コンテキスト指向プログラミング実現に向けた実行時プログラム書き換えフレームワークの提案

谷川郁太†1, 小倉信彦†2, 菅谷みどり†3, 渡辺晴美†1

近年，情報技術が生活空間に溶け込むアンビエント情報社会の実現に向け，環境適応型のソフトウェア開発技術が求められている．環境適応を実現するための技術として，コンテキスト指向プログラミング(Context Oriented Programming: COP)がある．COP では，プログラム実行時に，環境の状況すなわちコンテキストに応じて，振る舞いを変化させる．本論文では，COP のプログラム書き換えに関し，(1) 書き換え命令がプログラム中に散在し振る舞いを複雑にする問題，(2) システムの整合性を保つ安全な書き換えは容易ではない問題に着目する．以上を解決するために，我々は，(1) コンテキスト判定部分の独立，(2) 書き換え可能かを判定する条件をアノテーションで付すことを特徴としたフレームワークを提案し，実行時に書き換え可能であることを適用例により示し議論する．

A Framework for Dynamic Program Rewriting at Runtime towards Context Oriented Programming

Ikuta Tanigawa†1, Nobuhiko Ogura†2, Midori Sugaya†3, Harumi Watanabe†1

With the advent of ambient intelligence, the next generation of software systems requires the ability to dynamically adapt to various contexts which are their changes in their surrounding environment. Context oriented programming (COP) treats context explicitly, and provides mechanisms to dynamically adapt behavior in reaction to changes in context at runtime. We will focus on the following problems of COP: (1) behavior of system is too complex to expect, since rewriting program instructions are scattered in the program and (2) as keeping consistency and safety, dynamic program rewriting is difficult. To overcome these problems, we propose a framework which is expressed the following features: (1) supporting independent context decision part and (2) classes or methods are attached with guard conditions on annotation to decide to invoke program rewriting. Finally, to discuss novelty and usefulness, we will apply our framework to a case study.

1. はじめに

近年，情報技術が生活空間に溶け込むアンビエント情報社会の実現に向け，環境適応型のソフトウェア開発技術が求められている[1][2][3]．コンテキスト指向プログラミング(Context Oriented Programming: COP)は，このようなソフトウェア開発を主要な目的とした技術であり，Robert Hirschfeld [4]らの研究が最初である．COP では，プログラム実行時に，システムを取り巻く外部環境の状況すなわちコンテキストに応じて，実行時にソフトウェアを再構築し，振る舞いを変化させることが可能である[4][5][6][7]．

COP においてコンテキスト判定部分は重要な役割を果たす[6][7]．アンビエント情報社会ではコンテキストは多様で複雑であり，システムの実行前に環境状況の変化を網羅することは容易ではない．

例えば，近年のオペレーティングシステムや SNS 等のネットワークアプリケーションは稼働を止めずにソフトウェアのアップデートを実施する．これらのソフトウェアは，リリース前に多様なユーザ環境を十分に予測することは困難であるため，リリース後に誤り情報を収集し，オンラインで日々配信し，実行中に書き換えを行っている.従来の方法では，新しい振る舞いに書き換えることは容易であるが，コンテキストに応じて書き換えが必要かどうかを判定する部分は限定されていず，どこでも書き換え可能である．すなわち書き換え処理がプログラム中に散在することで，振る舞いの予測を難しくする問題がある[7]．COP では実行中に書き換えを行うため，

†1 東海大学
†2 東京都市大学
†3 芝浦工業大学

ⓒ 2014 Information Processing Society of Japan

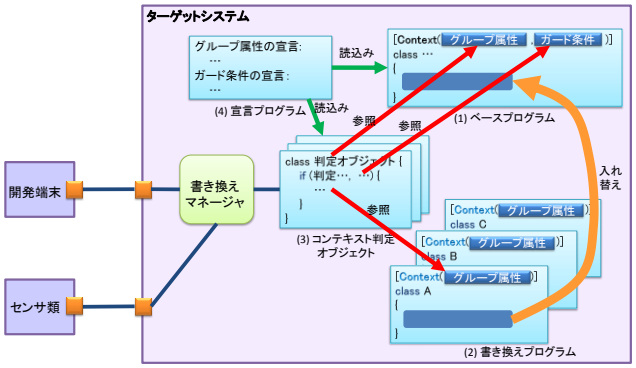

図1 プログラム書き換えフレームワーク

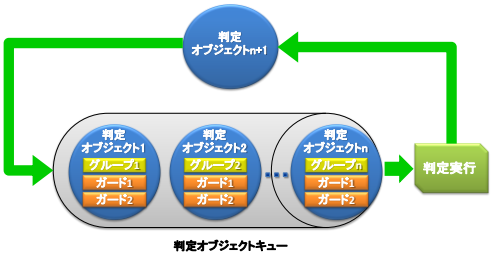

図2 プログラム書き換えの判定方法

システム内の状況が書き換え可能かについて判定できることも重要である．書き換え時には，システムの整合性をたもち，安全な実行を行う必要がある．しかし，COP においては，書き換え時のシステムの振る舞いを予測することが困難であることから，安全な書き換えの実現が難しい問題がある．

以上から，本論文ではコンテキスト判定に関し 2 つの問題に着目する．「(1) コンテキスト判定の散在，(2) 書き換えの安全性」．これらの問題を解決するために，本論文では以下の特徴を持つフレームワークを提案する．

(1)の問題を解決するためにコンテキストの状況を判定し， 書き換え指示を行う部分をコンテキスト判定プログラムとして独立させ，その部分も書き換え可能にする．(2)の問題を解決するために，ベースとなるプログラムのクラスとメソッドにアノテーションを付す．アノテーションには，クラスやメソッドが属するグループや種類を記すグループ属性と，書き換えを実行する際のガード条件を記す．ガード条件により(2)の問題を解決する．

代表的な COP である JCop[5]は，with で指定したレイヤを実行時に切り替えを行うことで，プログラムの切り替えを可能としている．Robert Hirschfeld らは，COP であるためには，振る舞いの多様性，レイヤ，アクティブ化，コンテキスト，スコープに対応していること[4]としている．我々の方法では，アノテーションのグループ属性をレイヤと同じ目的で利用することが可能である．

以上より，我々の方法において従来のコンテキスト指向プログラミング技術に貢献できる点は，コンテキストの状況を判定する部分を独立集中させ，書き換え可能な条件について明示したアノテーションを付すフレームワークの記述，および，実行時に書き換え可能にする仕組みである．また我々のフレームワークは C#で実現し，ベースとなるプログラムには文法的な拡張を加えず，アノテーション記述方法の規定に留まっているため，既存プログラムの再利用，学習の容易さが見込める．さらに，我々は提案フレームワークを支援するツールを，プログラムサイズ 7600 行程度のクラウドと連携した募金箱システムに適用し，通信が途切れずにプログラムが書き変わることを確認した[8]．

以下，2 章 3 章でコンテキスト書き換えについて提案する．2 章ではコンテキスト書き換えフレームワークにおけるプログラムの構成，3 章では，実行時にプログラムを書き換える方法について述べる．4 章では，2 章 3 章で提案した方法を例に適用し評価を行い，5 章でその結果を踏まえ議論する．6 章では，本論文での提案をまとめ，今後の課題について述べる．

2. コンテキスト指向プログラム書き換えフレームワークの提案

本章では，1 章で述べた COP を実現するためのコンテキスト指向プログラム書き換えフレームワークを提案する．本フレームワークは，コンテキストに応じたプログラムの切り替えを容易に行うことを目的とし，コンテキストの判定とその結果に応じて書き換えを行う部分を明確にしたフレームワークである．以降，2.1 節でフレームワークの概要について記し，2.2 節でフレームワークを定義する．

2.1. フレームワークの概要

本フレームワークの構成を図 1，2 にプログラム例を図 3 に示す．本フレームワークで図 1 に示した，あるコンテキストは，(1)ベースプログラムを(2)書き換えプログラムで書き変える．書き換えはコンテキストの状況に応じて行うため，(3)コンテキスト判定オブジェクトでコンテキストごとの状況を判定する．コンテキストオブジェクトによる判定では，判定に用いるコンテキストの状況を宣言する(4)宣言プログラムがあり，これらを管理するマネージャがある．マネージャは開発端末やセンサ類とつながっており，これから発行されるイベントにより，コンテ

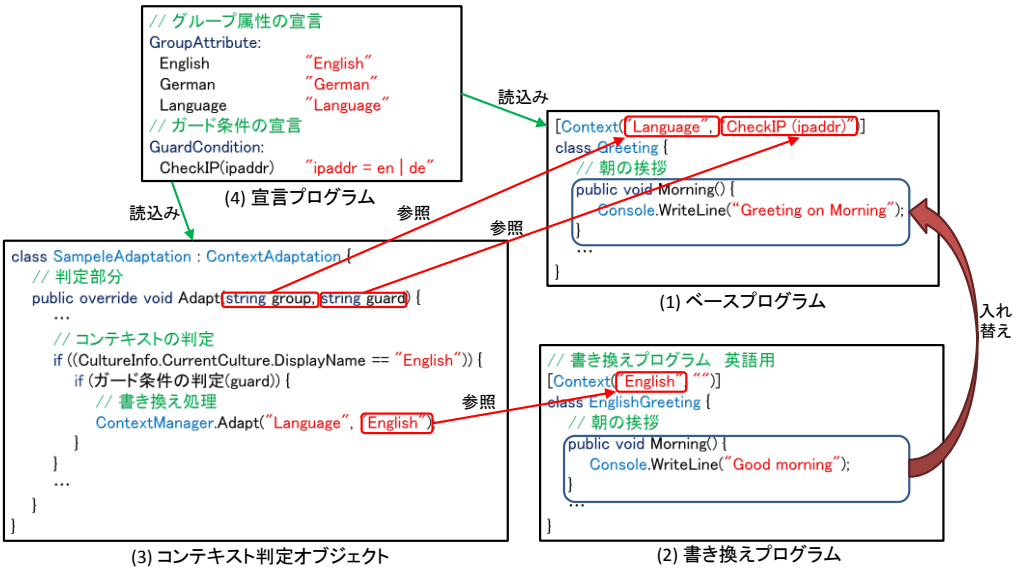

図3 プログラム記述例

キストを判定し，書き換えを行う．コンテキストごとの判定はマネージャが順番に(3)コンテキスト判定オブジェクトを実行することで行っている．図 2 にその様子を示す．グループ属性はクラスやメソッドが属するグループや種類を記すもので，コンテキストの判定やプログラムの書き換えは，このグループ単位で順次行う．ガード条件は安全に書き換えを行うために，クラス・メソッドの実行条件を表す記述である．

2.2. フレームワークの記述

本節では，フレームワーク記述について述べる．図 1 の(1)～(4)に示したプログラムに対応したプログラムの BNF を図 4 の(1)～(4)に示す．(1)(2)は図に示すように同一のフォームで記述する．以下，各々について説明する．

(1) ベースプログラム (2) 書き換えプログラム:

ベースプログラムは書き換えプログラムにより書き換える．プログラムが書き変わるコンテキストの状況と条件を明らかにするために，*Context* アノテーションと呼ぶ記述をクラスとメソッドに付ける．したがって，図 4(1)に示す通り，ベースプログラムと書き換えプログラムは *Context* アノテーションとクラスからなる．さらに，メソッドごとの書き換えも可能にするために，メソッドにも *Context* アノテーションを付けることができる．*Context* アノテーションはグループ属性とプログラム断片名とガード条件からなる．プログラム断片名は書き変えられるクラス・メソッドを明確にするもので，ベースプログラム・書き換えプログラムで名前が一致するクラス・メソッドが書き変えられる．

<Contextクラス> = <Contextアノテーション>, <クラス>;
<Contextメソッド> = <Contextアノテーション>, <メソッド>;
<Contextアノテーション> = "[Context(", <グループ属性>, ",", <ガード条件>, ")]";
<グループ属性> = <文字列>;
<ガード条件> = <文字列>;

(1) ベースプログラム，(2) 書き換えプログラム

<コンテキスト判定オブジェクト> = <クラス>
<クラス> = "class", <識別子>, ":", "ContextAdaptation", <クラスブロック>;
<適応メソッド> = "public override void Adapt(string group, string gurad)", <ブロック>;
<ブロック> = "{", { <ステートメント> | <適応処理> }, "}"
<適応処理> = "ContextManager.Adapt(", <グループ属性>, ",", <グループ属性>, ");";
<グループ属性> = <文字列>;

(3) コンテキスト判定オブジェクト

<宣言プログラム> = { <グループ属性宣言部> | <ガード条件宣言部> };
<グループ属性宣言部> = "GroupAttribute:", { <グループ属性宣言> };
<グループ属性宣言> = <属性名>, ((<属性名>, { ("&" | "|"), <属性名> }) | <文字列>);
<属性名> = <識別子>;
<ガード条件宣言部> = "GuardCondition:", { <ガード条件宣言> };
<ガード条件宣言> = <条件名>, ((<条件名>, { ("&" | "|"), <条件名> }) | <文字列>);
<条件名> = <識別子>, "(", { <識別子>, "," }, ")";

(4) 宣言ファイル

図 4 各プログラムの BNF

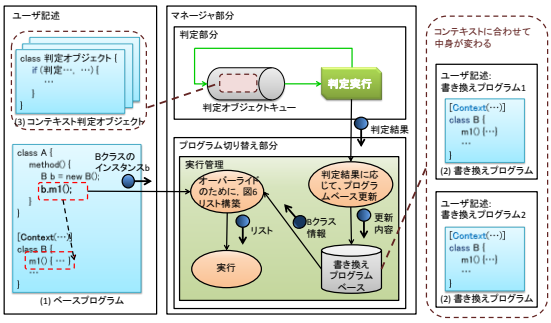

図 5 コンテキストに応じた書き換えの構造

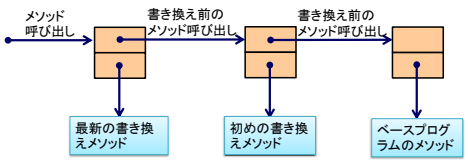

図 6 リストによる，適応前のメソッドの呼び出し

```
1  [Context("Language", "CheckIP(ip)")]
2  class Greeting {
3      public void Morning() {
4          Console.WriteLine("Greeting on Morning");
5      }
6      public void Afternoon() {
7          Console.WriteLine("Greeting on Afternoon");
8      }
9      public void Night() {
10         Console.WriteLine("Greeting on Night");
11     }
12 }
```

(1) ベースプログラム

```
1  class SampeleAdaptation : Adaptation {
2      // 判定部分
3      public override void Adapt(string group, string guard) {
4          if (CultureInfo.CurrentCulture.DisplayName == "English") {
5              if (CheckGuardCondition(guard))
6                  AdaptationManager.Adapt("Language", "English");
7          } else if (CultureInfo.CurrentCulture.DisplayName == "German") {
8              if (CheckGuardCondition(guard))
9                  AdaptationManager.Adapt("Language", "English");
10         }
11     }
12     // ガード条件の判定
13     private bool CheckGuardCondition (string guard) { … }
14 }
```

(3) コンテキスト判定オブジェクト

```
1  // 書き換えプログラム　英語用
2  [Context("English", "")]
3  class EnglishGreeting {
4      public void Morning() {
5          Console.WriteLine("Good morning");
6      }
7      public void Afternoon() {
8          Console.WriteLine("Good afternoon");
9      }
10     public void Night() {
11         Console.WriteLine("Good night");
12     }
13 }
```

```
14 // 書き換えプログラム　ドイツ語用
15 [Context("German", "")]
16 class GermanGreeting {
17     public void Morning() {
18         Console.WriteLine("Guten morgen");
19     }
20     public void Afternoon() {
21         Console.WriteLine("Guten tag");
22     }
23     public void Night() {
24         Console.WriteLine("Gute nacht");
25     }
26 }
```

(2) 書き換えプログラム

```
1  // グループ属性の宣言
2  GroupAttribute:
3    English          "English"
4    German           "German"
5    Language         "Language"
6  // ガード条件の宣言
7  GuardCondition:
8    EnglishIP(ipaddr)    "ipaddr = en"
9    GermanIP(ipaddr)     "ipaddr = de"
10   CheckIP(ipaddr)      EnglishIP(ipaddr) | GermanIP(ipaddr)
```

(4) 宣言プログラム

図7 適用例: 挨拶プログラム

(3) コンテキスト判定オブジェクト:

コンテキスト判定オブジェクトは，書き換えを行うためにグループ属性ごとにコンテキストの状況を判定するオブジェクトである．コンテキストの判定，ガード条件の判定と書き換え処理は図 4(3)の *Adapt* メソッドで記述する．*Adapt* メソッドでは引数として，グループ属性とガード条件がわたされるのでそれら用いて判定を行う．書き換えはグループ単位で行うので，書き換えの前後を表すグループ属性をわたすことで行う．

(4) 宣言プログラム:

宣言プログラムは(1)～(3)のプログラムで用いられるグループ属性・ガード条件を宣言するプログラムである．グループ属性は名前を付けることで宣言する．ガード条件は関数形式で宣言し，対応する条件式を記述する．関数形式になっているため，任意の変数に対して，ガード条件の判定が行える．また，グループ属性・ガード条件は，前で宣言したものを AND，OR で組み合わせて用いることができる．

3. プログラムの実行時書き換え

本章では，プログラムの実行時書き換えについて述べる．2 章より，本フレームワークでは，以下の時にプログラム書き換えが行われる．

(a) コンテキストの変化に適応するための書き換え
(b) システム実行中に新たなプログラムを送ることによる書き換え

以降，各節で(1)(2)の書き換え方法について述べる．

3.1. コンテキストに応じた書き換え

(1)を実現するための書き換え構造を図 5 に示す．本フレームワークのマネージャ部分では，コンテキストの判定を行う部分とプログラム切り替えを行う部分がある．コンテキストに応じた書き換えを実現するため，プログラム切り替え部分では，メソッド呼び出し時にコンテキストに応じた書き換えプログラムを用いて，メソッドをオーバーライドする．オーバーライド実現のために，書き換えプログラムを保持する書き換えプログラムベースがある．ここに保持される書き換えプログラムは，判定部分で判定した結果，有効になった書き換えプログラムであり，メソッド呼び出し時にこれらを用いオーバーライドする．

さらに，(1)の書き換えにおいて複数のコンテキストにより書き換えが重ねられた際，書き換え内容を全て有効にするために，書き換えプログラム上の記述を書き換え前の差分のみにしなければならない．そのため，書き換えプログラムで書き換え前のメソッドを呼び出す仕組みが必要となる．

上記を考慮すると(1)の書き換えは，メソッド呼び出しを図 5 の実行管理のところを介して行うようにし，そこで書き換えプログラムのクラス情報から生成したインスタンスとメソッド呼び出しが行われたインスタンスによって，図 6 に示すリストを構築する．リストを構築する理由は，最新の書き換えメソッドからベースプログラムまでを辿ることが可能な書き換え前メソッド呼び出しを実現するためである．これにより，書き換えを重ね複数のコンテキストに対応できる．リスト構築後，最初のノードからメソッドを呼び出し，書き換え前のメソッド呼び出しが行われるたびに，実行管理オブジェクトからリストを辿り，次のノードのメソッドを呼び出す．

ⓒ 2014 Information Processing Society of Japan

```
1  // 書き換えプログラム　イタリア語用
2  [Context("Italian", "")]
3  class ItalianGreeting {
4      public void Morning() {
5          Console.WriteLine("Buongiorno");
6      }
7      …
8  }
```

(2) 書き換えプログラム

```
1  class SampeleAdaptation : Adaptation {
2      public override void Adapt(string group, string guard) {
3          if …
4          // イタリア語環境の判定
5          } else if (CultureInfo.CurrentCulture.DisplayName == "Italian")) {
6              if (CheckGuardCondition(guard))
7                  AdaptationManager.Adapt("Language", " Italian");
8          }
9          …
10     }
11 }
```

(3) コンテキスト判定オブジェクト

```
1  // グループ属性の宣言
2  GroupAttribute:
3    …
4    Italian              "Italian"
5  // ガード条件の宣言
6  GuardCondition:
7    …
8    ItalianIP(ipaddr)    "ipaddr = it"
9    CheckIP(ipaddr)      EnglishIP(ipaddr) |
10                        GermanIP(ipaddr) |
11                        ItalianIP(ipaddr)
```

(4) 宣言プログラム

図8 適用例: 追加プログラム

3.2. 新たなプログラムを送ることによる書き換え

マネージャでは，開発端末と通信し新たに送信されたプログラムの受け取りを行う部分がある．新たなプログラムを受け取った場合に，該当するプログラムを入れ替えることで(2)の書き換えを実現する．プログラムはコンパイル後のランタイムライブラリの形式で送り，ロードするライブラリを変更することによって入れ替える．実行中のプログラムを入れ替えないために，コンテキスト判定，書き換えメソッド実行では排他制御を行う．

4. 適用

本章では，システムが所在する地域の言語で挨拶を表示するプログラム（以下，挨拶プログラム）を例に，2 章 3 章で提案したコンテキスト書き換えフレームワークを適用する．以下 4.1 では挨拶プログラムの説明，4.2 ではフレームワークへの適用について述べる．

4.1. 適用例: 挨拶プログラム

適用例の挨拶プログラムは，英語環境で実行した場合は”Good morning”と出力し，ドイツ語環境で実行した場合は”Guten morgen”と出力する．図 7 に適用例の挨拶プログラムを示す．また，図 8 では実行時のプログラム変更の例として，イタリア語対応のプログラムへ書き換えについて示す．

4.2. コンテキスト書き換えフレームワークの適用

本節では，挨拶プログラムにおけるコンテキスト書き換えフレームワークの適用箇所について述べる．以降，図 7 の(1)～(4)と図 8 について記す．

(1) ベースプログラム:

ベースプログラムには，書き換え前の *Greeting* クラスの振る舞いを記述する．*Greeting* クラスは言語によって書き変えられるグループである．図 4(1)の BNF に従い，図 7(1)の 1 行目に *Context* アノテーションを付け，グループ属性を”Language”としている．プログラム断片名は書き換えプログラムで書き換える対象を指定するために必要なもので，クラス名と同名の”Greeting”としている．ガード条件では，IP アドレスが該当する国のものである時のみ書き換えを許可するために，”CheckIP(ipaddr)”を与える．

(2) 書き換えプログラム:

書き換えプログラムには，英語環境，ドイツ語環境の際の *Greeting* クラスの振る舞いを記述する． 本プログラムで記述するクラスは，英語環境，ドイツ語環境の際に書き変えに用いられるグループに属する．図 4(1)の BNF に従い，図 7(2)の 2 行目，15 行目にそれぞれ，”English”，”German”をグループ属性とする *Context* アノテーションを記述する．これらのクラスは *Greeting* クラスの書き変えに用いるため，プログラム断片名には (1) ベースプログラムの *Greeting* クラスと同様に”Greeting”と記述する．

(3) コンテキスト判定オブジェクト:

コンテキスト判定オブジェクトは，書き換えを行うためにグループ属性ごとにコンテキストの状況を判定するオブジェクトである．例では，言語ごとにプログラムの振る舞いが変わるので，”Language”グループ属性における判定を行うコンテキスト判定オブジェクトを記述する．英語環境，ドイツ語環境の時に書き換えるので，それぞれの判定を4～7行目， 7～10行目で行っている．コンテキストの判定後の書き換えは図 4(3)の BNF に従い，ベースプログラム，書き換えプログラムのグループ属性をわたすことで行う．

(4) 宣言プログラム:

宣言プログラムでは，(1)～(3)で使うグループ属性・ガード条件を宣言する．図 4(4)の BNF に従い，2～5 行目でグループ属性，7～10 行目でガード条件を宣言している．ガード条件”CheckIP(ip)”はその前で宣言された” EnglishIP(ip)”と” GermanIP(ip)”を OR で組み合わせて宣言している．

実行時のプログラム変更:

実行時のプログラム変更により，コンテキスト判定や，コンテキストに対応するシステムの振る舞いを追加・変更することができる．例では，挨拶プログラムをイタリア語に対応させるために，図 8 の(2)書き換えプログラム，(3)コンテキスト判定オブジェクト，(4)宣言プログラムをフレームワークに送る．(2)の書き換えプログラムには，イタリア語環境の際の振る舞いを記述する．記述の方

ⓒ 2014 Information Processing Society of Japan

法は図 7(2)と同様である. (3)コンテキスト判定オブジェクトと(4)宣言プログラムではイタリア語対応のために, 図 7 の(3), (4)のプログラムに新たな判定やグループ属性, ガード条件を追加する. 図 8(3)の 7 行目の書き換えでは, 新たに追加した"Italian"グループ属性を用いているため, ベースプログラムが図 8(2)の書き換えプログラムに書き換えられる.

5. 議論

本章では, 1 章で述べた本研究の新規性・有用性について考察し, COP として必要な機能を有しているかについて議論する. まず前者についてであるが, 1 章で述べたとおり, 本研究の特徴は下記である. 4 章の適用例を踏まえ, 以下で議論する.

(1) コンテキスト判定部分の散在問題

図 7(3)と図 8(3)に示すとおり, コンテキスト判定部分を集中させ, その部分を図 8 のプログラムで書き変えることでイタリア語環境への対応できることを示した.

(2) 書き換えの安全性

挨拶プログラムの振る舞いが言語に依存していることを示すために, 図 7(1)ベースプログラムと(2)書き換えプログラムでアノテーションを用いてガード条件として IP アドレスより, プログラムの書き換え前に,想定されている国のものであるかを判断可能であることを示した.

(3) C#オリジナルの文法による記述

本フレームワークでは, 文法的な拡張は行わず, C#オリジナルの文法を用い, 既存ソフトウェアの移植が容易であり, 教育時間の短縮を見込んでいる. 挨拶プログラムから C#の文法を拡張していないことが分かる.

次に,COP として必要な機能を有しているかについて述べる. COP では, コンテキストに応じて構造を再構築し, 振る舞いを変えることができる. これは, 環境ごとの振る舞いをレイヤで管理し, 環境に応じてレイヤを切り替えることで実現している. 本フレームワークでは, コンテキストを判定し, クラスやメソッドをグループ単位で書き変えることが可能である. クラスにグループ属性のアノテーションを付し, このグループをレイヤとして見立て, 切り替えが可能である. 従って, レイヤによる書き換えと目指しているところは同じであり, COP と同様にコンテキストに応じた構造を再構築はできることから, 同様の目的を果たすことが可能である.

6. おわりに

本論文では, アンビエント情報社会の実現に向け, 環境適応型ソフトウェア開発技術が求められていることを述べ, 上記を実現する技術として COP を紹介した. COP の問題として, 「(1) コンテキスト判定の散在, (2) 書き換えの安全性」2 つに着目した. これらの問題を解決するために, 我々は以下の特徴を持つフレームワークを提案した. (1)の問題を解決するために, コンテキスト判定部分を独立させた. (2)の問題を解決するために, 書き換え対象となるクラスとメソッドに書き換える単位となるグループに加え, 書き換え可能かどうかを判定するためのガード条件を付した. 提案方法を挨拶プログラムに適用し, これらの問題を解決しているか, コンテキスト指向と言えるかどうかについて議論した. 今後は, コンテキストに応じた書き換えをより柔軟に, 安全に行えるためにフレームワークの記述を発展させたい.

参考文献

[1] N. Shadbolt: Ambient Intelligence, IEEE Intelligent Systems, Volume 18 Issue 4, pp. 2-3 , 2003.

[2] T. Ruiz-López, C. Rodríguez-Domínguez, M. J. Rodríguez, S. F. Ochoa, J. L. Garrido: Context-Aware Self-adaptations: From Requirements Specification to Code Generation, Ubiquitous Computing and Ambient Intelligence. Context-Awareness and Context-Driven Interaction, LNCS 8276, pp. 46-53, 2013.

[3] T. Rho, M. Appeltauer, S. Lerche, A. B. Cremers, and R. Hirschfeld: A Context Management Infrastructure with Language Integration Support, In Proceedings of the Third International Workshop on Context-Oriented Programming, COP'11, ACM Press, pp. 1-6 , 2011.

[4] R. Hirschfeld, P. Costanza and O. Nierstrasz: Context-oriented Programming, Journal of Object Technology, Vol. 7, No. 3, pp. 125-151, 2008.

[5] M. Appeltauer, R. Hirschfeld, J. Lincke,: Declarative Layer Composition with the JCop Programming Language, Journal of Object Technology, Vol. 12, No. 4, 2013.

[6] S. González, K. Mens, M. Col˘ acioiu and W. Cazzola: Context Trait, AOSD2013, pp. 209-220, 2013.

[7] M. Appeltauer, R. Hirschfeld, H. Masuhara, M. Haupt,and K. Kawauchi: Event-based Software Composition in Contextoriented Programming, "In Proceedings of the 9th International Conference on Software Composition, LNCS6144, pp. 50-65", 2010.

[8] 谷川 郁太, 小倉 信彦, 菅谷 みどり, 渡辺 晴美: 組込みソフトウェアプロトタイプ開発のためのプログラム動的書き換え, 研究報告システム LSI 設計技術(SLDM), Vol.35, pp. 1-4, 2014.

ⓒ 2014 Information Processing Society of Japan