ContextROS: ロボットペレーティングシステムへのコンテキスト指向プログラミングの適用

佐伯 優太[1,a)]　谷川 郁太[1,b)]　久住 憲嗣[1,c)]　福田 晃[1,d)]

概要：コンテキストアウェアなロボットの開発はロボット分野における重要な課題の 1 つである．このロボットは周囲の状況と内部の状態をコンテキストとして認識し，コンテキストに応じて振る舞いを変更する．このようなロボットは様々な状況に柔軟に対応できるため非常に有用である．オープンソースのミドルウェアであるロボットオペレーティングシステム（ROS）はロボット工学アプリケーションに広く使用されている．ROS はコンポーネントベースのピアツーピア分散システムであるため，優れた汎用性，再利用性，移植性を備えたアプリケーションの開発を容易にする．このことから，ROS を用いたコンテキストアウェアなロボットの開発が望まれる．しかし，ROS ではコンテキストに依存する振る舞いの記述をサポートするフレームワークが存在しないため，コンテキストに依存する振る舞いがプログラム全体に分散し，保守性を低下させることが課題となっている．本稿では，ROS にコンテキスト指向プログラミング（COP）を適用した ContextROS を提案する．ContextROS は ROS にレイヤの概念を導入し，コンテキストを明示し振る舞いをモジュール化する．また，ROS の通信機能を使用して，分散環境でのレイヤーのアクティベーション/非アクティブ化を実行する．ContextROS を簡単なアプリケーションに適用し ROS と比較して評価を行う．評価の結果，ContextROS は ROS の性能を損なうことなくコンテキストごとの振る舞いの記述の分散を小さくできることが確認できた．

キーワード：コンテキスト指向プログラミング, ロボットオペレーティングシステム, 組込みシステム

ContextROS: Context-Oriented Programming for the Robot Operating System

Saeki Yuta[1,a)]　Tanigawa Ikuta[1,b)]　Hisazumi Kenji[1,c)]　Fukuda Akira[1,d)]

Abstract: In recent years, the development of context-aware robots has been one of the crucial tasks in the robotics field. This robot changes its behavior according to the context, which includes its surrounding situation and the internal state. Robots such as these are highly useful because they can respond flexibly to various situations.
The Robot Operating System (ROS), which is open-source middleware, has been widely used for robotics applications. ROS facilitates the development of applications with excellent versatility, reusability, and portability because it is a component-based peer-to-peer distributed system. These advantages have encouraged many developers to employ ROS to develop context-aware robots. However, it is complicated to write context related codes as developers have to write the code without any framework supports. Thus, it is not easy to describe a context-aware program, because its maintainability tends to be low as the fragments of code pertaining to context dependence are distributed throughout the code.
This paper proposes ContextROS which applies the Context-oriented Programming (COP) to ROS. ContextROS adds a language element in the form of a layer to ROS to modularize the context dependence code explicitly. It also performs layer activations/deactivations for distributed nodes using ROS communication features. We apply ContextROS to a simple application and evaluate it by comparison with ROS. The results of the evaluation confirmed that ContextROS is confirmed to be able to reduce the dispersion of the description of the behavior of each context without compromising the performance of the ROS.

Keywords: Context-oriented Programming, Robot Operating System, Embedded system

1. はじめに

自動運転車両やマルチサービスロボットに用いられる重要な機能の一つとしてコンテキスト認識がある．これらのロボット等はコンテキストを認識し，コンテキストに応じて振る舞いを変更することでより柔軟なサービスを提供することが可能となる．例えば，災害救助ロボット [1] が人が立ち入ることができない場所へ向かう場合を考える．このロボットは災害状況とバッテリ残量に応じて移動方法を車輪走行またはプロペラ飛行に変更する．

オープンソースのミドルウェアであるロボットオペレーティングシステム（ROS） [2] はロボット工学アプリケーションに広く使用されている．ROS はメッセージベースのピアツーピア分散システムであり，優れた汎用性，再利用性，移植性を備えたアプリケーションの開発を容易にする．このことから，ROS によるコンテキストアウェアなロボットの開発が望まれる．しかし，ROS にはコンテキストに依存する振る舞いの記述をサポートするフレームワークが存在しないため，コンテキストの明示，コンテキスト依存な振る舞いのモジュール化，分散環境下でのレイヤアクティベーションが課題となる．

この課題を解決するために，コンテキスト指向プログラミング（COP） [3] の概念を ROS に適用した ContextROS を提案する．COP を用いることでコンテキスト依存な振る舞いの記述の複雑さが軽減される．ContextROS は COP のレイヤの概念を導入することで，コンテキストの明示とコンテキスト依存な振る舞いのモジュール化を行う．そして，ROS のトピック通信を利用しレイヤをアクティブ化/非アクティブ化することでコンテキスト依存な振る舞いの変更を行う．

本論文の構成は以下のとおりである．第 2 章ではコンテキストアウェアなロボットの要素技術を紹介する．第 3 章ではコンテキストアウェアなロボットの例を用いて課題を明確にする．第 4 章では提案手法の説明を行う．第 5，6 章では提案手法を簡単なアプリケーションに適用し，その性能を評価する．第 7 章では今後の課題について議論する．第 8 章では関連研究の紹介を行う．最後に第 9 章でまとめとする．

2. ロボットオペレーティングシステム

2.1 概要

オープンソースのロボットミドルウェアであるロボットオペレーティングシステム（ROS）はロボットアプリケーションの実装に広く使用されている．ROS はコンポーネントベースの分散システムの開発を容易にするという特徴を持つ．このため，汎用性，再利用性，移植性に優れたアプリケーションの開発が可能となる．ROS はノードと呼ばれるコンポーネントによってシステムを構築する．これらのノードを組み合わせ，互いに通信させることで特定の機能を実現する．これらのノードの多くはオープンソースのコミュニティによって提供されているが，自分で開発することも可能である．このようにノードを再利用し組み合わせることで迅速にシステムを開発することができる．

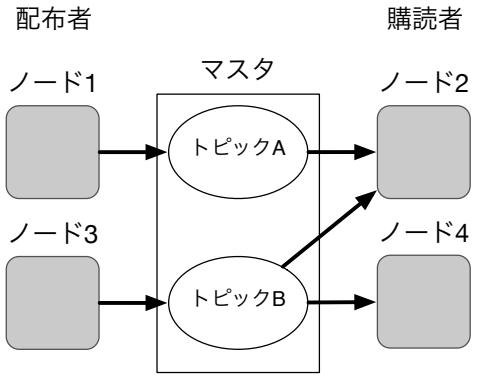

図 1: トピック通信

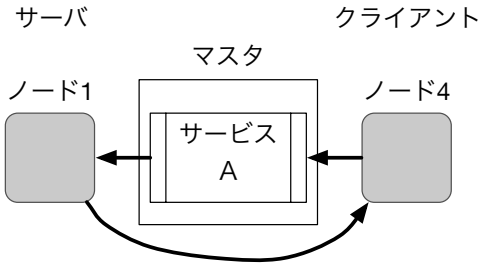

図 2: サービス通信

ROS ではノード間の通信を管理する ROS マスタと呼ばれるノードが提供されている．ROS マスタノードはシステム内のノードの命名や登録されたサービスを提供する．ノード間の通信はメッセージと呼ばれるデータの単位を使用して行われる．メッセージは特定のデータ構造を持ち，様々なプログラミング言語に関連するデータの属性を厳密に定義する．ROS にはノード間の通信方法としてトピック通信とサービス通信の 2 種類の通信方法が存在する．

2.2 通信方法

トピック通信を図 1 に示す．メッセージの送信者と受信者をそれぞれ配布者，購読者と呼ぶ．購読者は任意のトピックを事前に購読登録し，配布されたトピックを購読することでメッセージを受信する．配布者はメッセージをトピックという形で配布する．このようにトピック通信は宛先を指定せずトピックを介して通信することで非同期通信

[1] 九州大学
Kyushu University
[a)] y_saeki@f.ait.kyushu-u.ac.jp
[b)] tanigawa@f.ait.kyushu-u.ac.jp
[c)] nel@slrc.kyushu-u.ac.jp
[d)] fukuda@f.ait.kyushu-u.ac.jp

表 1: コンテキストと対応する振る舞いの関係

コンテキスト	位置推定方法	掃除方法
屋内	Gyro	掃き掃除
屋外	GPS	吸引掃除

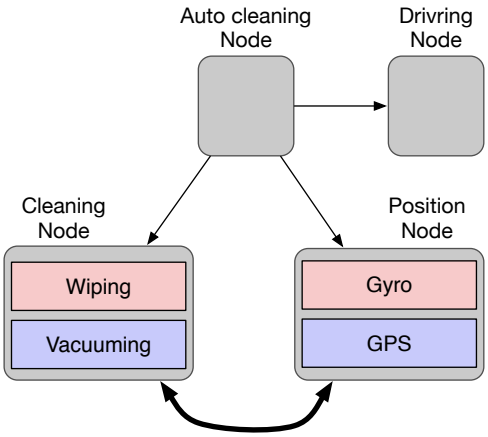

図 3: ROS による自動掃除ロボットの設計

を実現する．

サービス通信を図 2 に示す．サービス通信はリモートプロシージャコール（PRC）の一種であり，サービスを提供する側をサーバ，要求する側をクライアントと呼ぶ．トピック通信とは対象的にサーバとクライアントの間で同期通信が行われる．

2.3 ROS によるロボットの課題

ROS によって開発されたコンテキストアウェアなロボットの課題を自動掃除ロボットの例を用いて明らかにする．この自動掃除ロボットは位置推定を行いながら移動し掃除を行う．このロボットは屋内と屋外 2 つのコンテキストを認識し，コンテキストに応じて振る舞いを変更する．コンテキストと対応する振る舞いの関係を表 1 に示す．屋内ではジャイロで位置推定を行い，拭き掃除を行う．一方，屋外では GPS によって位置推定を行い，吸引掃除を行う．ROS によるこのロボットの設計を図 3 に示す．ロボットの各振る舞いはノード内に表現される．ここで屋外での振る舞いに注目すると，GPS と吸引掃除が別々のノードに分散していることがわかる．このように ROS による設計では各ノードにコンテキスト依存な振る舞いが分散し，プログラムの保守性を低下させる．

3. ContextROS フレームワーク

そこで，この問題を解決するため我々は ROS に COP を適用した ContextROS フレームワークを提案する．多くの COP 言語はコンテキストをレイヤとして明示的に扱い，そのレイヤをアクティベーションまたはディアクティベーションする．そして，アクティベーションされているレイヤに応じて振る舞いを変更するメカニズムを提供する．ここで，ROS に COP を適用するにあたってノードをまたがるレイヤの導入が課題となる．ROS は複数のノードで構成された分散システムであるため，システム全体の振る舞いを変更するためには図 4 に示すようなノードをまたがるレイヤが必要となる．

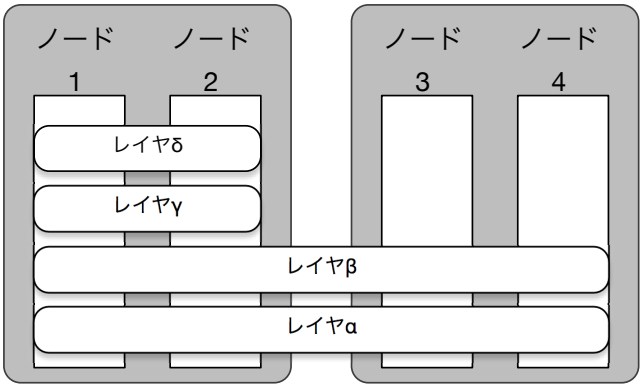

図 4: ノードをまたがるレイヤ

そこで，我々は ContextROS を図 5 に示すように設計した．ContextROS はノードをまたがりレイヤを管理するレイヤマネージャと単一ノード上で振る舞いの変更を行う下層 COP 言語の 2 つの要素から構成される．レイヤマネージャはアクティブレイヤを管理し，レイヤのアクティベーションまたはディアクティベーションが行われた際に他のノードへレイヤの情報を送受信する．下層 COP 言語はコンテキスト依存な振る舞いが実行された際に実行時のアクティブなレイヤに対応して振る舞いを変更する機能を提供する．ノード内で使用される下層 COP 言語は外部から非同期的にレイヤを切り替えることができるものとする．[4]

このように，ノードをまたがるレイヤをレイヤマネージャによってレイヤ情報を共有することで実現し，下層 COP 言語を用いて各ノードでの振る舞いを変更することで ROS において各ノードにコンテキスト依存な振る舞いを分散させることなくコンテキストに応じた振る舞いの変更を可能とする．

ROS はノード間でのメッセージの送信方法として同期通信と非同期通信の 2 種類の通信方法を提供しているため，COP マネージャと下層 COP 言語との間でのメッセージの送信方法を同期通信と非同期通信の 2 種類での実現方法を検討している．以下にそれぞれの特徴をまとめる．

(1) 非同期通信では通信の性質上レイヤのアクティブ化/非アクティブ化の順序が必ずしも保証されるとは限らない．しかし，ノードの追加，削除，切断などの例外条件への対応は比較的用意であり，ノードの増減に伴うレイヤアクティベーション時間の増加は許容できる範囲である．

(2) 同期通信はレイヤのアクティブ化/非アクティブ化の順序を保証できるが，ノードの数が増えると通信時間

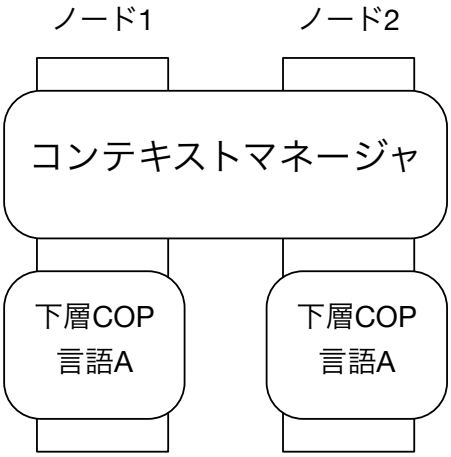

図 5: ContextROS

が長くなる．

4. プロトタイプの実装

4.1 設計

ContextROS のプロトタイプ実装について説明する．図 6 に ContextROS のプロトタイプ実装の設計を示す．ContextROS は ROS でサポートされている言語の 1 つである C++を用いて実装した．レイヤはプリコンパイラを使用して実現する．プリコンパイラはレイヤとそのレイヤに対応した振る舞いが記述されるレイヤ記述を解釈し，そのレイヤに応じて振る舞いを変更する関数を生成する．この振舞い変更関数をライブラリとして使用することにより，下層 COP 言語を実現する．そして，アクティブレイヤの情報を変数として保持し，ROS のトピック通信を利用したアクティベーションおよびディアクティベーション用の命令を用意することでレイヤマネージャを実現する．

4.2 レイヤ記述

レイヤ記述はコンテキストの明示を行い，コンテキスト依存な振る舞いのモジュール化を行う．レイヤ記述の BNF をソースコード 1 に示す．レイヤ記述はヘッダ宣言部とレイヤ宣言部で構成されている．ヘッダ宣言部はレイヤ宣言部より上に記述し，レイヤ宣言部で用いるライブラリの宣言を行う．レイヤ宣言部ではコンテキストに対応するレイヤが宣言される．各レイヤでは同じ名前で異なる振る舞いをもつ関数を定義する．レイヤ記述の例をソースコード 2 に示す．

4.3 レイヤ記述の解釈

ContextROS はレイヤ記述を静的に解釈し振る舞いを変更する関数とレイヤ情報を確認する関数を生成する．レイヤ記述の BNF に基づいて字句解析，構文解析を行い抽象構文木を生成する．生成された抽象構文木をもとに振る舞いを変更する関数を生成する．この関数はアクティブなレイヤの情報に対応するメソッドを実行する．

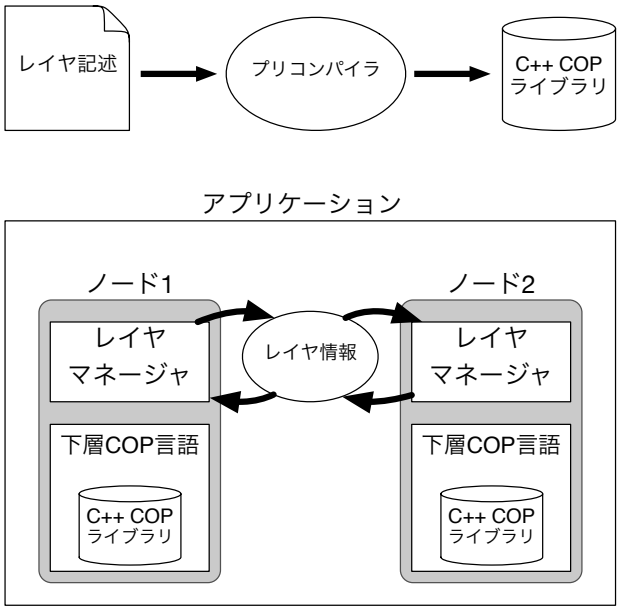

図 6: C++によるプロトタイプ ContextROS の設計

ソースコード 1: レイヤ記述の BNF

```
1  <レイヤ記述> = <ヘッダ宣言部>, <レイヤ宣言部>
2  <レイヤ宣言部> = <レイヤ宣言>
3    | <レイヤ宣言>, <レイヤ宣言部>
4  <レイヤ宣言> = "Layer", <レイヤ名>,
5    "[", <コンテキストメソッド群>, "]"
6  <レイヤ名> = <文字列>
7  <コンテキストメソッド群> = <メソッド>
8    | <メソッド>, <コンテキストメソッド群>
9  <メソッド> = <型>, <メソッド名>, <引数>,
10   "{", <振る舞い>, "}\n\n"
```

ソースコード 2: レイヤ記述の例

```
11 /*ヘッダ宣言部*/
12 #include <location>
13 /*レイヤ宣言部*/
14 Layer Indoor [
15     location getPos(){/*屋内での振る舞い*/}
16 ]
17 Layer Outdoor [
18     location getPos(){/*屋外での振る舞い*/}
19 ]
```

4.4 レイヤアクティベーション

ノード間をまたがるレイヤアクティベーションの実現のために ContextROS は専用のアクティベーション命令を用意する．この命令は ROS のサービス通信を使用して実

行される．アクティベーション命令はアクティベートするレイヤの名前を入力としアクティベーションの成否を出力する．ContextROS に登録されていないレイヤをアクティベーションした場合アクティベーションは失敗となる．このレイヤの確認にはレイヤ記述から生成される関数を用いる．

5. 評価

ROS に COP を適用する際のオーバーヘッドを調べるために 2 つの評価を行った．1 つは単純なアプリケーションを ROS と ContextROS で実装しパフォーマンスを比較すること，もう 1 つはレイヤアクティベーションの時間を測定することである．まず，パフォーマンスの比較について説明する．挨拶アプリケーションを単純なアプリケーションとして使用し，実行時間とソースコードの行数を比較した．挨拶アプリケーションは現在の時間とその時間に対応する挨拶を表示する．このアプリケーションは実行時の都市に応じて振る舞いを変更する．例えば，東京で実行されると JST と日本語の挨拶が表示され，ロンドンで実行されると GMT と英語の挨が表示される．次に、レイヤアクティベーションの時間の測定について説明する．ContextROS はレイヤアクティベーションの際にノード間で通信を行う必要があるため、通信の対象となるノード数が増加した場合の通信時間を評価した．ここでは，複数のノードを使用して挨拶アプリケーションを実行し，すべてのノードのレイヤーアクティベーションが完了するまでの時間を測定した．

本実験は MacBook Air 上の virtual box による ubuntu 仮想環境で行った．MacBook Air の性能，仮想環境を表 2 に示す．

ContextROS と ROS の性能の比較を表 3 に示す．ContextROS ソースコード行の数は ROS 行の数よりも少なくなった．これは ContextROS が振る舞い変更を行う関数を自動的に生成するためである．このアプリケーションを 100 回実行し，実行時間の平均を比較するとそれらの間に違いはほとんど見られなかった．これは ContextROS がリフレクションを使用せずに，静的に生成された関数を使用して振る舞いを変更しているためであり，この関数呼び出しにかかる時間のみが ROS との違いとなる．関数呼び出しに要する時間は短いので実行時間にほとんど影響を与えない．

レイヤアクティベーション時間とアクティベーション対象ノード数との関係を図 7 に示す．ContextROS の非同期通信によるレイヤアクティベーションでは対象ノードがそれぞれアクティベーションを行ったノードと通信してレイヤ情報を共有するため，図 7 のように対象数の増加に伴い通信時間が長くなりレイヤアクティベーション時間が増加する．

表 2: 評価実験の環境

OS	OS X El Caption 10.11.4
CPU	1.6 GHz Intel Core i5
Memory	8 GB 1600 MHz DDR3
Virtual environment	VirtualBox 5.16
VirtualOS	ubuntu 14.04 LTS
Virtual memory	4 GB
ROS	indigo 1.11.20

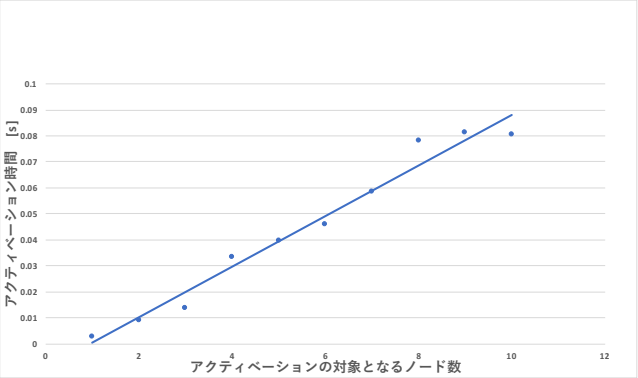

図 7: レイヤアクティベーション時間とアクティベーション対象ノード数との関係

現時点で実装されているレイヤを静的に解釈し，非同期通信であるトピック通信を用いたレイヤアクティベーションを行う ContextROS の性能評価をまとめる．ContextROS は振る舞いの変更と実行時間の観点から ROS の性能を損なうことなく COP を実現することができた．しかしながら，トピック通信によるノード間でのレイヤ情報の共有にはアクティベーションの順序が保証されず，アクティベーションの対象数が増加するにつれてアクティベーション時間が増加するという課題が残る．

6. 今後の課題

多言語実装 ROS は多言語をサポートし，様々な言語で開発されたノードが存在する．これらのノードが再利用可能であることを保証するために，ContextROS は複数の言語をサポートする必要がある．ContxtROS はノードの振る舞いを変更するために用いられる下層 COP 言語とこれらの COP 言語を管理する COP マネージャから構成される．そのため，ContextROS が複数の言語をサポートするためには各言語で振る舞いを変更する下層 COP 言語を実現し，COP マネージャが言語に依存しない形でレイヤ情報を管理できるようにすることが課題となる．

レイヤの共有方法 ROS にはノード間の通信方法としてトピック通信とサービス通信の 2 種類の通信方法が存在するが，非同期通信であるトピック通信では通信の順序が保証されない．レイヤアクティベーションの順序が保証されていない場合，システムの動作が保証さ

表 3: ROS と ContextROS の性能の比較

実装方法	実行時間 [μs]	ソースコードの行数
ROS	1653	32
ContextROS	1658	27

れないため問題が発生する．したがって，通信量が多少増加したとしてもアクティベーションの順番を保証する必要がある．

レイヤアクティベーションの一貫性 アクティベーション命令を用いる決定的なアクティベーションは制御フローをまたがる柔軟なレイヤアクティベーションを実現するが，制御が難しいという点で課題となる．特に意図しないタイミングでのレイヤのアクティベーションは振る舞いの一貫性を妨げ非常に危険である．この問題の解決策としてクリティカルセクションとトランザクション [5] の導入を検討している．

レイヤ相互作用の問題 COP ではレイヤ相互作用の問題 [6] が存在する．この問題はレイヤ同士が相互作用することが原因となって引き起こされる問題である．この問題は分散環境で複数のレイヤをアクティベートした際に発生する傾向があり，考慮する必要があると考えている．

7. 関連研究

COP の言語要素の 1 つにレイヤアクティベーションがある．ここでは既存の COP 言語でのレイヤアクティベーション [7] について説明する．

レイヤアクティベーションの代表的な方法としてアクティベーションしたい範囲をブロックで囲む方法がある．この方法を採用している言語として ContextJ [8] と ContextL [9] などが存在する．ContextJ におけるアクティベーションの方法をソースコード 3 に示す．“with” につづく括弧内でアクティベーションするレイヤを指定し，アクティベーションする範囲をブロックで囲むことでアクティベーションを行う．レイヤアクティベーションの制御が容易であり，複数の制御フローで異なるアクティブレイヤの状態を持つことができるという利点を持つが，制御フローをまたがりアクティブレイヤの状態を共有することが難しいといった欠点を持つ．

ソースコード 3: ContextJ におけるアクティベーション方法

```
with (Indoor) {
  getPos();    /*屋内での位置推定*/
}
```

ブロックで囲むアクティベーションに対して，アクティベーション命令を使用する決定的なアクティベーション方法がある．この方法を採用する言語として Subjective-C [10] などがある．Subjective-C におけるアクティベーションの方法をソースコード 4 に示す．アクティベーション命令実行後は非アクティベーション命令が行われるまで対象となるレイヤが常にアクティブとなり，アクティブレイヤの状態を制御フローをまたがり共有することができる．このため，制御フローをまたがるレイヤアクティベーションを用意に実現するが，レイヤアクティベーションに関して要求する性質（例：意図しない衝突を起こさないかどうか）を満たすかの保証は困難となる．ROS は分散システムであるため複数の制御フローが存在する．ROS で統制されたで振る舞いの変更を行うためには，このそれぞれの制御フローでアクティブレイヤの状態を共有する必要がある．そのため，ContextROS では複数の制御フローでアクティブレイヤの状態を共有できる決定的なアクティベーションが必要不可欠である．

ソースコード 4: Subjective-C におけるアクティベーション方法

```
[CONTEXT activateContextWithName:@"Indoor"];
[CONTEXT deactivateContextWithName:@"Outdoor"];
getPos();    /*屋内での位置推定*/
```

8. まとめ

ROS を使用したコンテキストアウェアなロボットの開発ではコンテキストの明示，コンテキスト依存な振る舞いのモジュール化，分散環境下でのコンテキストの共有などの問題を考慮する必要がある．本稿ではこれらの問題を解決するために ROS に COP を適用した ContextROS を提案した．ContextROS は COP に用いられるレイヤの概念を ROS に導入しコンテキストの明示とコンテキスト依存な振る舞いのモジュール化を行う．ContextROS では文法的拡張を行いレイヤの概念をレイヤ記述として実現した．分散環境下でのコンテキストの共有は ROS のトピック通信を使用したレイヤアクティベーションによって実現した．ContextROS の性能を評価するために ROS と ContextROS それぞれで簡単なアプリケーションを実装し，実行時間とソースコードの行数を比較した．その結果，ContextROS は ROS の性能を損なうことなくコンテキストに応じた振る舞いの変更を行った．

参考文献

[1] 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 7th International Workshop on Context-Oriented Programming*, ACM, p. 4 (2015).

[2] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A. Y.: ROS: an open-source Robot Operating System, *ICRA workshop on open source software*, Vol. 3, No. 3.2, Kobe, p. 5

(2009).

[3] Hirschfeld, R., Costanza, P. and Nierstrasz, O.: Context-oriented programming, *Journal of Object Technology*, Vol. 7, No. 3, pp. 125–151 (2008).

[4] Tanigawa, I., Ogura, N., Sugaya, M., Watanabe, H. and Hisazumi, K.: A structure of ac# framework ContextCS based on context-oriented programming, *Companion Proceedings of the 14th International Conference on Modularity*, ACM, pp. 21–22 (2015).

[5] Mattis, T., Rein, P. and Hirschfeld, R.: Transaction Layers: Controlling Granularity of Change in Live Programming Environments, *Proceedings of the 8th International Workshop on Context-Oriented Programming*, ACM, pp. 1–6 (2016).

[6] Watanabe, H., Tanigawa, I., Sugaya, M., Ogura, N. and Hisazumi, K.: A layer-structure diagram and a layer-interaction diagram towards a context-oriented development methodology for embedded systems, *Companion Proceedings of the 15th International Conference on Modularity*, ACM, pp. 125–130 (2016).

[7] Appeltauer, M., Hirschfeld, R., Haupt, M., Lincke, J. and Perscheid, M.: A comparison of context-oriented programming languages, *International Workshop on Context-Oriented Programming*, ACM, p. 6 (2009).

[8] Appeltauer, M., Hirschfeld, R., Haupt, M. and Masuhara, H.: ContextJ: Context-oriented programming with Java, *Information and Media Technologies*, Vol. 6, No. 2, pp. 399–419 (2011).

[9] Costanza, P. and Hirschfeld, R.: Language constructs for context-oriented programming: an overview of ContextL, *Proceedings of the 2005 symposium on Dynamic languages*, ACM, pp. 1–10 (2005).

[10] González, S., Cardozo, N., Mens, K., Cádiz, A., Libbrecht, J.-C. and Goffaux, J.: Subjective-c, *International Conference on Software Language Engineering*, Springer, pp. 246–265 (2010).