Vol.2014-SLDM-165 No.35
Vol.2014-EMB-32 No.35
2014/3/16

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

谷川郁太[†1]　小倉信彦[†2]　菅谷みどり[†3]　渡辺晴美[†1]

システム開発の初期段階では，システムの実装方法を試すことや完成予想を可視化すること等を目的として，プロトタイプ開発が行われることがある．プロトタイプ開発では上記を目的としているため，短期間で頻繁にプログラムの書き換えが行われる．組込みシステムではプログラム修正のたびに，機器を停止させプログラムを再度アップロードすることとシステム停止前と同じ状況を再現することにコストがかかるため，プログラムの書き換えの頻度が増えることは問題となる．本稿では上記の問題を解決するためのプログラム動的書き換え環境を提案する．本環境では開発端末からのウィーブ記述によって，稼働中の組込みシステムプログラムの部分書き換えを行い，プログラムの再起動，アップロードにかかるコストを削減する．また，書き換え前の状態を保持し，書き換え前と同じ状況で変更後のプログラムを実行することを可能とする．評価は掃除機ロボットの開発に適用することで行う．

Dynamic Program Rewriting at Runtime for Embedded Software Prototype Development

IKUTA TANIGAWA[†1]　NOBUHIKO OGURA[†2]
MIDORI SUGAYA[†3]　HARUMI WATANABE[†1]

Prototyping methods on embedded software are useful in initial stage of development process for evaluating algorithms of system behavior by trial and error. The period for the development process is desired to be short as possible. Whenever rewriting programs, we need to restart the system. For restarting, we must stop system, upload software, and occasionally link network. We meet often opportunities of rewriting programs. Therefore, the cost for rewriting is not small. Moreover, the resuming system is impossible to behave as same states before stopping, since embedded software receives influence from various and complicated environments. To overcome these restarting problems, the article presents a prototyping method by dynamic rewriting program at runtime. For evaluation of our method, it is applied to development of an automatic vacuuming robot.

1. はじめに

システム開発の初期段階では，システムの実装方法を試すことや完成予想を可視化すること等を目的として，プロトタイプ開発が行われることがある．プロトタイプ開発では上記を目的としているため，短期間で頻繁にプログラムの書き換えが起こる．度重なるプログラムの修正は，システムの再起動によるコスト増加をもたらすことは既知である．特に組込みシステムでは，プログラム修正のたびに機器を停止させ，プログラムを再度アップロードするためのコストがかかる．また，組込みシステムは外界からの影響を受けるため，システム停止前と同じ状況を再現することが難しいため、何か問題が起きたところで、システムの現状を保持したまま、プログラムの書き換えができることは重要である。さらに、プロトタイプ開発に置ける気づきは非正常系である可能性が高く、機能に横断的な処理もある。

本稿では、上記の問題を解決するために，プログラムを実行時に書き換えが可能な動的書き換え環境を提案する．本環境では開発端末からのウィーブ記述によって，稼働中の組込みシステムプログラムの部分書き換えをシステム横断的に行う．プログラムの動的書き換えによって，プログラムの再起動, アップロードにかかるコストが削減される．また，書き換え前の状態が保持されるため，書き換え前と同じ状況で変更後のプログラムを実行することが可能となる．提案環境は、C#のアノテーションを利用し、アスペクト指向の概念を実現する。また、組込みシステムであるターゲットに監視プログラムを配置することで実現する。

本研究と関連し、実行時に横断的関心事を扱える言語にダイナックアスペクト指向プログラミングが提案されている 1)2)をはじめ多く提案されている．また、C#のための動的開発言語としては loom.net 3)4)は実用的である。これらのプログラミング言語は分散システム向けであることが多い。一方、組込みシステムのプロトタイプ開発に向けて、スクリプト言語を利用した方法も近年国内で提案されている 5)6).

以降，2 章ではプログラム動的書き換え環境の提案を行い，3 章ではプログラム動的書き換え環境の実現方法について述べる．4 章で掃除機ロボットの開発に適用する．5 章でまとめと今後の課題について述べる．

†1 東海大学大学院 情報通信学研究科
Tokai University Graduate School of Information and Telecommunication Engineering
†2 東京都市大学 環境情報学部
Tokyo City University, Faculty of Environmental and Information Studies
†3 芝浦工業大学
Shibaura Institute of Technology

Vol.2014-SLDM-165 No.35
Vol.2014-EMB-32 No.35
2014/3/16

2. プログラム動的書き換え環境の提案

提案環境は，稼働中の組込みシステムのプログラム書き換えを開発端末から動的に行う．環境利用者はプログラムの作成時にそれぞれのメソッドに C#の属性を用いてマークを付ける．マークはメソッドの付加情報であり，通信に関連するなら”Communication”，デバッグ用のメソッドには”Debug”といった形で付ける．プログラム書き換えは書き換え先を上記のマークを用いて指定し，そこに対して書き換えたいプログラムをウィーブすることで実現する．これにより，ユーザは関連する機能や用途ごとに書き換えることが可能となる．

2.1 構成

図 1 にベースプログラムの作成から実行までの流れを示す．ベースプログラムは変更が加えられる前のプログラムで，前述のマークが付けられている．これをマーク解釈器に入力すると，マークの付けられたメソッドにウィーブされるプログラムを実行するための処理が埋め込まれる．埋め込まれる処理の詳細は 3.4 節で述べる．上記の変換後プログラムを C#コンパイラによりコンパイルし，生成されたアセンブリをウィーブに関する機能を提供するライブラリやウィーブされるプログラムのためのコンパイラと一緒にターゲットシステムにアップロードし，実行する．

実行時の動的書き換えの流れを図 2 に示す．図中の色付けされている要素は，図 1 の同じ色が付いた要素と対応関係にあることを示している．図の横断的なプログラムはそれぞれのメソッドにウィーブされるプログラムで，ウィーブ記述は横断的なプログラムをウィーブする箇所を指定するための記述である．一般的なアスペクト指向言語では，ウィーブ箇所とウィーブされるプログラムの記述は同一のファイルで行われるが，提案環境では再利用性を考慮して両者を分離している．上記の横断的なプログラムとウィーブ記述をターゲットシステムが理解可能な形式に変換し，シリアルデータとしてターゲットシステムに送る．ターゲットシステム側ではウィーブされるプログラムがシリアルデータとして送られてくるのを監視しており，シリアルデータが送られればウィーブを開始する．この際に，ウィーブされるプログラムはターゲットシステムに搭載されたコンパイラによって実行可能な形にコンパイルされる．ウィーブ後のプログラムはウィーブ前の状態を保持し，システムを再起動することなく実行される．

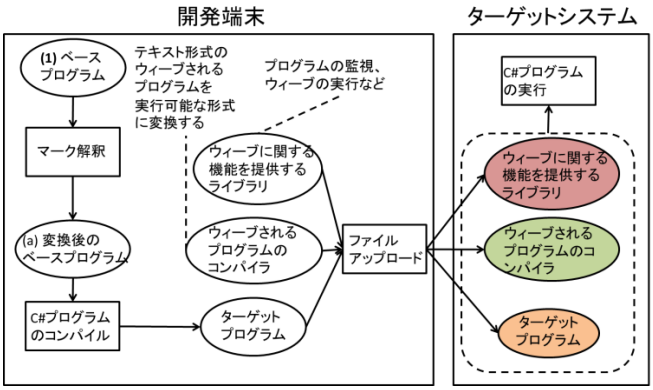

図 1　ベースプログラムから実行のプロセス

Figure 1　Process from base program to execution.

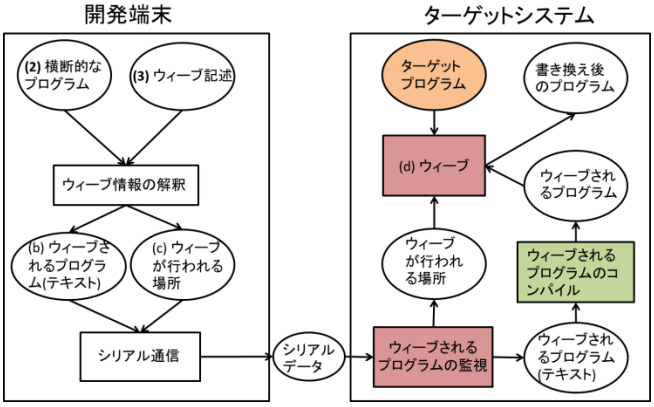

図 2　実行時のプログラム動的書き換えのプロセス

Figure 2　Process of rewriting program dynamically at runtime.

2.2 記述例

本節では，前節で述べたベースプログラム，横断的なプログラム，ウィーブ記述についての記述例を示す．それぞれ，前節の図 1，2 の(1)～(3)に対応している．

(1)の変更を加える前のベースプログラムの記述例を図 3 に示す．このプログラムは C#で書かれており，各のメソッドに属性を用いてマークを付けている．例では，SNS にツイートを行うメソッドに”Communication”マークを付け，ターゲットシステムに硬貨が入れられた際に実行されるメソッドには”Sensor”マークを付けている．ウィーブ記述では，ここに付けられたマークを基にウィーブ先を指定する．

(2)のターゲットシステムにウィーブされるプログラムの記述例を図 4 に示す. 記述は C#のクラス定義と同じ方法で行うが，”CrossCut”クラスを継承する必要がある．”CrossCut”クラスの”Run”メソッドをオーバーライドしたメソッドがウィーブ対象となるメソッドから呼ばれることとなる．以降ウィーブ対象となるメソッドをターゲットメソッドと呼ぶ．引数の”ctx”にはターゲットメソッドから得られる各種情報が格納されており，例ではターゲットメソッドの名前を参照している．

(3)のウィーブ記述の文法を図 5 の拡張 BNF に示す．ウィーブ記述では，マーク宣言とアタッチ，デタッチの 3 つの操作を行う．マーク宣言では，既存のマークを AND,

Vol.2014-SLDM-165 No.35
Vol.2014-EMB-32 No.35
2014/3/16

OR などを用いて組み合わせた新たなマークを宣言することができる．ここで宣言したマークは，アタッチやデタッチの操作で用いることができる．アタッチ，デタッチはマークによって指定されたメソッドに横断的プログラムを付け外しするための操作である．図 6 にウィーブ記述によるマーク宣言とアタッチの例を図 6 に示す．例では，”Communication”と”Sensor”のマークを OR によって組み合わせた新たなマーク”targetMark”を宣言し，次のステートメントでそのマークが示すメソッドの最後に図 4 の ”LogProcess” をアタッチしている．これにより，”Communication”または”Sensor”のマークが付いたメソッドに”LogProcess”が挿入される．

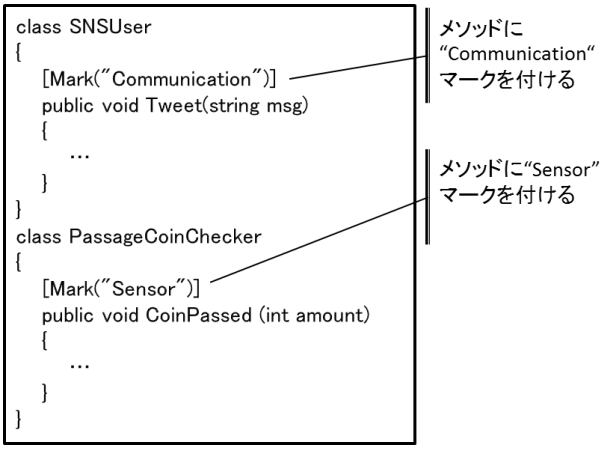

図 3　ベースプログラム

Figure 3　A base program

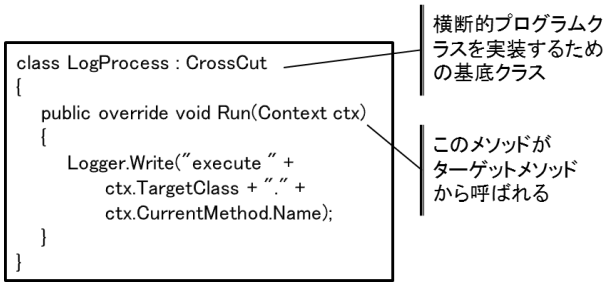

図 4　横断的プログラム

Figure 4　Cross cutting programs

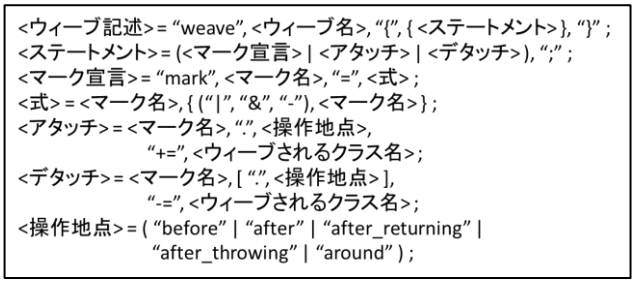

図 5　ウィーブ記述の拡張 BNF

Figure 5　EBNF of weaving program description

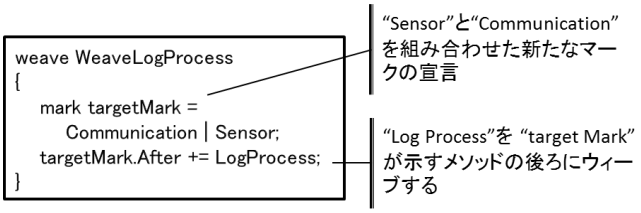

図 6　ウィーブ記述

Figure 6　Weaving program description

3. 実現方法

本章では，前章の図 1，2 の(a)～(d)のについての詳細について記し，提案環境の実現方法を示す．

(a)の変換後のベースプログラムは，図 7 に示す形で変換される．例は図 3 のベースプログラムを変換した後の結果で，ベースプログラムの SNSUser と PassageCoinChecker クラスを継承したクラスを定義している．継承したクラスではマークが付けられていたメソッドをオーバーライドして，オーバーライドしたメソッドにウィーブされた処理を実行するための仕組みが実装される．具体的には，メソッド名やターゲットの参照などを保持するコンテキストの初期化と各マークにウィーブされた処理を実行するポイントを追加する．ウィーブされた処理を実行するポイントを本論文ではジョインポイントという．オーバーライドしたメソッドに上記の仕組みを実装する理由は元となるメソッドを保持し，Around を実現できるようにするためである．Around は AspectJ 等の既存のアスペクト言語と同様にウィーブされるプログラムからターゲットメソッドを呼び出すことができる．

(b)のテキスト形式であるウィーブされるプログラムは，図 4 で示されるような横断的プログラムをスクリプトの形に置き換えたものである．図 8 に上記スクリプトを示す．スクリプト言語には Python が用いられており，これがターゲットシステムに届いたら，ターゲットシステム側で解釈，実行される．

(c)のウィーブが行われる場所は図 6 で示したようなウィーブ記述の情報を基にどのマークにどの処理が追加されるかという情報を XML 形式で記したものである．図 9 に図 6 のウィーブ記述により得られた上記ファイルを示す．ウィーブ記述では，”Communication”または”Sensor”のマークが付いたメソッドの最後に”LogProcess”をアタッチしている．図 9 の XML は上記のウィーブ記述に対応した形で生成されており，”Communication”と”Sensor”に”LogProcess”をアタッチする指示がそれぞれ記述されている．

最後に(d)のウィーブの実現方法について述べる．図 10 はウィーブの実現方法を示すクラス図である．” Weaver” クラスは図 7 の” Weaver” クラスと同一のものである．” Weaver” は各マークのジョインポイントを保持している．また，ウィーブされるプログラムが来ることを監視してお

Vol.2014-SLDM-165 No.35
Vol.2014-EMB-32 No.35
2014/3/16

り，ウィーブされるプログラムが来たら，各マークのジョインポイントに受け渡す．各マークのジョインポイントには，Before，Around，After ごとにウィーブされた処理を格納するための配列があり，Before，Around，After メソッドが呼ばれた際に，配列に格納された処理を全て実行する．ウィーブされた処理は図 7 に示される形で呼び出される．

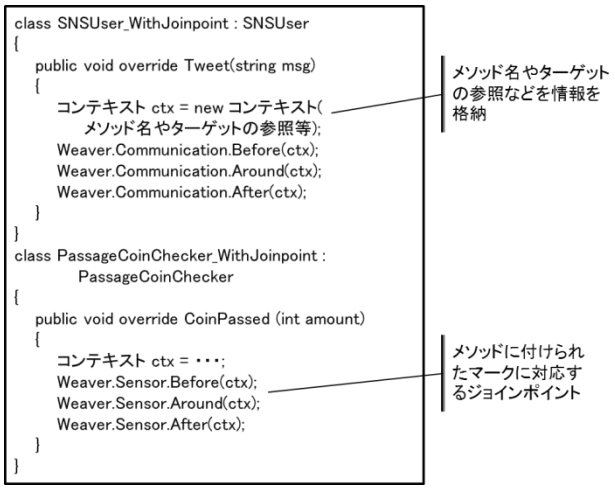

図 7　変換後のベースプログラム

Figure 7　A transformed base program

```
class LogProcess(CrossCut) :
    def Run(self, ctx) :
        Logger.Write("execute " +
            ctx.TargetClass + "." +
            ctx.CurrentMethod.Name)
```

図 8　ウィーブされるプログラム(テキスト)

Figure 8　Woven programs (text format)

```
<weave name="WeaveLogProcess">
  <mark name="Communication">
    <point type="after">
      <attach crosscut="LogProcess" />
    </point>
  </mark>
  <mark name="Sensor">
    <point type="after">
      <attach crosscut="LogProcess" />
    </point>
  </mark>
</weave>
```

Communicationマークに対する操作

LogProcessのアタッチ

図 9　ウィーブが行われる場所

Figure 9　Places for weaving programs

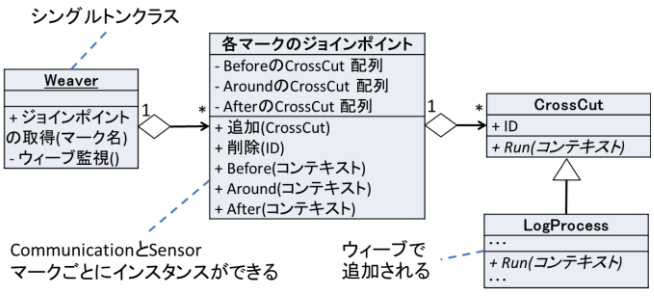

図 10　ウィーブのクラス図

Figure 10　Class diagram of weaving

4. 掃除機ロボット開発への適用

本章では、提案環境を用いた掃除機ロボット開発の流れを示す。

開発対象の掃除機ロボットは, PC とのシリアル通信により制御することが可能である．上記の通信において，PC からはロボットを動かすための指示を送り，ロボットからは各種センサの情報を PC に送る．図 11 にロボット制御システムの構成を示す．本システムは掃除機ロボットとの通信を実現する機能を持ち，ロボットの振る舞いを決定するプログラムを基に通信を行うシステムである．本システムにより実現したマップ作成システムの振る舞いを決定するプログラムに対して提案環境を適用する, また図 12 にその様子を示す．

提案環境を適用した結果，掃除機ロボットとの通信を確立し直すことなく，書き換え後の処理が実行された．

図 11　通信システムの構成

図 12　提案環境の適用

5. おわりに

本稿では、組込みソフトウェア開発の初期段階においてアルゴリズム選定に役立つためのプロトタイプ開発支援環境について紹介した。本システムは C#に、横断的関心事を扱えるようにアスペクト指向の概念を、C#のアノテーションの機構を用い拡張する。このアノテーションをマークと呼ぶ。実行時に書き換えるプログラムを監視するプログラムをターゲットに実装しておくことで､監視プログラムは、書き換えプログラムが送られてくると、マークが指定した箇所を書き換える。本環境を評価するために自動掃除機ロボットへの適用を実施した。本適用ではターゲットである掃除機ロボットとの通信を途切れることなくプログラムを書き換えることを示した。

今後は、メモリや性能にも考慮しながら、実用的な組込みシステム開発環境を目指していきた。

参考文献

1) D.Ansaloni, W. Binder, P. Moret, A. Villazon: Dynamic Aspect-Oriented Programming in Java: The HotWave Experience Transactions on Aspect-Oriented Software Development IX Lecture Notes in Computer Science Volume 7271, 2012, pp 92-122.
2) A. Popovici, T. Gross and G. Alonso: Dynamic Weaving for Aspect-Oriented Programming,, In Proceedings of the 1st International Conference on Aspect-Oriented Software Development, ACM Press, pp.141--147, 2002.
3) A. Rasche, W. Schult, A. Polze: Self-adaptive multithreaded applications: a case for dynamic aspect weaving , ARM '05 Proceedings of the 4th workshop on Reflective and adaptive middleware systems Article No. 10 , 2005.
4) LOOM.NET
http://www.rapier-loom.net/
5) 志田駿介, 菅谷みどり, 倉光君郎: Tiny Konoha: ET ロボコン向けのスクリプト処理系の簡素化, 組込みシステムシンポジウム 2012 論文集, pp.31-38, 2012.
6) 松本 亮介, 岡部 寿男: 組み込みスクリプト言語 mruby を利用した Web サーバの機能拡張支援機構, 情報処理学会研究報告 IOT, インターネットと運用技術 2012-IOT-18(6), 1-6,2012.