コンテキスト指向プログラミングにおける優先度に応じたレイヤスケジューリング手法の提案

谷川 郁太[1,a)]　久住 憲嗣[1]　小倉 信彦[2]　菅谷 みどり[3]　渡辺 晴美[4]　福田 晃[1]

概要：コンテキストアウェアなソフトウェアの開発技術として，コンテキスト指向プログラミングがある (Context-Oriented Programming: COP)．COP はコンテキストを明示的に扱い，実行時のコンテキストの変化に動的に適応するためのメカニズムを提供する．COP はコンテキストアウェアな組込みシステムの実現のために期待できる．COP による振る舞い変更は，横断的関心事を扱うことができ，サービスの切り替えのみではなく，デバイス故障や通信途絶といった非正常系の処理への適用も有用である．これら非正常系の処理の変更は，事故の発生や製品の故障を防ぐために，正常系の処理の変更よりも優先して行われる必要がある．我々は，上記のために，振る舞い変更の優先度を，適応すべき状況ごとに設定する仕組みを備えた，COP フレームワークを開発した．本フレームワークでは，振る舞い変更のための処理の最中に，より優先度の高い変更の要求がある場合に，現在実行中の変更処理を中断して，優先度の高い変更を行う．本稿では，優先度に応じた振る舞い変更のための仕組みを提案し，振る舞い変更にかかる時間に関して適用・評価する．

キーワード：コンテキスト指向プログラミング，組込みシステム，動的書き換え

Layer Scheduling Method with the Layer Priority for Context-Oriented Programming

Tanigawa Ikuta[1,a)]　Hisazumi Kenji[1]　Ogura Nobuhiko[2]　Sugaya Midori[3]　Watanabe Harumi[4]
Fukuda Akira[1]

Abstract: Context-oriented Programming (COP) is an approach that is suitable for context-aware software development. COP treats context explicitly and provides mechanisms to adapt dynamically to the changes in the context at runtime. We believe that the approach could apply the application of context-sensitive embedded systems. Embedded systems often contain abnormal controls such as a device failure or a communication disconnection. COP separate the cross-cutting concerns like abnormal controls from a normal one. A changing from normal to abnormal control requires reducing response time to prevent accidents or product failure. This paper proposes the preemptive layer scheduling method with the layer priority for COP. In the method, the system activates in order from higher to lower priority, and when high priority request has arrived, the system interrupts the lower priority layer controls and activates higher one. Finally, we evaluate the performance of the changing behavior.

Keywords: Context-Oriented Programming, Embedded System, Dynamic Rewriting

1. はじめに

近年，組込みシステムにおいて環境適応型のシステムが求められている [1]．環境に応じた全体書き換えの問題を解決するプログラミング技術として，コンテキスト指向プログラミング (Context-Oriented Programming: COP) がある [2][3][4][5]．COP はコンテキスト (Context) に依存した振る舞いをレイヤとしてモジュール化し，実行時のコン

[1] 九州大学
Kyushu Uniersity
[2] 東京都市大学
Tokyo City University
[3] 芝浦工業大学
Shibaura Institute of Technology
[4] 東海大学
Tokai University
[a)] tanigawa@f.ait.kyushu-u.ac.jp

テキストの変化に応じてレイヤを切り替えるプログラミング方法である．

COP のレイヤによる振る舞い変更は，コンテキストに依存する横断的関心事を扱うことができるため，サービスの切り替えのみではなく，デバイス故障や通信途絶といった非正常系の処理への適用も有用である．このような非正常系の処理は横断的関心事となり得ることが知られている [6]．

組込みソフトウェアの機械制御における非正常系処理を COP で実現する場合，事故の発生や製品の故障を防ぐために，非正常系処理の変更要求から振る舞い実行までの応答時間は重要である．非正常系の処理の変更に時間がかかる要因としては，他の正常系の変更が終了するのを待つ場合があること，振る舞いを変更するための準備に時間がかかる処理が必要となる場合があることなどが挙げられる．

我々は，非正常系処理の変更要求から振る舞い実行までの応答時間を改善するために，優先度に応じたレイヤスケジューリング手法を提案する．本手法では，上記の非正常系の処理の変更に時間がかかる要因を解決するために，緊急性の高い変更は他の変更を中断してでも先に行う仕組みや，振る舞い変更が完了するまでの間に代わりの特別な振る舞いを行う仕組みを実現する．

本稿の目的は，COP による非正常系処理の切り替えにおいて，変更要求から振る舞い実行までの応答時間によって起こりうる問題を解決することである．そのために，本稿では変更要求から振る舞い実行までの応答時間を向上させるうえで困難な事柄や，それらを解決するための方法を示す．また，提案手法によってどれだけ応答時間を向上させ，非正常系処理の変更の問題に貢献できるのかについて，適用・評価を行う．

我々は提案手法を評価するために，以前に提案した COP フレームワークである RT-COS[6][7][8] に，優先度に応じたレイヤスケジューリング手法を実装した．評価では，変更要求から振る舞い実行までの応答時間を十分に短縮できることを示すために，本フレームワークを用いて，それぞれ優先度が異なるレイヤを切り替え，振る舞い変更にかかるまでの時間を測定する．

以降，2 節では関連研究として，既存の COP 言語とそれらの要素技術を紹介する．3 節では COP による非正常系処理の切り替えにおいて，解決すべき問題や問題解決が困難な要因を示し，それらを解決する優先度に応じたレイヤスケジューリング手法を提案する．4 節では提案手法の適用例と結果を示し，それを基に提案手法の評価を行う．5 節で本稿での提案をまとめ，今後の課題を述べる．

2. 関連研究

本節では，COP における優先度に応じたレイヤスケジューリング手法を提案するために，COP の概要を示し，提案手法に関連する COP の要素技術であるメソッドディスパッチとレイヤアクティベーションについて紹介する．

2.1 COP の概要

本稿の冒頭で述べた通り，COP は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．COP は，Robert Hirschfeld らの研究 [2] が最初である．Robert Hirschfeld らは，COP が対応すべきである要素として，振る舞いの種類，レイヤ，アクティベーション，コンテキスト，スコーピングを挙げている [2]．振る舞いの種類はベースとなるクラスやメソッドの部分定義として表され，それらをグループ化したモジュールがレイヤである．さらに，COP はこれらレイヤを実行時のコンテキストに応じて動的にアクティベーションする機構を持つ．アクティベーションされたレイヤに応じてメソッドの実行先を対応する振る舞いの種類にディスパッチする．これにより，実行時のコンテキストに応じてプログラム中の複数のメソッドを一度に変更することが可能となる．コンテキストはシステムの振る舞いを変えうる要因であり，プログラムから観測可能なものである．コンテキストとしては，システムを取り巻く外部環境やシステムの内部状態，あるいはそれらの変化の順序などが挙げられる．Robert Hirschfeld らは上記の論文においてレイヤがアクティベーションされるスコープを明示的にすることの必要性を述べているが，COP 言語によっては必ずしもこれが満たされているとは限らない．レイヤアクティベーションの方法の違いについては 2.2 節で述べる．

これまでに，様々な COP 言語が提案されてきた．これらはベースとなるプログラミング言語を拡張することで実現されている [9]．ContextL[5] は最初に COP の拡張を行った言語である．ContextL は Lisp をベースとしており，Common Lisp のオブジェクトシステムを拡張している．それに続き，動的プログラミング言語のための COP ライブラリが開発された．これらのライブラリはベースとする言語で提供されるメタレベルの機能を用いて実現されている．Smalltalk をベースとする ContextS[10]，Ruby をベースとする ContextR[11]，JavaScript をベースとする ContextJS[12]，Python をベースとする ContextPy[13] や PyContext[14] などがある．これら動的プログラミング言語以外に Java を拡張したものとして ContextJ[4]，ContextJ*[2]，ContextLogicAJ[15]，JCop[3]，EventCJ[16] などがある．

2.2 COP の要素技術

2.1 節の COP 言語を特徴付ける要素として以下のものが挙げられる [9][17]．

- メソッドディスパッチ実現手段
- レイヤアクティベーションの方法

以降，提案手法を実現する上で重要な上記の要素について，既存手法のものを紹介し，それぞれの利点・欠点について記す．さらに，提案手法ではどの方法を採用したのかについて述べる．

Malte Appeltauer らはメソッドディスパッチの実現手段を二つの方法に分類している [9]．一つはメソッド実行のたびにプロキシオブジェクトを介する方法である．プロキシオブジェクトではメッセージ送信後にアクティブなレイヤを調べ，その内容に応じて実行するメソッドをディスパッチする．もう一つはアクティブなレイヤが変更されるたびに，各クラスでメッセージ送信時に実行されるメソッドを変更する方法である．前者はメソッド実行のたびに実行するメソッドを判断するため，メソッド実行時のオーバヘッドが大きくなる．後者はレイヤアクティベーションのたびに複数のクラスのメッセージ送信時に実行されるメソッドを変更するため，レイヤアクティベーション時のオーバヘッドが大きくなる．

我々は緊急時レイヤのアクティベーションの完了にかかる時間を短くするために，前者の方法を採用した．また，前者の方法では，メソッド実行時の処理を COP 機構側で一か所で管理できるため，ディスパッチの仕組みを柔軟に変更しやすい利点がある．一方で，優先度の高いレイヤのアクティベーション時には，事故やシステムの故障を防ぐために，メソッド実行の応答時間が求められる．そのため，このような場合は特別なレイヤのメソッドを後者の方法により実行することとした．

紙名はレイヤアクティベーションの方法と影響範囲によって COP 言語の分類を行っている [17]．レイヤアクティベーションの方法は以下のものがある．

with ブロック: アクティベーションしたい範囲をブロックで囲む方法である．この方法ではレイヤがアクティブである範囲を明示的に表せるが，制御フローをまたがるアクティベーションを表現するのが難しくなる．また，with ブロックがプログラム中に散在するという問題もある [3]．

決定的な活性化: アクティベーションのための命令が用意されており，それらの命令を実行後，ずっとそのレイヤがアクティブとなる方法である．制御フローをまたがるアクティベーションが容易であるが，意図しないレイヤアクティベーションの衝突が起こる恐れがある [17]．

イベント駆動: 青谷らが提案した EventCJ[16] で行っている方法である．この方法ではイベント宣言とレイヤ遷移規則を用いてレイヤアクティベーションを宣言的に指定する，イベント宣言で宣言したイベントが発生した際に，レイヤ遷移規則に応じてレイヤやコンテキストを切り替える．この方法の特徴としては，レイヤアクティベーションの指定が宣言的に行われることでアクティベーションのためのコードが散在しないことや，レイヤ遷移規則を用いることでモデル検査を行いやすく，仕様との一致を検査できることが挙げられる．

また，レイヤアクティベーションの影響範囲としては，スレッドごと，あるいはインスタンスごとのアクティベーションやアプリケーション全体の振る舞いが変更される大域的なアクティベーションがある．

提案手法では，イベント駆動で影響範囲が大域的なアクティベーションを採用している．イベント駆動方式はレイヤアクティベーションやコンテキスト推定が散在せず，COP を実現するための機構から管理することが容易であるため，提案手法を実現するのに向いている．with ブロックによる方法はブロックの位置によって，優先すべきレイヤを決定しやすいが，with ブロックがプログラム中に散在するため，非正常系の処理が横断的関心事となって表れることを解決できない問題がある．

提案手法はメソッドディスパッチやレイヤアクティベーションについては，既存の手法を採用している．これらに加え，優先度に応じたレイヤスケジューリング手法を実現し，緊急性の高い振る舞い変更の応答時間を短縮することが，本研究の目的である．

3. 優先度に応じたレイヤスケジューリング手法の提案

本節では，COP による非正常系処理の切り替えにおいて，解決すべき問題や問題解決が困難な要因を示し，それらを解決する優先度に応じたレイヤスケジューリング手法を提案する．また，本手法は以前に提案した COP フレームワークである RT-COS[6][7][8] に機能追加を行うことで実現したため，RT-COS の概要も示す．

3.1 解決すべき問題

本稿冒頭で述べた通り，COP はコンテキストに依存する横断的関心事を扱うための技術であり，横断的関心事となり得る非正常系の処理の切り替えに有用である．図 1 に，COP のレイヤを用いた非正常系処理への切り替えの例として，災害現場に物資を運ぶレスキューロボットの構造を示す．災害現場では，電波状況や足場の不安定さから，通信途絶やデバイス故障が起こり得る．図に示す構造では，これらの場合に対処するための処理をレイヤに切り分けて，複数のモジュールでの処理を一度に変更できるようにしている．

非正常系の処理の変更では，事故の発生や製品の故障を防ぐために，変更要求から振る舞い実行までの応答時間が重要である．例えば，図 1 では通信途絶やデバイス故障の際に，動作制御モジュールの振る舞いをゆっくり進むように変更しており，異常が発生した場合でも動作が安定しやすいようにしている．このような変更が遅れると，事故の

発生や製品の故障につながる可能性がある．非正常系の処理の変更に時間がかかる要因として，他のレイヤのアクティベーションを待つ必要がある場合や，アクティベーション時に使用するデバイスの切り替えのような時間がかかる処理が必要な場合などが挙げられる．

本研究の目的は，優先度に応じたレイヤアクティベーション手法を提案し，非正常系処理の変更のための応答時間の短縮することで，上記の問題を解決することである．以下に，COP において，レイヤアクティベーションから振る舞い実行までの応答時間が増大する原因を記す．提案手法によって，これらの問題を解決することで，非正常系処理の変更のための応答時間を短縮する．

(1) 他のレイヤアクティベーションの待ち時間：
レイヤのアクティベーション要求が複数ある場合，緊急性の高いレイヤアクティベーションが待たされる可能性がある．これにより，緊急性の高い変更が反映されるまでに時間がかかることとなる．

(2) アクティベーション完了までの待ち時間：
アクティベーション時に，使用するデバイスの切り替えのような時間がかかる処理が必要な場合に，完了までに時間がかかる．

(3) メソッドディスパッチの待ち時間：
2.2 節で述べた通り，プロキシオブジェクトによるメソッドディスパッチは時間がかかる．

提案手法では，(1) の問題をレイヤごとに優先度を付け，緊急性の高いレイヤアクティベーションから先に行うようにすることで解決する．また，優先度の高いレイヤのアクティベーション要求が発生した際に，すでに優先度が低いレイヤをアクティベート中の場合がある．このような場合のために，優先度が低いレイヤのアクティベーションを中断し，優先度の高いレイヤのアクティベーションから行う仕組みが必要となる．

(2) による応答時間増加の問題は，優先度が高いレイヤのアクティベーション完了までの間に，代わりに実行される特別なレイヤを用意することで解決する．これにより，異常が起きたまま動き続けることで，事故や製品の故障が発生することを防ぐ．

また，このような緊急時の処理は素早く行われる必要があるため，(3) の問題を解決する必要がある．そのため，アクティベーション完了までの間に，代わりに実行されるレイヤでは，メソッドディスパッチの方法を変更する．これらの仕組みを 3.2 節の RT-COS に追加することで，提案手法を実現する．上記の仕組みの詳細については 3.3 節で述べる．

3.2 COP フレームワーク RT-COS

我々は，以前に提案した COP フレームワークである RT-COS[6][7][8] に機能追加を行うことで，提案手法を実現する．本節では RT-COS の目的や構造・仕組みを示し，次節で提案手法を実現するために変更すべき箇所について述べる．

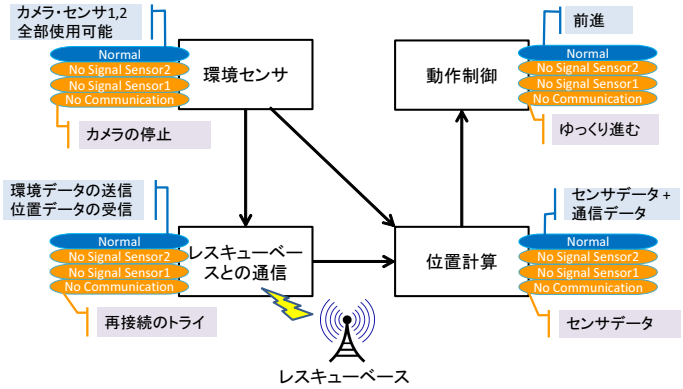

図 1 COP のレイヤを用いた非正常系処理への切り替えの例
Fig. 1 Example of the changing from normal service to abnormal service with COP

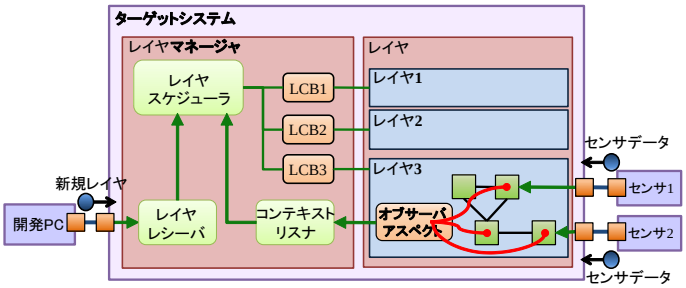

図 2 RT-COS の構造
Fig. 2 A structure of RT-COS

```
class Layerα : Layer {
  class A : ILayerActivationListener {
    M1() {...}
    M2() {...}
    void OnLayerActivation(Layer layer) {
      初期化処理1();
      中断ポイント();
      初期化処理2();
    }
  }
}
```

図 3 レイヤアクティベーション時の初期化プログラム例
Fig. 3 Initialize of layer activation

RT-COS は COP フレームワークであり，プロトタイプとして C#をベースとしたものを開発してきた．本フレームワークは以下の仕組みを備えている．(a) レイヤアクティベーションの優先順位やレイヤ同士の排他・依存などの関係を明示的に扱う仕組み．(b) レイヤアクティベーション

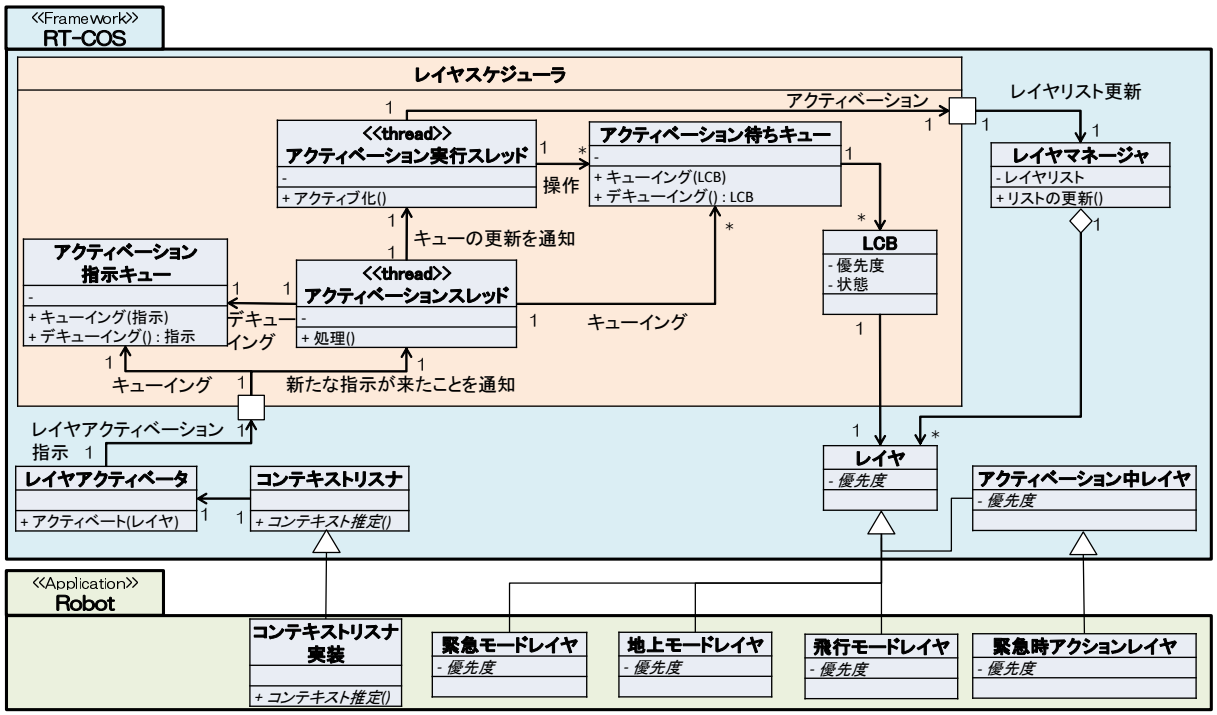

図 4 レイヤスケジューラの構造
Fig. 4 A structure of Layer Scheduler

と振る舞い実行の間で互いにそれぞれの完了を待ち同期を取る仕組み．(c) コンテキスト推定をサービス実現のためのプログラムから分離する仕組み．提案手法は，RT-COS の (a) の仕組みを変更することによって実現する．

本フレームワークの構造を図 2 に示す，RT-COS では，図 2 のオブザーバアスペクトから，各レイヤにおけるイベントを発行する．コンテキストリスナで発行されたイベントに基づきコンテキスト推定を行い，アクティベーションするレイヤをレイヤスケジューラに通知する．レイヤスケジューラでは，(a) の優先度に基づくレイヤアクティベーションを行う．複数のレイヤが同時にアクティベーションされた場合に，優先度の高いレイヤからアクティベーションが行われる仕組みとなっている．レイヤの優先度や状態は LCB(Layer Control Block) が持っており，レイヤスケジューラはこれらの情報に基づき，レイヤアクティベーションの順番を決定する．

3.3 提案

本節では，優先度に応じたレイヤスケジューリング手法を提案する．本手法では，3.1 節で示した問題を，3.2 節で述べた通り，RT-COS のレイヤスケジューラやメソッドディスパッチの仕組みを変えることで実現する．以降，各節で 3.1 節で求められることを，どのような方法で実現したのかについて記す．

3.3.1 横取り可能な優先度に応じたレイヤスケジューリング

本節では，優先度に応じたレイヤスケジューリングについて述べる．3.1 節の問題 (1) で述べた通り，複数のレイヤアクティベーションがある場合，他のレイヤアクティベーションの待ち時間によって，緊急性の高いレイヤのアクティベーションに時間がかかる．この問題を解決するために，レイヤごとにアクティベーションの優先度を設定し，それに応じてアクティベーションの順番を決定する必要がある．また，レイヤアクティベーションの実行中に，より優先度の高いレイヤのアクティベーション要求が発生した場合は，実行中のレイヤアクティベーションを中断し，優先度の高いレイヤをアクティベーションする必要がある．3.1 節の (1) を解決する上で，我々が以前に提案した RT-COS では，優先度の低いレイヤのアクティベーションを中断する仕組みがないため，レイヤスケジューラの機能を拡張する必要がある．

レイヤアクティベーションを中断する際の問題として，どの時点で中断するかということが問題になる．レイヤアクティベーション時にレイヤスケジューラが行うことは次の二点である．(i) ユーザ記述のレイヤアクティベーション時の初期化処理，(ii) メソッドディスパッチで実行されるメソッドの更新．このうち，(ii) の処理を途中で中断すると，一部のメソッドの実行のみが変更される恐れがある．(i) はセンサからの応答待ちなど，(ii) に比べ中断しても問題ない処理が存在すると考えられる．そのため，我々は，

(i) でアクティベーションを中断しても良いタイミングをユーザが指定する方法を採用した．図 3 にレイヤアクティベーション時の初期化プログラムの例を示す．レイヤアクティベーション時の初期化は，レイヤに含まれる各クラスで ILayerActivationListener インターフェースを実装することにより実現する．中断可能な箇所の指定は，初期化時に実行されるメソッド中で，中断用のメソッドを用いることにより行う．これにより，ユーザが指定した時点でレイヤアクティベーションが中断されるようになる．

我々は，レイヤアクティベーションを途中で中断することを可能にするために，レイヤスケジューラの処理を 2 つのスレッドに分けた．上記レイヤスケジューラの構造を図 4 に示す．図 2 でも示す通り，レイヤアクティベーションの指示は，コンテキストリスナからレイヤスケジューラに対して出す．レイヤスケジューラはレイヤアクティベーションの指示を基に，2 つのスレッドを用いてレイヤアクティベーションを行う．一つはコンテキストリスナから出された指示を基に，どのレイヤをアクティベートするかを決定するアクティベーションスレッド，もう一つは上記で示したアクティベーションのための処理 (i)(ii) を実行するスレッドである．これら二つのスレッドに分けることにより，レイヤスケジューラは，レイヤアクティベーション実行途中に，より優先度の高いレイヤのアクティベーション要求があることを知ることが可能となる．

レイヤスケジューラは，レイヤの状態や優先度を図 2 の LCB で管理しており，LCB の状態・優先度ごとにキューが用意されている．レイヤスケジューラでは，これらを用いて順番にアクティベーションを行う．レイヤの状態遷移を図 5 に示す．非アクティブなレイヤはアクティベーション指示があれば，優先度に応じたアクティベーション待ちキューにキューイングされ，順番が来ればアクティベーション処理が行われる．アクティベーション中に，より高優先度レイヤのアクティベーション指示があれば，処理を中断し，アクティベーション待ちキューに再度キューイングされる．この中断のための状態遷移は提案手法で追加したものである．

上記を実現するための，レイヤスケジューラアルゴリズムを図 6 に示す．コンテキストリスナからアクティベーション指示が来ると，アクティベーションスレッドが立ち上がる．このスレッドでは，指示内容を基にアクティベーション待ちキューを更新し，アクティベーション実行スレッドにキューの更新を伝える．アクティベーション実行スレッドは，ユーザが指定した中断ポイントで，アクティベーション待ちキューの内容から，処理を中断すべきか判断する．

3.3.2 高優先度レイヤアクティベーション時のメソッドディスパッチ

3.1 節の問題 (2)(3) に関しては，メソッドディスパッチ

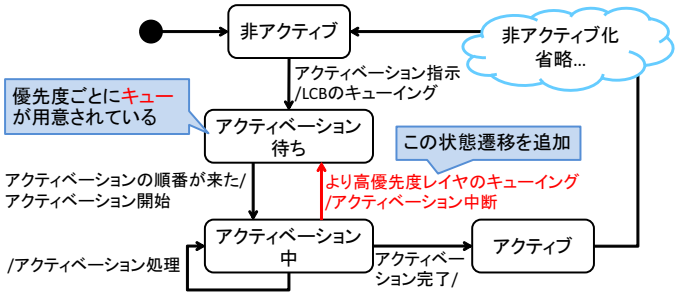

図 5 レイヤの状態遷移
Fig. 5 State transition of a layer

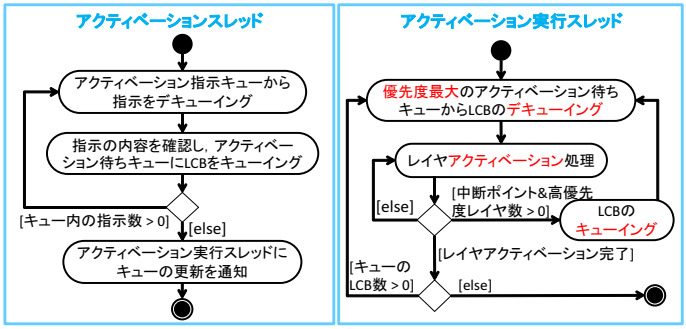

図 6 レイヤスケジューラアルゴリズム
Fig. 6 Layer scheduling algorithm

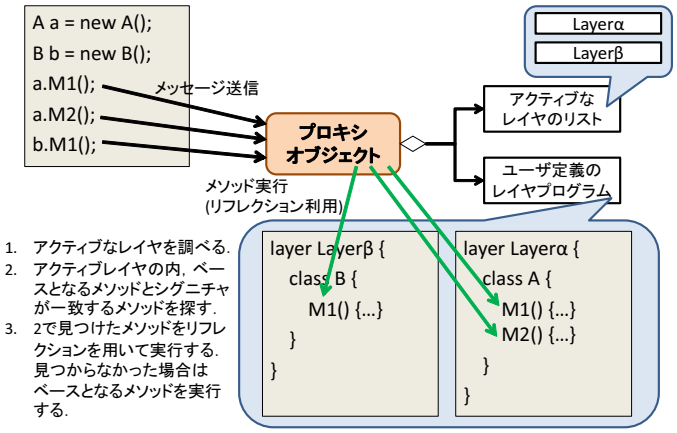

図 7 メソッドディスパッチの仕組み
Fig. 7 A mechanism of method dispatch

の方法を変更することで解決する．本節では，高優先度レイヤアクティベーション時のメソッドディスパッチの方法について述べる．図 7 に RT-COS におけるメソッドディスパッチの仕組みを示す．RT-COS では，メソッドディスパッチをメソッド実行時にプロキシオブジェクトを介する方法で実現している．2.2 節で述べた通り，この方法は各クラスでメッセージ送信時に実行されるメソッドを変更する方法よりも，レイヤアクティベーションにかかる時間が短いが，メソッド実行に時間がかかる．提案手法では問題 (2)(3) のために，優先度の高いレイヤのアクティベーション時には，アクティブなレイヤを探す手順を省略する仕組みを実現する．

表 1 適用結果：メソッド実行の時間
Table 1 Application result: execution time of the method

計測時の状況	計測結果[μs]	分散	実行されるメソッドのレイヤ
(a)アクティベーション前	0.716868508	0.077460	ベースレイヤ
(b)Aアクティベーション中	0.711668699	0.079967	ベースレイヤ
(c)Bアクティベーション中	0.423076576	0.020967	アクティベーション中レイヤ
(d)Bアクティベーション完了	0.653060689	0.107476	Bレイヤ
(e)Aアクティベーション完了	0.760683137	0.059651	Bレイヤ

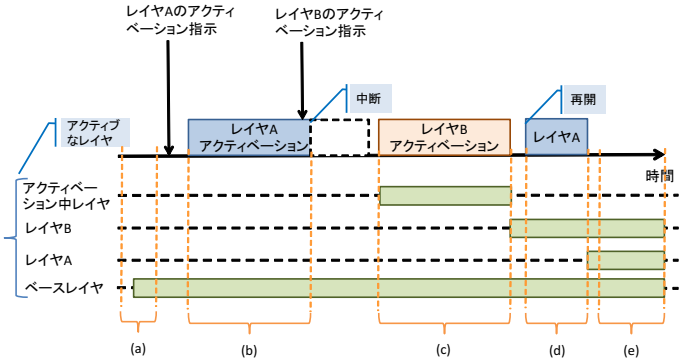

図 8 評価プログラムの流れ
Fig. 8 The flow of the evaluation program

提案手法では，優先度が高いレイヤのアクティベーション完了までの間に，代わりに実行される特別なレイヤを用意する．このレイヤは図 4 のアクティベーション中レイヤに当たる．アクティベーション中レイヤは，高優先度のレイヤアクティベーション中にアクティブになり，メソッドディスパッチでは図 7 の手順 2 が省略される．また，これを実現するために，アクティベーション中レイヤには他のレイヤと異なり，以下のルールを設けることとする．

(i) レイヤアクティベーション初期化処理を行わない：
処理をすぐに切り替えられるように，初期化処理で時間を取られないようにする．

(ii) プログラム中に 1 つだけしか存在できない：
高優先度のレイヤアクティベーション中に実行すべきレイヤを特定する時間を削減するために，アクティベーション中レイヤはプログラム中で一つだけとする．

これらの制約は，上記の特別なレイヤが求められる役割から，重要な問題とはなりえない．このレイヤは，高優先度のレイヤアクティベーション時の応急処置となる振る舞いを表すことを目的としており，具体的な振る舞いとしては，ロボットの停止など単純なものが考えられる．これらの制約によって実現が困難となる振る舞いについては，上記の仕組みの対象とせず，通常のレイヤを用いて実装することとする．

このような仕組みを実現することで，3.1 節の問題 (2)(3) を解決し，高優先度レイヤアクティベーション時の応答時間を改善する．

4. 適用・評価

本節では，提案手法適用時のメソッド実行時間を計測することで，変更要求から振る舞い実行までの応答時間が削減できることを示す．計測の流れを図 8 に示す．図の上部はアクティベーションの流れ，下部はアクティブになっているレイヤを表している．計測では，低優先度レイヤ A と高優先度レイヤ B をアクティベートする．アクティベートの指示は A,B の順番で行い，レイヤ A のアクティベート途中にレイヤ B のアクティベート指示を出す．アクティブなレイヤが無い場合は，プログラム実行時点でアクティブなベースレイヤという特別なレイヤのメソッドが実行される．各レイヤのメソッドの処理では，変数のインクリメントを一回だけ行うこととする．応答時間の評価では，図 8 の (a)-(e) の区間でそれぞれメソッド実行時間を 100 回計測し，その平均を求めることとする．

ベンチマーク環境は，CPU が Intel Core i3 (1.5GHz, 4 コア)，メモリは 4GB を搭載し，64 ビットの Windows7 で.net フレームワーク 4.0 で実行した．表 1 に計測結果を示す．

3.1 節 (3) で述べた通り，メソッドディスパッチの待ち時間は，レイヤアクティベーションから振る舞い実行までの応答時間が増大する原因となる．表 1(c) の結果から，高優先度レイヤのアクティベーション中は，3.3.2 節で述べたメソッドディスパッチによって，メソッドの実行時間が短縮されることが明らかである．また，レイヤを探す手順が省略された分，実行時間のばらつきも他より少なく，安定している．これにより，3.1 節 (3) の問題を解決している．(1)(2) についても，計測のために評価プログラムを実行した結果，図 8 の通りに振る舞うことが確認でき，それぞれ解決できることが明らかとなった．メソッドディスパッチの時間短縮が，全体の性能にどのような効果を与えるかについての評価は今後の課題である．

5. おわりに

本稿では，コンテキスト指向プログラミングにおける，優先度に応じたレイヤスケジューリング手法を提案した．本手法は，組込みソフトウェアの機械制御を対象としており，非正常系処理変更の応答時間によって，事故の発生や

製品の故障が起こるのを防ぐことを目的としている．

本手法は非正常系処理の変更要求から振る舞い実行までの応答時間を短縮するために，以下の仕組みを持つ．(1) 横取り可能な優先度に応じたレイヤスケジューリング，(2) 高優先度レイヤアクティベーション時のメソッドディスパッチ．(1) によって，高優先度レイヤのアクティベーションにおいて，低優先度のレイヤアクティベーションの完了を待つ時間を短縮することが可能となる．(2) によって，例外発生という緊急時の応答時間を削減することを可能とする．

我々は上記の手法について評価するために，以前に提案した COP フレームワークである RT-COS[6][7][8] の機能を拡張することで提案手法を実現した．また，2 つの優先度が異なるレイヤをアクティベーションし，その時の振る舞いとメソッド実行時間を計測することにより，変更要求から振る舞い実行までの応答時間が削減できることを示した．

今後は，レイヤアクティベーションを中断することによって起こり得る問題を明らかにし，レイヤスケジューリングアルゴリズムをさらに実用的なものとしたい．

参考文献

[1] Shadbolt, N. : Ambient Intelligence, IEEE Intelligent Systems, Volume 18 Issue 4, pp. 2-3 , (2003).
[2] Hirschfeld, R., Costanza, P. and Nierstrasz, O. : Context-oriented Programming, Journal of Object Technology, Vol. 7, No. 3, pp. 125–151, (2008).
[3] Appeltauer, M., Hirschfeld, R. and Lincke, J. : Declarative Layer Composition with the JCop Programming Language, Journal of Object Technology, Vol. 12, No. 4, pp. 4:1–37, (2013).
[4] Appeltauer, M., Hirschfeld, R., Haupt, M. and Masuhara, H. : ContextJ: Context-oriented Programming with Java, In Proceedings of the JSSST Annual Conference 2009, pp. 1–15, (2009).
[5] Costanza, P. and Hirschfeld R. : Language Constructs for Context-oriented Programming: An Overview of ContextL. In DLS '05: Proceedings of the 2005 symposium on Dynamic languages, pp. 1–10, (2005).
[6] Watanabe, H., Sugaya, M., Tanigawa, I., Ogura, N. and Hisazumi, K. : A Study of Context-Oriented Programming for Applying to Robot Development, Proceedings of the Workshop on Context-oriented Programming (COP) 2015, ECOOP 2015, (2015).
[7] Tanigawa, I., Ogura, N., Sugaya, M., Watanabe, H. and Hisazumi, K. : A Structure of A C# Framework ContextCS based on Context-Oriented Programming, MODULARITY Companion'15, pp. 21–22 (2015).
[8] 谷川郁太, 小倉信彦, 菅谷みどり, 渡辺晴美 : 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, 組込みシステムシンポジウム 2014 論文集, 情報処理学会, pp. 84–89, (2014).
[9] Appeltauer, M., Hirschfeld,R., Haupt, M., Lincke, J. and Perscheid, M. : A Comparison of Context-oriented Programming Languages, In Proceedings of the Workshop on Context-oriented Programming (COP) 2009, ECOOP 2009, pp. 1–6, (2009).
[10] Hirschfeld, R. Costanza, P. and Haupt M. : An Introduction to Context-Oriented Programming with ContextS, In Generative and Transformational Techniques in Software Engineering (GTTSE) II, Springer LNCS 5235, pp. 396–407, (2008).
[11] Schmidt, G. : ContextR & ContextWiki. Master's thesis, Hasso-Plattner-Institut, Potsdam, (2008).
[12] Lincke, J., Appeltauer, M., Steinert, B. and Hirschfeld R. : An Open Implementation for Context-oriented Layer Composition in ContextJS. In Elsevier Journal on Science of Computer Programming, Special Issue on Software Evolution, (2011).
[13] Schubert, C. : ContextPy & PyDCL - Dynamic Contract Layers for Python, Master's thesis, Hasso-Plattner-Institut, Potsdam, (2008).
[14] von Lowis, M., Denker, M. and Nierstrasz, O. : Context-oriented Programming: Beyond Layers, In ICDL '07: Proceedings of the 2007 international conference on Dynamic languages, volume 286 of ACM International Conference Proceeding Series, pp. 143–156, (2007).
[15] Appeltauer, M., Hirschfeld, R. and Rho T. : Dedicated Programming Support for Context-aware Ubiquitous Applications, In Proceedings of the 2nd International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM) 2008, pp. 38–43, (2008).
[16] 青谷知幸, 紙名哲生, 増原英彦: オブジェクト毎の層遷移を宣言的に記述できる文脈指向言語 EventCJ, コンピュータソフトウェア, Vol. 30, No.3, pp. 130–147, (2013).
[17] 紙名哲生: 文脈指向プログラミングの要素技術と展望, コンピュータソフトウェア, Vol. 31, No. 1, pp. 3–13, (2014).