Vol.2016-ARC-219 No.10
Vol.2016-SLDM-175 No.10
Vol.2016-EMB-40 No.10
2016/3/24

コンテキスト指向プログラミング手法における ロボット事例への適用検討

小野建也† 菅谷みどり†

概要：近年ロボット環境が多様化しており，多様化した環境に応じたコンテキストアウェアな動作がロボットに求められている．コンテキストアウェアな振る舞いをプログラミングで記述する試みの一つとしてコンテキスト指向プログラミングがある.これはコンテキストに依存した振る舞いをまとめて，実行時のコンテキストに応じて振る舞いをまとまりごとに変更するものである．本研究では既存の手法であるオブジェクト指向プログラミングとコンテキスト指向プログラミングでのロボットプログラムの記述においての比較を行い，コンテキスト指向プログラミングの適用による利点を検討する．

1. はじめに

近年ロボットの小型化，高性能化により，より身近なサービスや環境でロボットが利用されるようになっている．例えばドローンは災害救助や，宅配サービスに利用されることが考えられている[1]. また，iRobot Create[4]が提供する汎用の走行ロボットは，通常の掃除サービス以外に，データセンタの温湿度管理[2]など様々なサービスに応用されるようになってきた． このように，ロボットを用いたサービスが多様化することにより，ロボットが動作する環境も，多種多様に広がってきている．

こうした広がりにおいて，ロボットは置かれている環境や状況に応じて振る舞いを変更し，最適な動作を選択することが期待される．置かれている環境や状況に適応するには，絶えず外界の情報を収集し，その変化により，振る舞いを変更する仕組みが必要である．コンテキスト指向プログラミング(Context-Oriented Programming)は，こうした環境や状況(以下，コンテキスト)の変化に適応することを目的とした提案であり，近年，ロボットへの適用が検討されている[3]. COP は，コンテキストに依存した振る舞いをモジュール化し，実行時のコンテキストの変化に応じてそれらのモジュールを切り替えるプログラミング方法として, RT-COS の仕組みをロボット適用することを提案している[13].ここでは，コンテキストに依存する振る舞いを一つのレイヤとし，それらを，コンテキストの変化により切り替えるための仕組みが提案されている．特に，リアルタイム性能などロボットで必要とされる性能要求に対して，本仕組みは適していると考えられる．しかし，実際に RT-COS を用いたサンプルは存在せず，それが既存の言語実装より優れているか否かについては十分な実証例が示されておらず，その有効性が客観的に示されていない問題がある．

そこで，本研究は，コンテキスト指向プログラミングのロボット向け実装である RT-COS と，既存のオブジェクト指向プログラミング(以下 OOP)での実装を比較し，その有効性について議論することを目的とした．目的の実現にあたり，まず初めに，具体的なロボットの動作例を示し，次に，OOP での設計，実装を行った．さらに，COP の設計，実装を行い，それぞれ設計，実装の評価を行った．その結果，設計では，COP は多様なコンテキストを扱うような大規模なアプリケーションであるロボットにおいて有用であると言えることが分かった．また，実装ではコード行数で比較し，その結果であることが分かった．本論文の構成は以下の通りである．まず 2 章で，ロボットプログラムの課題について述べ，次に 3 章で，COP の提案について述べる．4 章では，COP と OOP での設計・実装 を行い，5 章でそれらの評価，比較を行った．6 章では記述を行ったうえでの評価結果の考察を行い，7 章で全体の結論を述べる．

2. ロボットプログラムの課題

2.1 ロボットプログラムの実例

まず，ロボットプログラムの例をあげて，その課題について述べる．ここでは，複数のコンテキストを取り扱うために，ネットワークの通信状態とバッテリーの残量に応じて処理が切り替わる掃除ロボットの動作を考える．ここでは，コンテキストを，通信状態と，バッテリーの残量の状態とする．状態に応じて動作が切り替わる例を以下の図 1 に示す．

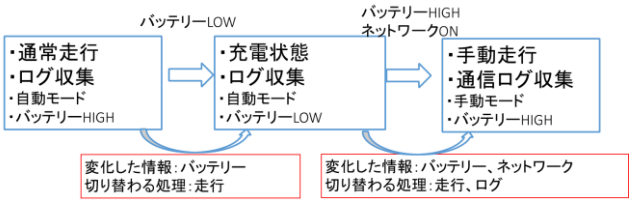

図 1 ロボットの状態の遷移

掃除ロボットを例にとって説明する(図 1). 掃除ロボットは，単体では掃除をするものであるが，様々な状況の変化により，走行方法などを変換して動作する．ここでは，バッテリーとネットワーク通信が発生する場合の，状況の変化に応じたロボットの変化について述べる．

ここでは，二つのコンテキストと，それに応じた三つの

† Shibaura Institute of Technology

Vol.2016-ARC-219 No.10
Vol.2016-SLDM-175 No.10
Vol.2016-EMB-40 No.10
2016/3/24

振る舞いの変化を扱うものとする.二つコンテキストとは,バッテリー残量と，ネットワークの接続とする.

この例では，掃除ロボット(iRobot Create) は基本動作として,掃除をしながら走行とログの収集を行うものとする.ロボットの振る舞いとして，走行を考えた場合，コンテキストにより，以下の３つのロボットの走行という振る舞いが切り替わる.

(1) 自動走行
(2) ドック探索および充電
(3) 手動走行

この振る舞いの切り換えの契機となるコンテキストには,バッテリーの残量と，通信の二つが存在する.

（振る舞い１）通常動作として，(1) の自動走行を行う．ここで自動走行とは走行中に壁を検知し，もし壁があったら方向転換をするといった走行である．自動走行では，停止することなく床をくまなく移動し掃除する．本走行では,走行が行われるに従いバッテリー残量が低下(バッテリーLOW) する．バッテリー残量が 0% になると，ロボットは自動走行し続けることができなくなる．急停止によるサービス停止や障害をふせぐために，事前にバッテリーを充電する必要がある.

（振る舞い２）充電を行うためには，自動走行をドックの探索へ切り替える必要がある．また，ロボットはドックへ戻り充電を行う．この際のロボットの振る舞いは(2) となる．また，充電が終了し(バッテリーHIGH)自動走行に戻ったとする.

（振る舞い３）このとき,外部からロボット操作のために,ネットワークを通じた，ロボットへの接続要求が発生したとする(ネットワーク). この要求に対応するためには，通常の走行ではなく通信情報の命令で動く手動走行に切り替える必要がある．また収集するログも通常のセンサ情報のみでなく通信情報を含めたものとなる.

2.2 課題

2.1 の例に示したように,（振る舞い１）から（振る舞い２）へは，バッテリー情報というコンテキストの変化により，状態変化がもたらされた．二つめは，バッテリー情報と通信情報というコンテキストの変化により,(振る舞い２)から（振る舞い３）への変化がもたらされた．ここで重要な点は，後者はログ収集と走行の二つの振る舞いが変化したことである.

またこの例では屋内のみ，掃除動作のみを想定したがこれ以外にも機能や利用される状況は考えられる．例えば屋外での走行を想定した場合がある．屋外では走行する範囲がほぼ無限と考えられるため特定のルートを走行することになる．その場合，屋外での自己位置推定のために GPS などを使い走行を行う必要があると考えられる．また，掃除だけではなく落し物探索の機能もあると考える．この場合は送受信されるデータに画像ファイルが増え，走行ルートも独自のものとなる．画像ファイルが送受信されるとなると通信のタイミングも大きく変わってくる.

このようにロボットを汎用的に利用しようと考えると,複数のコンテキストに応じて，複数の振る舞いを変更させる必要がある．しかも，複数のコンテキストに対する，複数の振る舞いの変更を実現するには，多種多様な環境に加え様々な動作を想定することになり，その動作の切り替えや切り替えに対応する情報の扱いのための処理が増大することが予測される.

2.3 既存手法

ソフトウエア開発を支援する既存の設計手法としてオブジェクト指向プログラミングがある．オブジェクト指向プログラミングではオブジェクト単位でプログラムを分割することで開発を容易にする手法である．オブジェクト指向プログラミング は本来,データ構造をオブジェクトとして表現し，プログラムはオブジェクトの相互作用として定義する[5] .

ロボットの構成をオブジェクト指向で表現するには，まず,オブジェクトとして定義するものを決める必要がある.ここでは，振る舞いが異なる部品として，センサ，モータをオブジェクトとして定義する．また，このオブジェクトを組み合わせてソフトウエアの開発を行う.

オブジェクト，メソッドごとに状況に対応した処理を記述する場合，それぞれのオブジェクト，メソッドごとに状況を管理する必要があり，一つの状況に対しての記述が様々なところに分割されて記述されることになる．前述のロボットの例の場合，走行処理とログの収集の処理の中にそれぞれバッテリー，ネットワーク接続状態に応じた分岐が書かれることになる．この場合，状況に対しての動作が様々な場所に散在することになり全体としての動作が分かりにくくなる．また状況ごとにオブジェクトを分割しクラスとしてまとめる場合，複数の状況の組み合わせ事に対してその組み合わせの分だけオブジェクトを作る必要がある．前述のロボットの例の場合, 仮にバッテリー残量を十分/不十分の 2 値，ネットワークの接続を ON/OFF の 2 値として取ったとする．この場合，想定される組み合わせは 2×2 の 4 通りになるといえる．ここにさらに屋内外の情報が入ったとするとさらに倍になり倍々になって増えていくと想定できる．こうなると動作が切り替わる状況が一つ増えるごとに組み合わせが倍になりクラスの数が増大すると考えられる.

3. 提案

3.1 目的

2 節で述べたように，ロボットソフトウエアでは多様な状況が存在し，それぞれの状況に応じた柔軟なふるまいが求められる．そのようなロボットソフトウエアでは状況に応じた処理の切り替えが多岐にわたり，処理に切り替えに

Vol.2016-ARC-219 No.10
Vol.2016-SLDM-175 No.10
Vol.2016-EMB-40 No.10
2016/3/24

要する情報も多量になり構造が複雑化する．前述の例においてはバッテリー残量とネットワークの接続状態の二つ情報とそれの組み合わせで 4 通りの状況が考えられる．またそれに加わる外部情報に対して，想定される状況の組み合わせが倍々に増えていくというものであった．このように複雑化するロボットソフトウエアの開発は既存の方法ではプログラムが大きくなり処理の分岐の複雑化する．このため多様で複雑なロボットソフトウエアに対応するには既存の手法では問題があるといえる．本研究ではこのようなロボットソフトウエアの開発に対応することができる手法を検討する．

3.2 コンテキスト指向プログラミング

状況に応じた柔軟なふるまいを設計するプログラミングのアプローチとしてコンテキスト指向プログラミング[6]がある．コンテキスト指向プログラミングはオブジェクト指向プログラミングの派生のひとつである．

コンテキスト指向プログラミングではコンテキストに応じて振る舞いをダイナミックに変更する．コンテキストとは実態（人，物，情報，場所）の状況（周辺環境，背景）を特徴づけるために用いられる情報のことで，ロボットプログラミングにおいてはセンサなどを通じて数値化されてシステム取り込まれる環境情報のことをさす．

コンテキスト指向プログラミングの大きな特徴の一つとして，コンテキストに依存した振る舞いをレイヤという単位でまとめることができる．それぞれのレイヤをコンテキストに応じてアクティベイト／ディアクティベイト（ON/OFF）させることで動作を切り替える．レイヤがアクティベイトするとそのレイヤに属する処理のみが，レイヤがアクティベイトする前の処理を上書きされる形で切り替わる．またそれぞれのレイヤは独立しており，ソースコード間で影響しあわないという特性を持っている．このためほかのレイヤの処理を考慮せずにソースの記述ができる．

これらの特性を用いてコンテキストに応じて柔軟な処理の切り替えを行う．コンテキストに対してコンテキストごとの処理としてコードを分割,記述でき,それらを組み合わせることでアプリケーションを実現する．そのため多様なコンテキストを扱うソフトウエアの開発においてコードの複雑化を軽減できると考えられる．

3.3 既存の手法と比較

既存の手法としてオブジェクト指向プログラミングを取り上げる.オブジェクト指向では分割の単位がオブジェクトである．オブジェクトの単位としては通常ものとしての単位で変数とその振る舞いという単位である．これに対してコンテキスト指向はコンテキスト(レイヤ)単位である．これはコンテキストに応じた処理のまとまりとしての単位で状況単位ともいえる．これはオブジェクトの上位概念ともとれる．例えば複数のクラスがひとつのコンテキストによって振る舞いを変える場合それらのクラスをまとめて一つのレイヤとした扱うことになる．多様な状況が想定されるロボットソフトウエアでは状況単位で処理のまとまったコンテキスト指向プログラミングが有用と考える．

3.4 既存の COP 言語

コンテキスト指向プログラミングの研究として言語の実装がある[7]．今回調査したコンテキスト指向を実装した言語に ContextJ[8]がある．それぞれ java ベースの言語で java の機能を拡張する形で実装が行われている．ContextJ ではレイヤを layer により定義している．layer はクラス内に実装される．クラス内でレイヤごとにメソッド記述する．レイヤがアクティベイトしている場合，そのレイヤ内のメソッドが実行される．また，proceed を使うことによりアクティベイトしている他のレイヤのメソッドを実行することができる．

レイヤのアクティベイトは with ブロックを用いることで実現している．with に指定したレイヤがブロック内でアクティベイトされる．レイヤの指定にはレイヤ名を用いる．ContextJ の特徴の一つとしてレイヤのアクティベイトの範囲がブロック内に限定されているというのがある．この場合レイヤのアクティベイトの部分が明示的になる．しかしレイヤをアクティベイトするたびに明示的に宣言する必要があるため with ブロックがプログラム中に散在する．

このほかにも COP を実装している言語として JCOP[9]や EventCJ[10]，ServalCJ[11]などがある．

これ以外にコンテキスト指向プログラミングを実装しているものとして言語ではなく，C#の COP フレームワークとして RT-COS[12,13]がある．今回はこれを用いて実装を行っている．RT-COS の詳細については後述する．

3.5 ロボットに適用した場合

実際にロボットプログラミングに適用することを考える．動作を想定するアプリケーションについて，掃除ロボットを使うものとする．ロボットのふるまいとして前述のネットワークの通信状態とバッテリーの残量に応じて処理が切り替わる掃除ロボットを考える．この場合バッテリー残量とネットワーク接続状態がコンテキストに当たる．またこのコンテキストに応じた処理をそれぞれまとめることでレイヤを記述することになる．以下の図 2 では走行処理とロギング処理において，それぞれの処理がクラス内にまとめられている場合とレイヤにまとめた場合を示した．

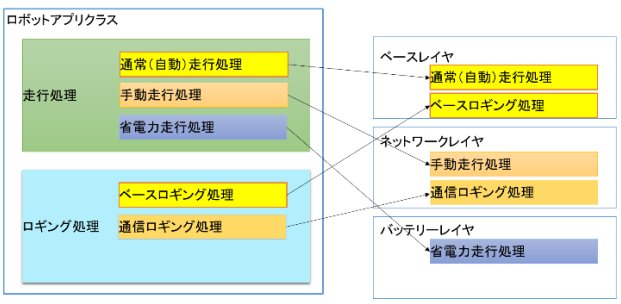

図 2　レイヤ分割の例

Vol.2016-ARC-219 No.10
Vol.2016-SLDM-175 No.10
Vol.2016-EMB-40 No.10
2016/3/24

4. 設計・実装

4.1 RT-COS 概要

本研究では，実装は，C#で COP の開発環境を実装しているフレームワーク RT-COS[12,13]を用いて行った．RT-COS では前述の ContextJ と異なりレイヤのアクティベイトの範囲が明示的なものではなくレイヤのアクティベイトとディアクティベイトを指示により切り替える．（図 3）

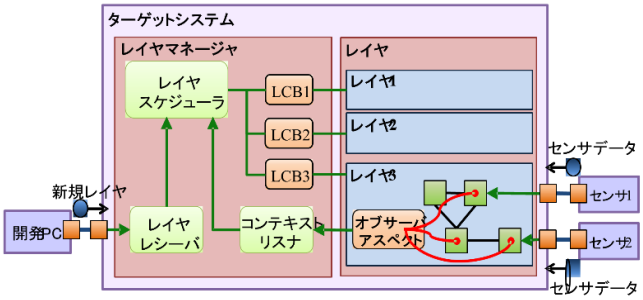

図 3　RT-COS の構造[12,13]より

RT-COS では切り替えるレイヤを事前にユーザがレイヤとして登録する必要がある．また，レイヤの切り替えは専用の命令により直接レイヤを指定してレイヤのアクティベイト/ディアクティベイトを行う．レイヤのアクティベイト/ディアクティベイトの命令があると別スレッドでレイヤスケジューラがレイヤのアクティベイトを行う．レイヤスケジューラでは複数のレイヤのアクティベイトが同時に起こった時にどのレイヤからアクティベイトするかを決定する．この時の優先度を決定はレイヤの登録時に行う．
また，コンテキストの判定とレイヤのアクティベイト部分をレイヤの記述からアスペクトとして分離する仕組みとしてオブザーバアスペクトがある．通常のアプリケーションの処理部分とコンテキストに応じた動作の切り替え部分を分離することでコードの見通しをよくする．コンテキストの更新部分にあたるメソッドにオブザーバアスペクトを適用(ウィーブ)する．オブザーバアスペクトの内部で適用したメソッドから判定を行うコンテキストを取得し，コンテキストに応じたレイヤのアクティベイトを行う．

4.2 設計

コンテキスト指向の有用性の検証のため実際のアプリケーションの設計を行った.また既存の手法との比較としてオブジェクト指向での設計も行った．アプリケーションは前述のネットワーク接続状態とバッテリー残量に応じて動作を変更するロボットアプリケーションとした．それぞれの記述方法ではロボットの動作の記述として掃除処理，と走行処理とログの収集処理があるものとする．実際に動作させるロボットは iRobotCreate2[4]を用いた．

掃除処理では掃除を行うためブラシのモータとバキュームを起動する．またバッテリー残量が低下した場合省電力のため，ブラシとバキュームを停止する．走行処理では通常時は壁を判定しながら方向転換を繰り返し，前面をくまなく走行を行うものとする．また充電残量が低下した場合ドックの位置を探索しながら走行を行う．ログ収集処理ではセンサ情報をファイルに書き出すものとした．今回のアプリケーションではコンテキストとして充電残量を利用したので書き出す情報は充電残量とした．またネットワークが接続状態になった場合，通信によって送られてきた情報も書き出すものとする．

またネットワーク接続受付を行っており起動時に接続待ちを開始し，別スレッドで接続要求があるまで接続受付を行うものとする．

オブジェクト指向ではコンテキスト指向と振る舞いの似たストラテジーパターンで設計を行った．それぞれの状態を記述するクラスとしてネットワークの ON/OFF とバッテリー残量の HIGH/LOW のそれぞれ 2 通りの組み合わせで 4 つのクラスを切り替えて動作するものとする（図 4）．またそれぞれのクラスに掃除の処理の clean メソッド，走行処理の drive メソッド，ログ収集の log メソッドを記述する．UpdateContext では各種環境情報から処理を切り替えるコンテキストの値を更新する．ChangeStrategy ではコンテキストに応じてロボット動作の切り替えを行う．

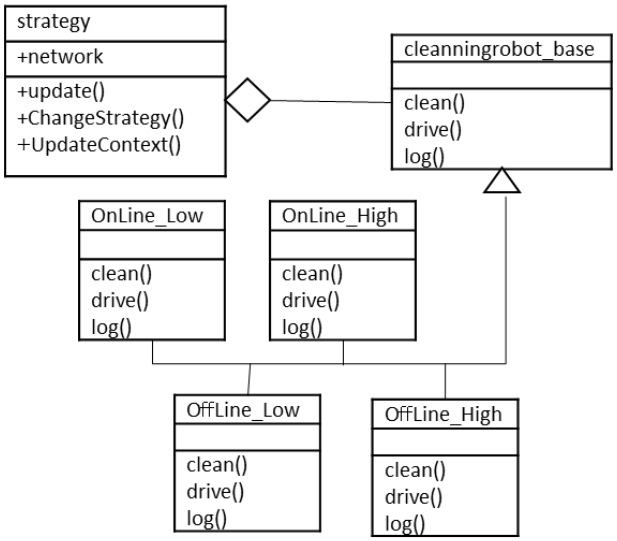

図 4 ストラテジーパターンクラス図

コンテキスト指向では，動作の切り替えを行うそれぞれアクティベイト/ディアクティベイトを行い切り替わるレイヤ（図 5）として，AutoModeLayer，LowBatteryLayer，OnLineLayer の 3 つを切り替えて動作させるものとした．通常は AutoModeLayer が動作しており，バッテリー残量が低下した場合 LowBatteryLayer がアクティベイトし，ドックを探索し充電を行うものとする．またネットワーク接続状態になった場合 OnLineLayer がアクティベイトし，手動走行を行う．それぞれのコンテキストは Sensor クラスと Network クラスより取得を行う．Sensor ではロボットのセンサ値を受け取り，格納する．Network ではソケット通信の管理と接続状態の受け渡しを行っている．コンテキストの判定はそれぞれのレイヤのオブザーバアスペクトにより

Vol.2016-ARC-219 No.10
Vol.2016-SLDM-175 No.10
Vol.2016-EMB-40 No.10
2016/3/24

行われる．オブザーバアスペクトはコンテキストの対応するメソッドにウィーブされる．またオブザーバアスペクト内でレイヤのアクティベイトが行わる．

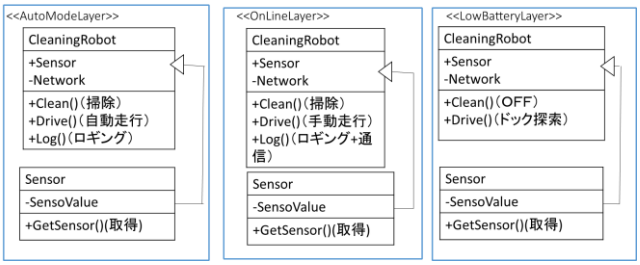

図 5　レイヤ

4.3 実装

走行処理の実装では自動走行処理はロボット前方のバンパセンサを利用し，壁と衝突した場合方向を転換する．またバッテリー残量が低下した場合は赤外線センサの値を判定しドックに向かって走行を行う．手動走行処理では受信した情報によりアクセル，ブレーキ，ターンを行う．

掃除処理は通常時では起動時にブラシのモータとバキュームの起動を行う．またバッテリー残量が低下した場合は省電力のためブラシのモータとバキュームの停止をして掃除動作を停止する．さらに通信状態ではブラシのモータとバキュームの起動と停止を操作できるものとした．

ログ収集処理ではバッテリー情報を書き出す．また通信状態では受信したデータを追加で書き出すものとした．

オブジェクト指向での実装ではコンテキストに応じた処理の切り替えを strategy クラスの ChangeStrategy メソッドで行うものとした．また処理の切り替えに用いるコンテキストの更新を UpdateContext メソッドで行うものとした．それぞれの状況に対応したクラスはコンテキストに応じてロボットの動作のクラスのインスタンスを付け替えることで動作の切り替えを実現した．想定する状況は自動走行かつバッテリー残量が十分である Offline_High，通信接続中かつバッテリー残量が十分である Online_High, 自動走行かつバッテリー残量が不足している Offline_Low, 通信接続中かつバッテリー残量が不足している Online_Low である．

これに対してコンテキスト指向ではそれぞれの状況に対応した処理をレイヤとして定義した．またレイヤごとにオブザーバアスペクトを用いて動作の切り替え部分を記述した．それぞれのレイヤごとにレイヤのアクティベイトに用いるコンテキストに対応したメソッドにアスペクトをウィーブし，コンテキストの変化に応じてレイヤのアクティベイトを行うものとした．

OnLineLayer ではネットワークの接続中の処理を行う．外部からの接続要求があった場合は通信がはじまったとして OnLineLayer をアクティベイトする．OnLineLayer では通信情報に応じた手動走行を行い，通信ログを出力するものとする．LowBatteryLayer のアクティベイトの判定方法は上記のオブジェクト指向と同様にバッテリーの残量の割合で行っている．また通常時は AutoModeLayer がアクティベイトしており，自動走行，掃除，ログ収集をそれぞれ行うものとした．

5. 評価

5.1 評価方法

定量的な評価方法としてコード行数，保守容易性（コードメトリックス値）[14]を用いて評価を行った．保守容易性はコード行数, サイクロマティック複雑度[15], Halstead[16]複雑度を用いたコードの保守性を表し，数値が高い方が保守性が高く 0 から 100 の値をとる．

サイクロマティック複雑度とは条件分岐の複雑さを示すもので，判定条件の数をNとすると複雑度V_1は

$$\mathrm{V_1 = N + 1}$$

であらわされる．

また Halstead 複雑度とは演算子やオペランドの複雑さを示すもので，演算子の種類をn_1，オペランドの種類をn_2，演算子の数をN_1，オペランドの数をN_2とすると複雑度V_2は

$$V_2 = (N_1 + N_2)\log(n_1 + n_2)$$

であらわされる．

保守容易性 M は$\mathrm{V_1}$，V2，コード行数 L より

$$\mathrm{M = 171 - 5.2 * \log V2 - 0.23 * V1 - 16.2 * \log L}$$

であらわされる．

5.2 評価結果

表 1 に結果を示した．全体としては OOP で記述した方がコード行数は多く, 保守容易性は COP の方が高い結果となった．COP の方がコード行数が多くなった理由としては RT-COS を用いるための API の利用と通常の継承とは異なるベースレイヤの記述の必要があったためと思われる．また保守容易性の値において OOP の方が低い値が出た理由は，ストラテジーの切り替え部分のサイクロマティック複雑度が比較的高く，処理を切り替える部分の複雑な分岐がひとつのクラスに集約されているためだった．これに対して RT-COS レイヤの切り替え部分のオブザーバアスペクトはそれぞれで分岐を行っているので個々のサイクロマティック複雑度が低い．これはそれぞれが独立しており一か所に変更を加えてもほかの個所への影響が少ないといえる．

表 1　コード評価

	コード行数	保守容易性
オブジェクト指向	437	79
コンテキスト指向	483	83

COP を用いて記述をした主観的な評価としてはレイヤを用いて設計を行う場合オブジェクト指向とは別の観点でモジュールを分ける必要があり，OOP の設計方法，経験だけでは設計することが難しい．例えば複数のレイヤがアクティベイトする場合，どのレイヤの優先度が高いか，どのメソッドが優先的に呼ばれるかなどがある．一方でそれぞれのレイヤ内では他のコンテキストについて把握する必要

Vol.2016-ARC-219 No.10
Vol.2016-SLDM-175 No.10
Vol.2016-EMB-40 No.10
2016/3/24

がなく，状況ごとの一つのアプリケーションとして見通しがいいといえる．そのため記述のために他のレイヤを考慮する必要がなく個々のレイヤは比較的記述はしやすい．

6. 考察

評価からコード行数ではコンテキスト指向プログラミングの方が多いという結果が出たがコードの保守性は高いという結果が出た．このためコンテキスト指向プログラミングに有用性がある可能性があるといえる．しかしコンテキスト指向プログラミングの方が高い保守性が出た理由の一つとして分岐がそれぞれのオブザーバに分離しており簡潔に書かれていたからというのがある．それに対してオブジェクト指向はストラテジーの切り替え部分の一か所なのでサイクロマティック複雑度が高いと判定されている．これはコンテキスト指向プログラミングでの部分ではなく RT-COS の機能によるものともいうことができる．よって有用性が出たのはコンテキスト指向プログラミングではなく RT-COS が要因といえるかもしれない．

またコード行数においてはコンテキスト指向プログラミングの方が多いものとなった．しかしコンテキスト指向プログラミングではオブジェクト指向プログラミングにあったコードの重複が少なくなっている．例えば低電力状態では走行処理が通信状態にかかわらずドック探索である．このようにコード重複の多さにおいてはコンテキスト指向プログラミングが優れているといえるが有用である結果が得られなかった．しかし想定される状況が増えるとするとクラスの数が増大することを考えるとアプリケーションの規模が大きくなるにつれて有用性は高まっていくのではないかと思われる．

設計，実装を行った主観としては設計の段階では既存の考え方，方法では難しいといえる．このため開発においてはアプリケーションの全体を正確に把握する必要がある．それに対して，それぞれのレイヤの記述は比較的容易なので記述, 管理の段階において有用性があるといえる. また，設計においての手法が確立していないということと，経験がないということを考えると設計，記述の慣れによっては開発の難易度が下がるかもしれない．

7. 結論

本研究では定量的な評価においてコンテキスト指向プログラミングの方が保守性が高く，多くのコンテキストを扱うロボットアプリケーションにおいてコンテキストごとの処理の切り分けが行えることで組み合わせの複雑化を回避できると思われる．しかし有用な結果が出た要因がコンテキスト指向プログラミングであっただけではないともいえる．また，アプリケーションの規模においてもプログラミングの有用性が上がると思われるという見解が得られた．コンテキスト指向プログラミングの不十分な点として設計の手法，言語としての実装がある．実際の開発に用いるには設計の手法が確立しておらず，今後も適用例を考える必要がある．また実装を行うための言語を利用するのが困難なため実際の記述，動作を確認することができない．今後これらの課題を解決する必要があるといえる．

参考文献

[1] “Amazon Prime Air” Amazon http://www.amazon.com/b?node=8037720011(2016/02/23)
[2] “IBM Roomba-Based Robot Measures Data Centre Heat” http://www.techweekeurope.co.uk/workspace/ibm-roomba-data-centre-heat-emc-117925 (2016/02/23)
[3] Harumi Watanabe, Midori Sugaya, Ikuta Tanigawa, Nobuhiko Ogura, Kenji Hisazumi
A Study of Context-Oriented Programming for Applying to Robot Development
COP'15 Proceedings of the 7th International Workshop on Context-Oriented Programming Article No. 4
[4] “iRobotCreate2”.iRobot. http://www.irobot.com/About-iRobot/STEM/Create-2.aspx (2016/02/03)
[5] Eugene Kindler,Ivan Krivy. (2011). “Object-Oriented Simulation of systems with sophisticated control”. International Journal of General Systems: 313–343.
[6] Rovert Hirschfeld,Pascal Costanza,Osccar Nierstarasz, “Context-Oriented Programming”. Journal of Object Technology. 2008, 7(3), p.125-151
[7] Malte Appeltauer,Robert Hirschfeld,Michael Haupt,Jens Lincke,Michael Perscheid. “A Comparison of Context-oriented Programming Languages”. COP’09 International Workshop on Context-Oriented Programming Article. 2009, No. 6
[8] Malte Appeltauer,Robert Hirschfeld,Michel Haupt, Hidehiko Masuhara. “ContextJ:Context-oriented programming with Java”. コンピュータソウエア,28(1):272-292, 2011
[9] Malte Appeltaue,Rovert Hirrschfeld,Hidehiko Masuhara, Michael Haupt,Kazunori Kawauchi. “Event-specific software composition in context-oriented programming”. In Proceedings of the International Conference on Software Composition 2010 (SC’08), volume 6144 of LNCS, pages 50-65, 2010.
[10] Tomoyuki Aotani,Tetsuo Kamina,Hidehiko Masuhara. “Featherweight EventCJ: a core calculus for a context-oriented language with event-vased perinstance layer transition”. In COP’11, 2011.
[11] Tetsuo Kamina,Tomoyuki Aotani,Hidehiko Masuhara. “Generalized layer activation mechanism through contexts and subscribers”. In 14th International Conference on Modularity, MODULARITY 2015, pages 14–28
[12] Ikuta Tanigawa “C#ベースのコンテキスト指向プログラミング開発環境”http://tanigawaikuta.bitbucket.org/(2016/01/06)
[13] Ikuta Tanigawa,Nobuhiko Ogura,Midori Sugaya,Harumi Watanabe,Kenji Hisazumi.“A structure of a C# Framework ContextCS Based on Context-Oriented Programming”.MODULARITY Companion 2015 Companion Proceedings of the 14th International Conference on Modularity, Pages 21-22.
[14] “コードメトリックス値”. MicroSoft.https://msdn.microsoft.com/ja-jp/library/bb385914.aspx (2016/02/03)
[15] Thomas J. McCABE. “A Complexity Measure”.IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-2, NO.4, DECEMBER 1976
[16] Maurice H. Halstead. “Elements of Software Science”.Elsevier Science Inc. New York, NY, USA ©1977.