tokai University School of Information and Telecommunication Engineering

Vol.2012-OS-123 No.5
Vol.2012-EMB-27 No.5
2012/12/5

内で定義されたイベントハンドラのみとする．基底クラスや外部のクラスで定義されているイベントハンドラは対象外とする．本章各節でそれぞれの詳細と例を示す．

2.1 各文の順番，内容が全く同じケース

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順番，内容が全く同じであるなら，比較対象は統合可能な冗長なイベントハンドラと判断できる．ここでいう順番とは，各文が実行される順番のことである．また，内容が同じというのは，それぞれの文が意味的に同じということである．以下に内容が同じである例と異なる例をそれぞれ示す．

<内容が同じである例>

- int a = 0;と int b = 0;のようにローカル変数名以外は同じである場合
- uint a = 1;と uint a = 1U のようにリテラルのプリフィックスが異なっても，最終的な型が同じである場合
- sbyte a = 15;と sbyte a = 0x0f;のように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
- メンバ変数 m があり，m = 0;と this.m = 0;のように this の有無以外は同じである場合
- 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
- int a = 1 + 2 * 3;と int a = 1 + (2 * 3);のように括弧が付いていても，計算の順序は変わらない場合
- イベントハンドラ内で呼び出し可能な定数 C(const または readonly のついた変数)があり，int a = C; (C=1とする)と int a = 1;といった場合
- 組込み型の整数変数 a があり，if(a < 10)…と if(a <= 9)…のように文中に含まれる条件式の書き方が異なっていても判定結果が同じである場合
- 組込み型の整数変数 a があり，a += 1;と++a;と a++;のようにインクリメントのみの文で最終的な結果が変わらない場合(デクリメントの場合も同様)

<内容が異なる例>

- ユーザ定義型の変数 a があり，++a;と a++;のように，文中のインクリメントが前置，後置異なる場合(デクリメントの場合も同様)．これは，演算子のオーバーロードの内容が異なる可能性があるため
- if(a++ < 10) {…}と if(++a < 10) {…}といったように，インクリメント・デクリメントの前置，後置で意味が違ってしまう場合(文中にインクリメント・デクリメント以外の要素が入っていれば，前置，後置で意味が変わる)
- お互いに参照先が同じである参照型の変数 a，b があり，a.fnc();と b.fnc();のように同じメンバ，メソッドを呼び出している場合．これは，変数の参照先が途中で変わることを考慮している

各文の順番，内容が全く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，まったく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．

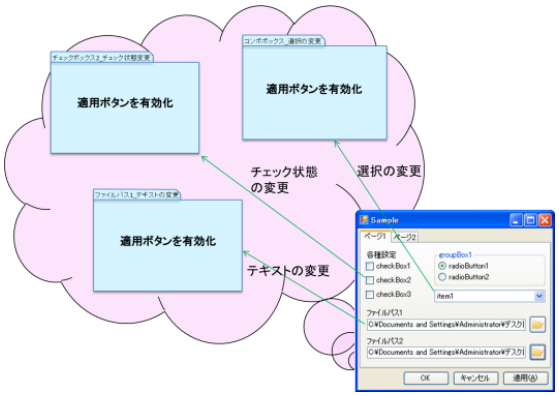

図 3 適用ボタン有効化の例

2.2 一部のリテラル，参照が異なるケース

基本的には 2.1 節の条件を満たす文で構成されており，一部の文のリテラル，参照が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．さらに，イベントハンドラの引数には統合方法の関係上 sender 引数(イベント発生元のコントロールの参照)が必要となる．sender 引数が無いものは統合の対象外とする．このようなイベントハンドラの統合方法は 3 章で述べる．

以下に，一部のリテラル，参照が異なるが，統合対象となるようなイベントハンドラの例を示す．また，図 4，5 にその様子を示す．

(1) 電卓において，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るケース．例えば 1 ボタンのイベントハンドラでは，ボタンが押された際に内部に”1”を送り，何らかの処理後に画面を更新する．2 ボタンでは，内部に入力されるものが”2”であり，=ボタンでは，”=”が入力される．入力以外の処理内容は全て同じである．

(2) 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る場合，ボタンごとにイベントハンドラができるケース．図 5 において，上のファイル指定ボタンから指定したファイルのパスは上のテキストボックスに，下のボタンで指定したファイルのパスは下のテキストボックスに表示する．これらのボタンのイベントハンドラは，表示先のテキストボックスの参照が異なるだけである．

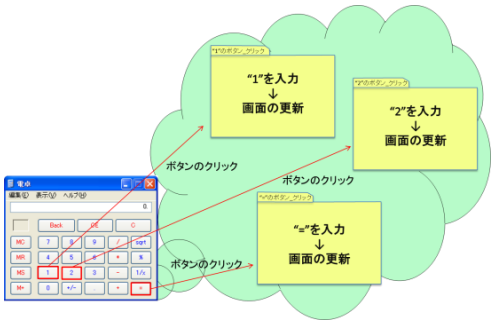

図 4 電卓ボタンの例

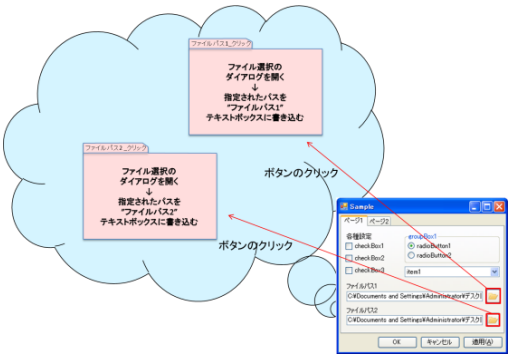

図 5 ファイルパス指定の例

一部のリテラル，参照が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．このようなイベントハンドラが複数あることは冗長であり問題である．

3. 冗長なイベントハンドラ統合手法

冗長な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，2 章で示した冗長な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．

3.1 統合の手順

3.1.1 同じ内容のイベントハンドラ統合手順

2.1 節の例では適用ボタンを有効にするイベントハンドラがコントロールごとにできる．これらイベントハンドラの内容はまったく同じである．このように，まったく同じ内容のイベントハンドラを統合する手順を以下に示す．また，図 6 に手順の流れを示す．

(1) 各コントロールのイベントにイベントハンドラを追加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，統合可能なイベントハンドラ(2.1 節で示された条件に合うイベントハンドラ)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追加･削除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．

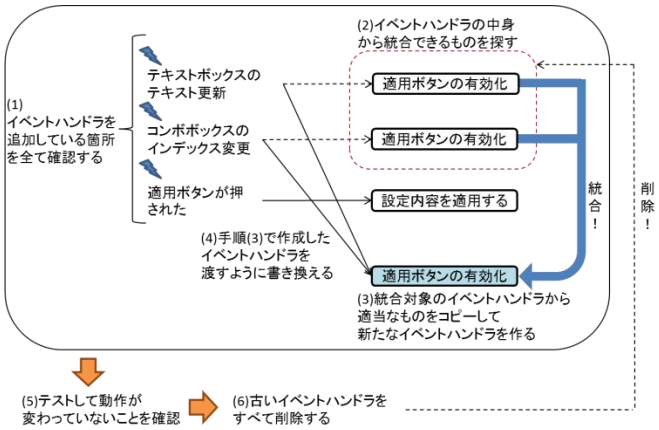

図 6 冗長なイベントハンドラの統合手順

3.1.2 リテラル，参照が異なるイベントハンドラ統合手順

2.2 節の例(1)，(2)ではそれぞれ，リテラル，参照がイベントの発生元によって異なる．このようなイベントハンドラを統合するにはリテラル，参照が異な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#の Dictionary 型(HashMap の発展)メンバを用意することにより解決する．図 7 に 2.2 節の例(1)，(2)におけるイベントハンドラ統合例を示す．

先で述べた内容が全く同じである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．(2)(3)(4)に代わる手順を以下に示す．さらに，図 8 に 2.2 節の例(1)へ下記手順を行う時の流れを示す．

i. 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，2.2 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．さらに，異なっているものが参照である場合は，参照先を変更している箇所が無いかを確認する必要がある．変更している箇所がある場合は，そのイベントハンドラは統合対象から外す．
ii. 手順(1)の確認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のイベントに渡されているか調べる．ここで，同じコントロールから発生させられるイベントハンドラが複数あり，それらにリテラル，参照異なる箇所がある場合は統合対象から外す．クリックした時とダブルクリックした時でリテラル，参照が異なる場合などがこれにあたる．
iii. イベント発生元に応じた値を返すための Dictionary 型メンバ，Dictionary 型メンバの Value にあたる内部クラス(各イベントハンドラ内で異なるリテラル，参照を確保するための変数を集めたもの)，前述の Dictionary メンバに各コントロールに対応するリテラル，参照を登録するためのメソッドが無い場合，それらを追加する．
iv. Dictionary 型メンバの Value 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，参照を確保するためのメンバ変数を追加する．さらに，コン

ストラクタからそれらのメンバを設定できるようにする．

v. Dictionary 各コントロールに対応するリテラル，参照を登録するためのメソッドにおいて，手順 ii で調べたイベント発生元コントロールの参照を Key とし，Value は対象イベントハンドラの手順 iv のクラスとして Dictionary 型メンバに登録する処理を追加する．すでにある場合は適切な形に書き換える．これを全ての統合対象イベントハンドラに対して行う．

vi. v 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．追加する箇所は InitializeConponent()メソッドの直後とする．

vii. 統合対象のイベントハンドラの内，適当なものを 1 つコピーし，対象のイベントハンドラ間でリテラル，参照が異なる部分を Dictionary 型メンバに sender 引数を Key として与え，帰ってきたオブジェクト中の適切なメンバ(手順 iv で追加したメンバ)を呼び出す処理に書き換える．

viii. 統合対象であるイベントハンドラを追加・削除している箇所を手順 vii のイベントハンドラを渡すよう書き換える．

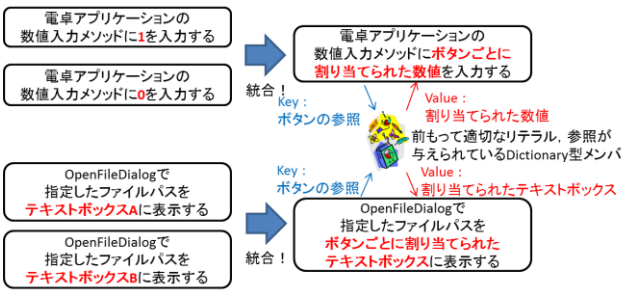

図 7 例(1)，(2)のイベントハンドラ統合例

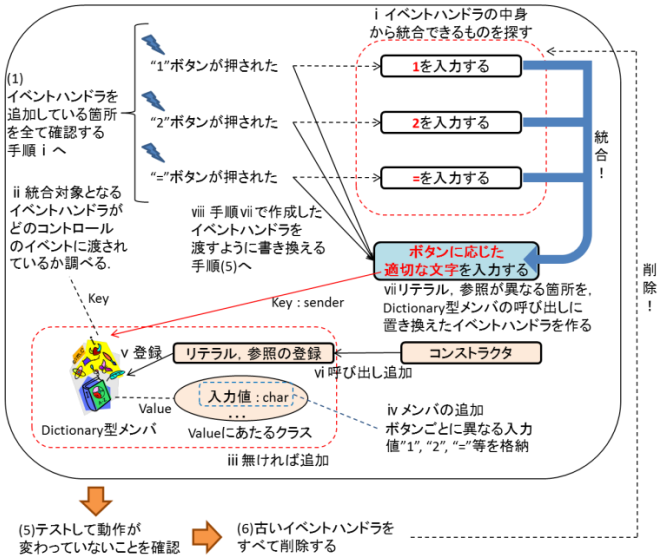

図 8 電卓における統合手順

3.2 自動化

上記の手順を自動化するためには，ソースコードを解析し，統合対象となることが可能なイベントハンドラを適切に探し出さなければならない．そのためには，全てのイベントハンドラの内容を比較する必要がある．図 9 は，どのメソッドが比較対象となるイベントハンドラかを判定する様子を表したものである．TargetForm クラス内の全てのメソッド中のイベントハンドラ受け渡しをチェックし，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調べる．定義されていれば，それは比較対象のイベントハンドラとして扱う．これは 3.1 節の手順(1)に相当する．

イベントハンドラの比較は同一または類似しているイベントハンドラを見つけるという点で，コードクローンの検出に似ている．コードクローン検出において，検出のための粒度は文字，字句(Token)[4]，行[5]，文[6]，関数・手続き・クラス定義[6]が挙げられる[3]．我々が以前に提案した冗長なイベントハンドラ統合ツール[13]では字句単位のコードクローン検出法[4]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，字句解析には比較的少ない手間で行うことができるという利点もある．

冗長なイベントハンドラ統合ツールでは，統合対象であるイベントハンドラの定義全体を字句並びに変換し，それらを比較することで統合可能なイベントハンドラかどうかを判断している．統合可能かどうかは 2 章で示した条件に基づき判断している．2 章条件では，中括弧の有無や this の有無などが違っても意味が同じであれば統合対象としているので，字句並びに変換する際に中括弧や this を補っている．図 10 に字句並び比較の様子を示す．

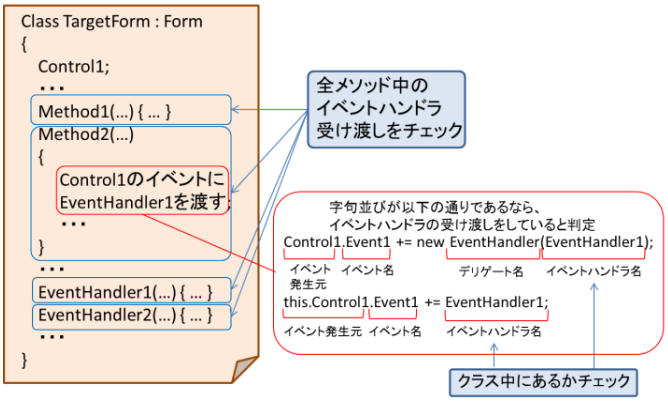

図 9 比較対象となるイベントハンドラの判定

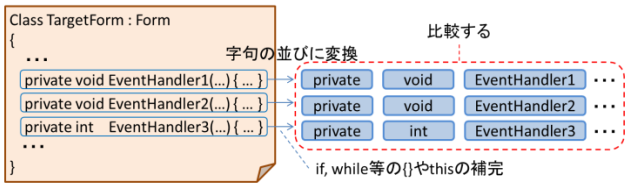

図 10 字句並び比較

コードクローンの検出とイベントハンドラ比較の最大の違いは対象範囲である．コードクローンの検出ではソースコード中で部分的に同一あるいは類似している部分を見つけだすことが目的である．そのため，ソースコードの全体からコードクローンを検出しなければならない．また，コードクローンの大きさはクローンによって異なる．一方，イベントハンドラの比較は比較対象となるイベントハンドラを図 9 の方法より見つけだし，後は比較対象であるイベントハンドラ間で 2 章条件に当てはまるかどうかを判定するのみである．そのため，対象となるのは各イベントハンドラの定義全体となる．

上記より，イベントハンドラの比較はコードクローンの検出よりも簡単であり，より単純な方法で実現できる．そのため，イベントハンドラの比較ではコードクローン検出のように複雑なクローン発見マッチングアルゴリズムは必要とならない．基本的にはイベントハンドラの各字句が全て等しいか比較するのみである．

4. 外部の振る舞いを保つ

本章では，3 章で示した冗長なイベントハンドラの統合手順によって，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，元からある変数や出力の変化内容が統合前と変わらない．

(2) 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．

4.1 同じ内容のイベントハンドラ統合手順

3.1.1 節の同じ内容のイベントハンドラ統合手順では，対象となるイベントハンドラの処理内容が全て同じであるため，統合前と統合後のイベントハンドラの処理内容も当然同じである．また，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．そのため，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．上記より条件(1)は満たされる．

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(2)も満たされ，外部の振る舞いは変わらないことが示せる．

4.2 リテラル，参照が異なるイベントハンドラ統合手順

3.1.2 節のリテラル，参照が異なるイベントハンドラの統合手順では，リテラル，参照が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．ここで返される値が統合前のリテラル，参照と変わっていなければ，4.1 と同様に条件(1)が満たされる．返される値が統合前のリテラル，参照と変わらないことは，2.2 節条件と 3.1.2 節手順の以下の決まりが全て守られることにより保証できる．

A) 統合対象のイベントハンドラに sender 引数(イベント発生元のコントロールの参照)があること

B) 異なっているものが参照である場合に，その参照の参照先を変更している箇所が無いこと

C) 同じコントロールから発生させられるイベントハンドラが複数ある場合，それらにリテラル，参照異なる箇所が無いこと

A)が無ければそもそもこの手法を実現することができないので必要である．注意しなければならないのはユーザ定義のコントロールである．ユーザ定義のコントロールで sender という名前だがイベント発生元のコントロールの参照ではない引数を渡すようなイベントを定義されると，Dictionary メンバから正常な値が帰らない恐れがある．sender 引数は object 型(全ての参照型の基底クラス)の引数でイベントハンドラ内からは sender という引数が本当にイベント発生元のコントロールの参照であるという保証をするのは不可能である． sender 引数がイベント発生元のコントロールの参照であることを保障するためには，イベントを発生させている箇所を確認する必要がある．ただし，実際にこのような問題が起こることはまれであると思われる．

B)はコントロールによって異なるリテラル，参照を登録する処理がインスタンス化の段階でしか呼ばれないために必要となる．もし，参照先が途中で変わるような参照を呼び出すイベントハンドラを統合してしまうと，統合後のイベントハンドラでは参照先が変わらないため，統合前のイベントハンドラと振る舞いが変わってしまう．ただし，実際には参照先を途中で変更するような参照などほとんどない．

C)はクリックした時とダブルクリックした時でリテラル，参照が異なる場合などのケースに対応するために必要となる．

次に重要となる点はコントロールごとのリテラル，参照を登録するタイミングである．登録前にイベントハンドラが実行されると Dictionary 型メンバにアクセスする際に例外が発生してしまう．そのため，登録は早めに行いたい．しかし，各コントロールのインスタンス化の前に登録することはできない． 3.2.2 節の手順vi で InitializeConponent() メソッドの直後で登録するのは，各コントロールのインス

タンス化を行うのがこのメソッドだからである．このメソッドでイベントが発生することはユーザが直接手を加えない限りないので，この直後ならば問題ない．

上記より，Dictionary 型メンバから返される値が統合前のリテラル，参照と変わらないため，4.1 と同様に条件(1)が満たされる．また，条件(2)もインスタンス化の際にコントロールごとのリテラル，参照を登録するメソッドを追加しただけなので満たされる．

5. 評価

3 章で提案した冗長な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 Twitter の連動システム「ぶたツイ」で行った．D2C コンテストはデバイスとクラウドを用いる新たな役立つサービスを提案するコンテストである．

5.1 ぶたツイとは

「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，Twitter 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．表 1 にぶたツイの構成．図 11 にぶたツイとその機能を示す．

表 1 ぶたツイの構成

CPUボード	Armadillo-440
OS	Windows Embedded Compact 7
開発ツール	Visual Studio 2008
言語	C#
.NET	.NET Compact Framework 3.5
行数(空行，コメント行除く)	7006

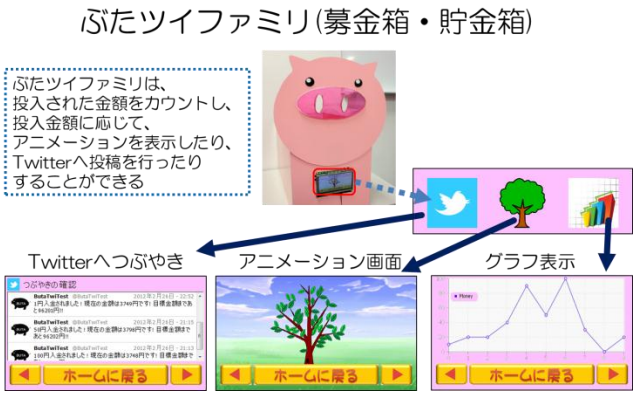

図 11 ぶたツイ

5.2 評価結果

5.2.1 適用箇所

評価はぶたツイの各スクリーンに対して適用することで行う．各スクリーンクラスは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．図 12 にぶたツイのクラス図と適用する箇所を示す

適用により効果が見込めるのはスクリーン遷移メソッドを呼び出すイベントハンドラ(図 13)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(図 14)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 2.1 節の例と同じケースである．

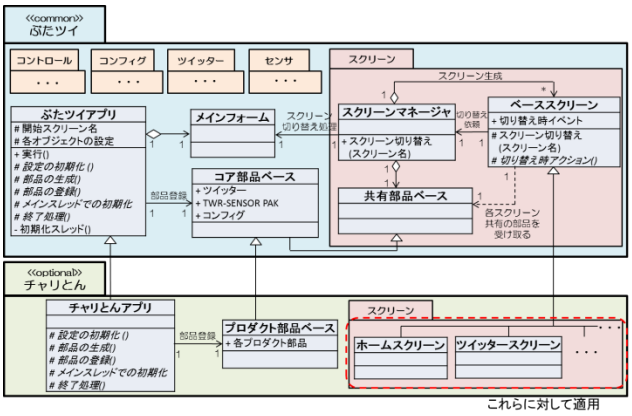

図 12 適用箇所

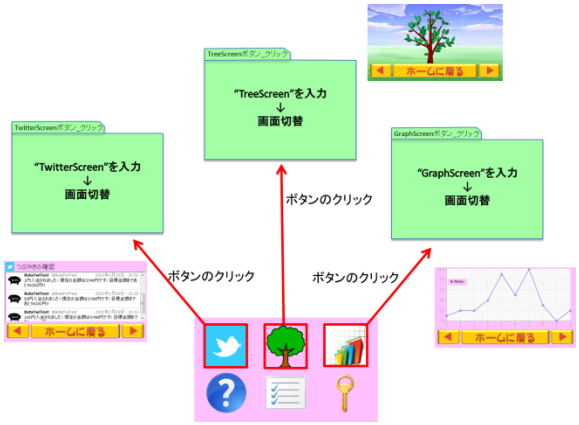

図 13 スクリーン遷移による冗長なイベントハンドラ

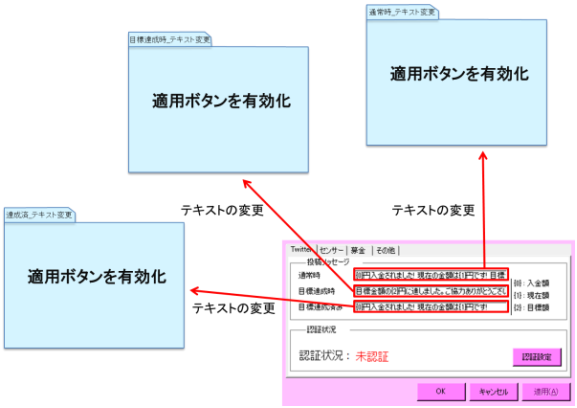

図 14 適用ボタン有効化による冗長なイベントハンドラ

5.2.2 適用結果

適用した結果，行数やイベントハンドラ数がどのように変わったかを表 2 にまとめる．コメント，空行は除いている．行数は思ったより減っていない．これはリテラル，参照の違いを解消するためのコード(Dictionary 型メンバの宣言, Value を示すクラスの定義, 登録のためのメソッド)が，削減されたコードと同等かそれ以上だったためである．一方イベントハンドラ数は 30%も削減された．行数はあまり変わらなかったが，冗長なイベントハンドラが減った分可読性，保守性は向上したといえる．

表 2 適用結果

	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

6. おわりに

本論文では，C#における冗長なイベントハンドラを統合する機能を持ったリファクタリング手法を提案し,「ぶたツイ」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．

今後は，Form や UserControl 以外の組込み特有である冗長なイベントハンドラの事例を調べ，それらの事例に対しても提案手法が有効であるかを評価する．

参考文献

1) 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).
2) W. F. Opdyke : Refactoring Object-Oriented Frameworks, PhD Thesis, University of Illinois at Urbana-Champaign, (1992).
3) 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).
4) 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).
5) 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).
6) L. Prechet, G. Malpohl, M. Phlippsen : JPlad: Finding plagiarisms among a set of programs, Technical Report, Fakultat fur Informatik Universitat Karlsruhe, (2001).
7) 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).
8) 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).
9) 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).
10) 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).
11) R. Komondoor, S. Horwitz : Using slicing to identify duplication in source code, Proc. the 8th International Symposium on Static Analysis, pp.40-56, Paris, France, (2001).
12) J. Krinke : Identifying similar code with program dependence graphs, Proc. the 8th Working Conference on Reverse Engineering, pp.301-309, Suttgart, Germany, (2001).
13) 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, (2012).
14) D2C コンテスト(Device2Cloud Contest) http://www.d2c-con.com/