修士論文

# モデル駆動開発における例外発生に伴う複雑度の抑制に関する研究

阿部　航

主指導教員 鈴木正人

北陸先端科学技術大学院大学
先端科学技術研究科
（情報科学）

令和３年３月

Abstract

In the process of software development, the source code becomes complicated every time specifications are added or changed. It becomes difficult to grasp the contents. As a system development method, UML (Unified Modeling Language) is a language that unifies the description when modeling a system in object-oriented design, and because it is visualized, it is easier to understand the contents than the source code. Model Driven Development (MDD) is a method that automatically generates a source code template from a UML model, and consists of a class diagram, sequence diagram, use case diagram, etc. as a model. MDD makes it easy to maintain model and code consistency in model development. However, UML itself may become complicated and maintainability may deteriorate. The sequence diagram itself, in which exceptions occur due to the addition and change of specifications and their processing appears, becomes complicated, and the maintainability deteriorates, which adversely affects the automatically generated source coat.

The purpose of this study is to propose a method to suppress the complexity of software that should consider numerous exception sequences for the addition of functions in MDD. The complexity of the sequence diagram is measured using several complexity metrics, and the factors that increase the complexity due to the addition of functions are clarified.

First a case study of system development is performed. Design the target system and upgrade it by adding functions multiple times. As the configuration becomes more complicated, the class diagram and sequence diagram become more complicated, which affects the decrease in maintainability. Therefore, each version evaluates this degree of complexity to identify the cause of the rapid increase in complexity. The system developed in the case study is a route search problem. There is a moving body and a mesh-like movable area, and the current position of the moving body can be obtained from the position detection device. The direction of movement is limited to four directions, north, south, east, and west, and the start point and end point of movement are given from the outside. In addition, the movement of the moving body to the movement start point is excluded. The functional requirement of version 1 is to move in the shortest path, and the constraint is not to go back. Class diagrams and sequence diagrams designed to meet this requirement were created, and the complexity of the sequence diagrams was measured using RFC(Response for a Class). As a result of measurement for each UC (Use Case), the complexity of all UC0, UC1 and UC2 was 1.

The functional requirement of version 2 is that the required time is set for each route and the sum of them is minimized. Also, as a constraint, "required time"

is added as information required for route calculation.The functional requirement of version 2 is that the required time is set for each route and the sum of them is minimized. Also, as a constraint, "required time" is added as information required for route calculation. As a result of RFC measurement for each UC, the complexity was unchanged for UC0 and UC1 and UC2 was 3. This value is three times that of V1.

The functional requirement of version 3 is that the required time changes depending on the time of day. In addition, "current time" is added as information required for route calculation. As a result of RFC measurement for each UC, the complexity was unchanged for UC0 and UC1 and UC2 was 5. This value is five times that of V1. In addition, when measuring with RFC considering the parameters, V3 was 11 for UC2, showing a significant increase.

When exception handling was added in version 4, the complexity increased sharply due to the increase in execution paths. Therefore, the reason for the rapid increase is that there are many messages with arguments in the execution path that accompanies the occurrence of an exception.

In order to suppress the increase in complexity, we propose a method of grouping the objects that appear in the sequence diagram. It was confirmed that this method can reduce the complexity.

# 目 次

# 図 目 次

# 表 目 次

# 第1章　はじめに

## 1.1　研究背景

ソフトウェア開発の過程において、仕様が追加･変更される毎にソースコードは複雑になり内容の把握は難しくなる。システム開発手法としてUML(Unified Modeling Language) は、オブジェクト指向設計においてシステムをモデル化する際の記述を統一した言語であり、視覚化されるためソースコードよりも内容の把握が容易である。

また、モデル駆動開発 (Model Driven Development 以下、MDD) は UML のモデルからソースコードのひな型を自動的に生成する手法であり、モデルとしてクラス図、シーケンス図、ユースケース図などから構成される。MDD はモデル開発におけるモデルとコードの一貫性保持が容易である。

しかし、UML 自体が複雑になり保守性が悪化する可能性がある。仕様の追加・変更に伴う例外発生とその処理が出現するシーケンス図自体が複雑になり、保守性が悪化することで自動生成されるソースコートにも悪影響を及ぼす。

## 1.2　研究目的

本研究の目的は MDD における機能追加に対し、多数の例外シーケンスを考慮するべきソフトウェアの複雑度を抑制する手法を提案することである。システム開発の成果物であるシーケンス図に対していくつかの複雑度指標を用いて複雑度を測定し、機能追加による複雑度の増加要因を明らかにする。

## 1.3　本論文の構成

本論文は、本章を含めて全 5 章で構成される。

第 2 章にてソフトウェアメトリクスに関する関連研究と技術を紹介する。

第 3 章にて、システム開発のケーススタディを行い、機能追加による例外発生がモデルの複雑度にどのような変化をもたらすかを指標を用いて計測し、その要因を明らかにする。

第 4 章にて、複雑度の値を抑制する手法をモデルに適用し、値抑制の評価を行う。

第 5 章にて、まとめと今後の課題を述べる。

# 第2章　関連研究と技術

## 2.1　関連研究

青田の研究 [1] は、ソースコードを分析する手法のソフトウェアメトリクスが抱えている問題を挙げ、ソフトウェアの品質向上という目的に対して十分ではないことを述べている。特に、指標の中でも複雑度の値は品質を客観的に示しやすいとし、機能要求実現のための条件分岐の多さは複雑度の高さと比例するため、機能要求の質や量についても考慮する必要があるとしている。

津田の研究 [2] は、UML のクラス図を対象とした品質測定手法を提案している。従来の手法は設計工程の成果物の抽象度が統一されていない場合が多く、品質測定に課題があると述べている。そこで開発分野から独立した単一のクラス図から問題個所を検出することで評価の抜け漏れを防ぐ手法の評価実験を行い、提案メトリクスの有用性を示している。

米山の研究 [3] はソフトウェア保守の中でも適応保守を対象としており、モデル駆動開発における成果物のクラス図とシーケンス図を CK メトリクスにより計測し、適応保守を進めるにつれて複雑度が増加することを確認している。また、機能追加に伴う条件分析の値が考慮されていないという問題に対して改良メトリクスを提案し、複雑度の抑制に有効であることを確認している。

## 2.2　CK メトリクス

CK メトリクス [4] とは、Chidamber と Kemerer が 1994 年に提案したソフトウェアメトリクスである。オブジェクト指向開発の設計における複雑度を測る指標として広く用いられている。CK メトリクスは以下の 6 つの指標から複雑度を測定する。

**WMC(Weighted methods per class)**
　　クラス内のメソッドの合計によりクラスの複雑度を示す

**RFC(Response for a Class)**
クラスがメッセージを受け取った際に、実行するメソッドの数を示す

**NOC(Number of Children)**
クラスが持つサブクラスの数を示す

**CBO(Coupling Between Object classes)**
クラスがメソッドとインスタンス変数を参照する他クラスの数を示す

**DTI(Depth of Inheritance Tree)**
クラスのオブジェクト階層のルートからの継承レベルの深さを示す

**LCOM(Lack of Choesion in Methods)**
クラス内でインスタンス変数を共有しないメソッドのペア数から共有するメソッドのペア数を差し引いた数であり、クラスのメソッド内の凝集性を示す

次章では、システム開発の成果物であるシーケンス図に対し、RFC と WMC を用いて複雑度の測定を行う。

# 第3章　複雑度増加の要因分析

## 3.1　対象システム

経路探索問題を例とする。移動体と網目状の移動可能範囲があり、移動体は位置検出装置から現在位置を取得できる。移動方向は東西南北の 4 方向に限定され、移動開始点、移動終了点は外部から与えられる。また、移動体の移動開始点への移動は対象外とする。

| | |
|---|---|
| 目標 | 制約に沿った開移動開始点から移動終了点までの移動経路を求める |
| 制約 | 遠回りしてはいけない |

本ケーススタディでは対象システムを設計し、複数回の機能追加によるバージョンアップを重ねる。各バージョンでこの複雑度の度合いを評価し、急激に複雑度が増加する原因を特定する。

## 3.2 バージョン1

### 3.2.1 機能要求

経路探索システムの機能要求を以下に示す。

| | |
|---|---|
| 機能要求 | 移動開始点と移動終了点を与えられたときに、それらを最短で移動する経路を返す |

バージョン 1 ではこの制約は最低限のものとする。すなわち、後戻りしない。

### 3.2.2 設計

経路探索システム（バージョン 1）は以下の要素から構成される。

- 移動体:移動可能範囲内を走行する。現在位置と方向を保持し、現在位置が移動終了点と一致するまで移動方向を決定し、移動を繰り返す
- 移動可能範囲:走行体が移動可能な範囲を定義する
- 移動要求:移動開始点と移動終了点の座標を保持する
- 位置検出装置:移動体の現在位置を通知する

クラス図を図 3.1 に示す。

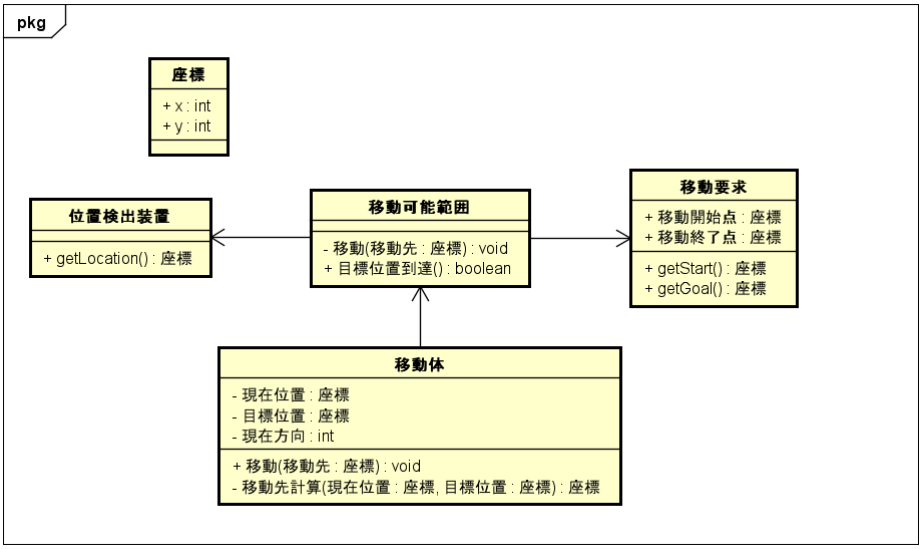


図 3.1: クラス図 V1

移動体の主たる動作は以下のメッセージから構成される。また、ユースケース (Use Case) は UC0、UC1、UC2 の 3 つがある。

**UC0**

1:移動体は移動可能範囲から目標位置を取得する

**UC1**

2:移動体は位置検出装置から現在位置を取得する

**UC2**

3:移動体は現在位置と目標位置から移動先を計算する
4:移動体は移動先に移動する
4.1:移動可能範囲は移動先が目標位置に到達したか調べる

メッセージ 3 では移動候補となる東西南北の 4 方向を調べる。シーケンス図を図 3.2 に示す。

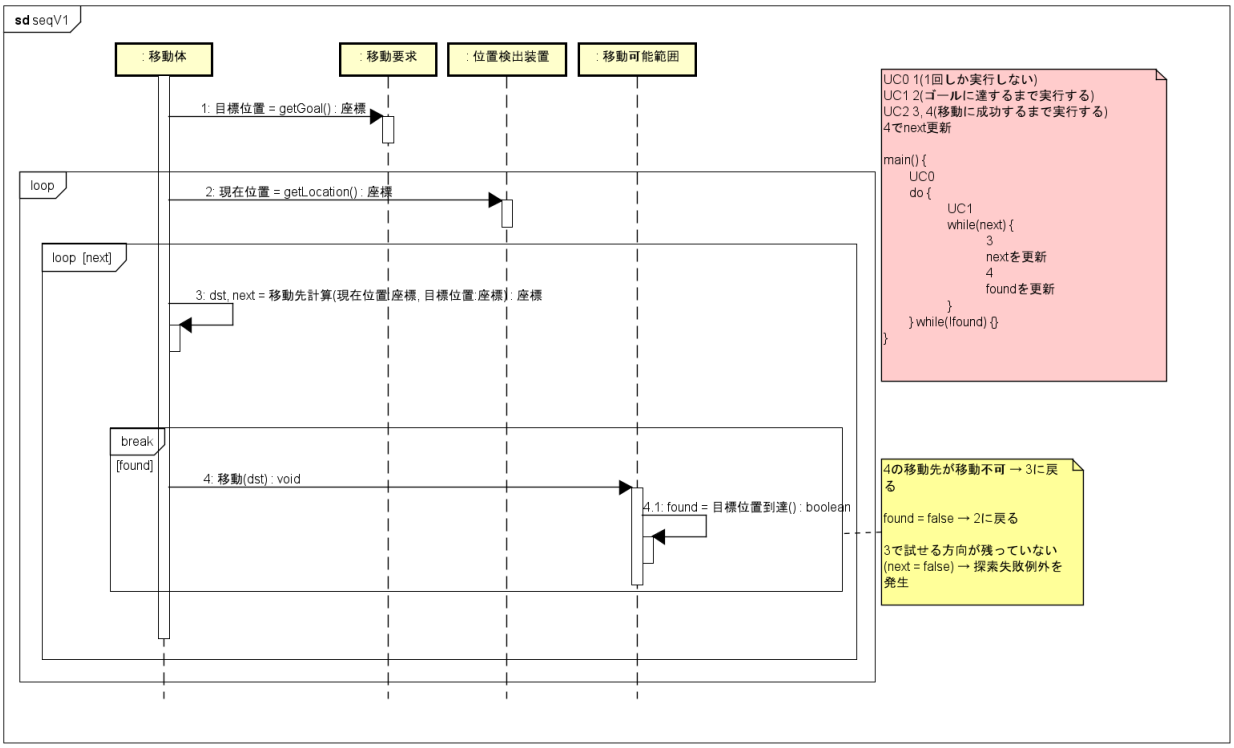


図 3.2: シーケンス図 V1

シーケンス図 3.2 のループ処理「loopo」および「loop (next)」の役割と終了条件を以下に示す。

**loop**
: 移動するたびに移動体の現在位置を取得する。終了条件は目標位置に到達することである。

**loop (next)**
: 4 方向の移動先計算を行う。終了条件は移動に成功することである。

### 3.2.3 複雑度評価

機能要求実現によるシーケンス図中の代替シーケンス増加に伴い、実行パスは増加する。実行パスの増加は複雑度の増加に影響し、保守性が低下する。ゆえに、シーケンス図におけるオブジェクト間の交信数に基づいた複雑度指標を定め、計測を行う。

複雑度指標と RFC(Response for a Class) を用いる。クラスがメッセージを受け取った際に、実行するメソッドの数を指標である。ただし、オブジェクト間のメッセージ交信と引数は以下のように取り扱う。

- 他のオブジェクトとのメッセージ交信は 1
- 自オブジェクトとのメッセージ交信は 0
- 引数の個数は複雑度に影響を与えない

シーケンス図 3.2 の複雑度計測をする。

**UC0**

1:目標位置 = getGoal():座標
　他オブジェクトとの交信のため 1

**UC1**

2:現在位置 = getLocation():座標
　他オブジェクトとの交信のため 1

**UC2**

3:dst, next = 移動先計算 (現在位置:座標, 目標位置:座標):座標
　自オブジェクトとの交信のため 0

4:移動 (dst):void
　他オブジェクトとの交信のため 1

4.1:found = 目的地到達 ():boolean
　自オブジェクトとの交信のため 0

V1 のシーケンス図の複雑度をユースケース毎に計測した結果が表 3.1 である。

表 3.1: 複雑度指標の計測結果

| | V1 |
|---|---|
| UC0 | 1 |
| UC1 | 1 |
| UC2 | 1 |

## 3.3　バージョン2

### 3.3.1　機能要求の変更

バージョン 2 では機能要求を以下のように変更する。

| | |
|---|---|
| 機能要求 | 経路ごとに所要時間が設定されており、その和を最短にする |
| 制約 | 経路計算に必要な情報として「所要時間」が追加<br>移動体のみでは最短経路の計算ができない |

移動先の計算に用いるパラメータが追加されることを考慮し、計算は外部システムに委託し、移動体は移動のみ行う拡張を施す。よって経路探索アルゴリズムという新たなオブジェクトを新たに作成した。

### 3.3.2　設計の変更

追加、変更された構成要素を以下に示す。

- 経路探索アルゴリズム:移動体の情報 (現在位置, ゴール地点, 所要時間) から移動方向の計算を行う
- UC1 の 2 及び UC2 全体の責務が移動体から経路探索アルゴリズムに変更
- 辺情報:各辺の重み情報を保持する

クラス図を図 3.3 に示す。

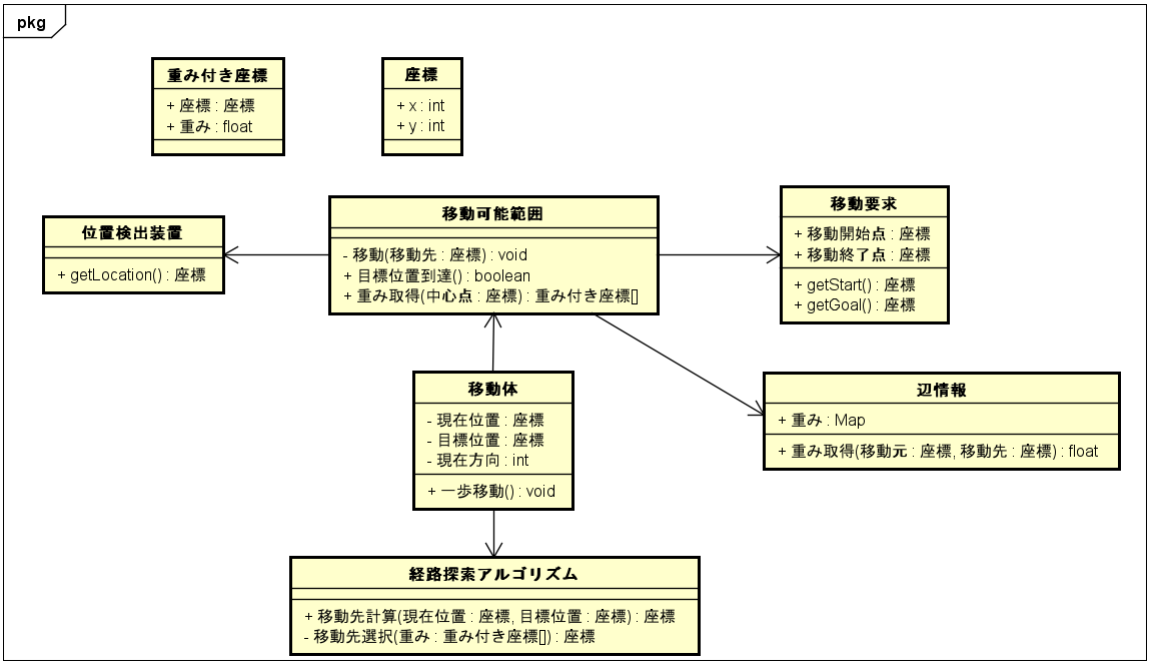


図 3.3: クラス図 V2

移動体の主たる動作の追加・変更を以下に示す。

**UC2**

3:移動体は経路探索アルゴリズムの移動先計算関数を呼び出す
3.1:経路探索アルゴリズムは移動可能範囲オブジェクトから重み情報のリストを取得する
3.2:経路探索アルゴリズムは重み情報のリストから移動先を選択する

シーケンス図を図 3.4 に示す。

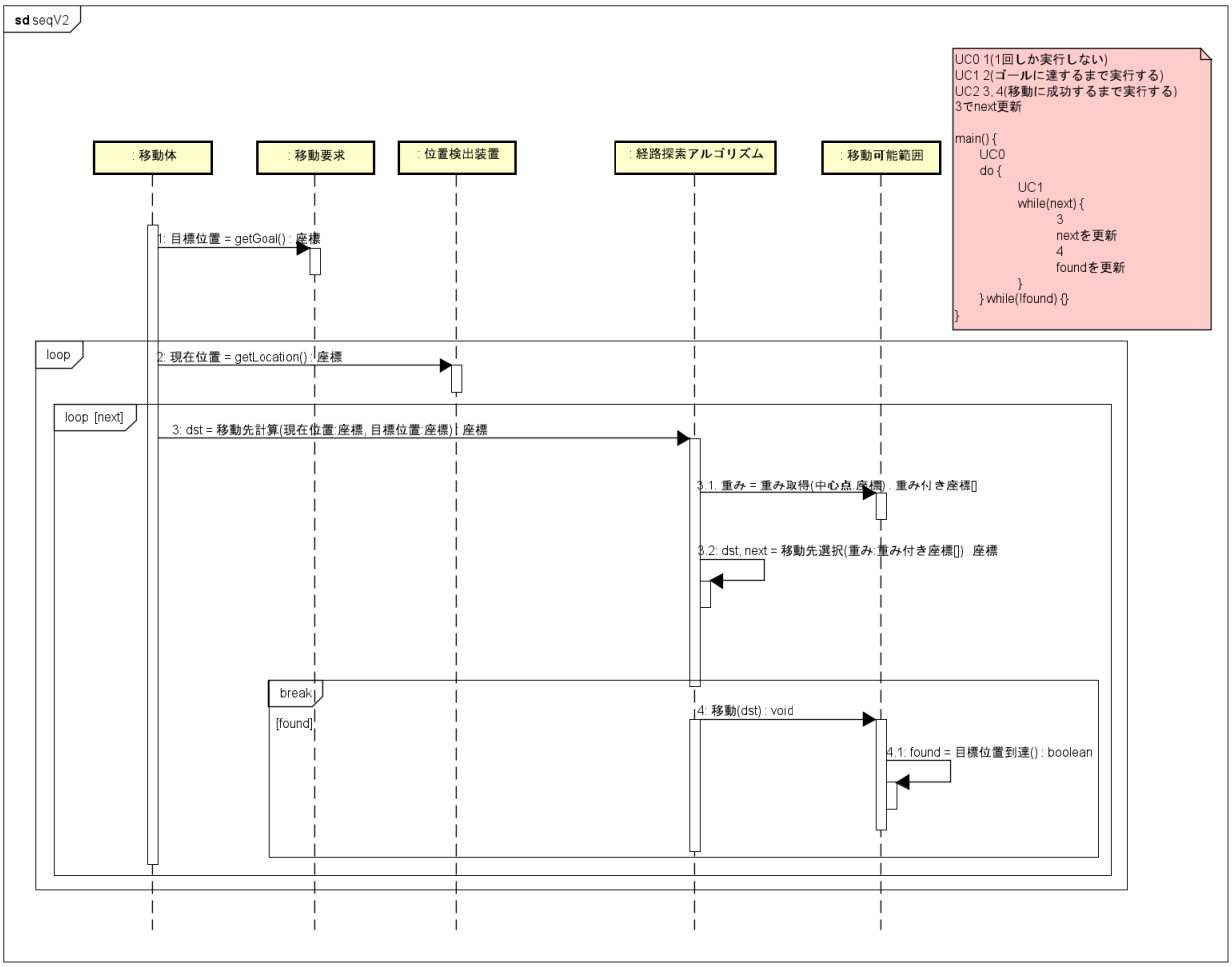


図 3.4: シーケンス図 V2

### 3.3.3 複雑度評価

シーケンス図 3.4 の複雑度計測をする。

**UC0**

1:目標位置 = getGoal():座標
　他オブジェクトとの交信のため 1

**UC1**

2:現在位置 = getLocation():座標
　他オブジェクトとの交信のため 1

**UC2**

3:dst, next = 移動先計算 (現在位置:座標, 目標位置:座標):座標
　他オブジェクトとの交信のため 1

3.1:重み = 重み取得 (中心点:座標):重み付き座標 []
　他オブジェクトとの交信のため 1

3.2:dst, next = 移動先選択 (重み:重み付き座標 []):座標
　自オブジェクトとの交信のため 0

4. 移動 (dst):void
　他オブジェクトとの交信のため 1

4.1found = 目的地到達 ():boolean
　自オブジェクトとの交信のため 0

V2 のシーケンス図の複雑度をユースケース毎に計測した結果が表 3.2 である。V1 に比べ UC2 の複雑度が 3 倍になった。

表 3.2: V2 の複雑度指標の計測結果

| | V2 |
|---|---|
| UC0 | 1 |
| UC1 | 1 |
| UC2 | 3 |

## 3.4　バージョン3

### 3.4.1　機能要求の変更

バージョン3では機能要求を以下のように変更する。

| | |
|---|---|
| 機能要求 | 時間帯によって所要時間が変化し、最短の経路を選択する |
| 制約 | 経路計算に必要な情報として「現在時刻」を追加 |

### 3.4.2　設計の変更

V2では重みの値はクラス「辺情報」が保持しており、この値は変化しない。V3では重みの時間変化を取り扱うために、新たに2つのクラスを導入する。クラス「重み関数」の役割は時刻から重みを与えることであり、クラス「重み関数マップ」の役割は辺の両端の座標からクラス「重み関数」のオブジェクトを与えることである。

重みはクラス「重み関数」のメソッドの戻り値として実現される。クラス「辺情報」からクラス「重み関数」への関連を追加する。

クラス図を図3.5に示す。

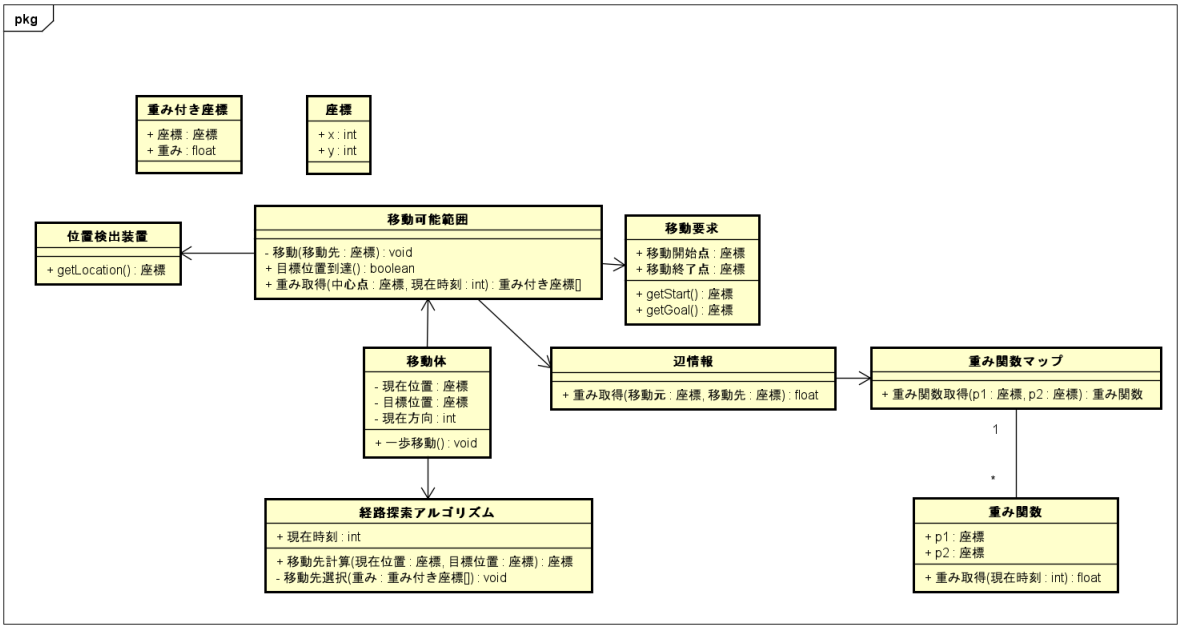


図 3.5: クラス図V3

移動体の主たる動作の追加・変更を以下に示す。

**UC2**

3.1.1:移動可能範囲は重み関数マップから重み関数を取得する
3.1.2:移動可能範囲は重み関数から重みを取得する

シーケンス図を図 3.6 に示す。

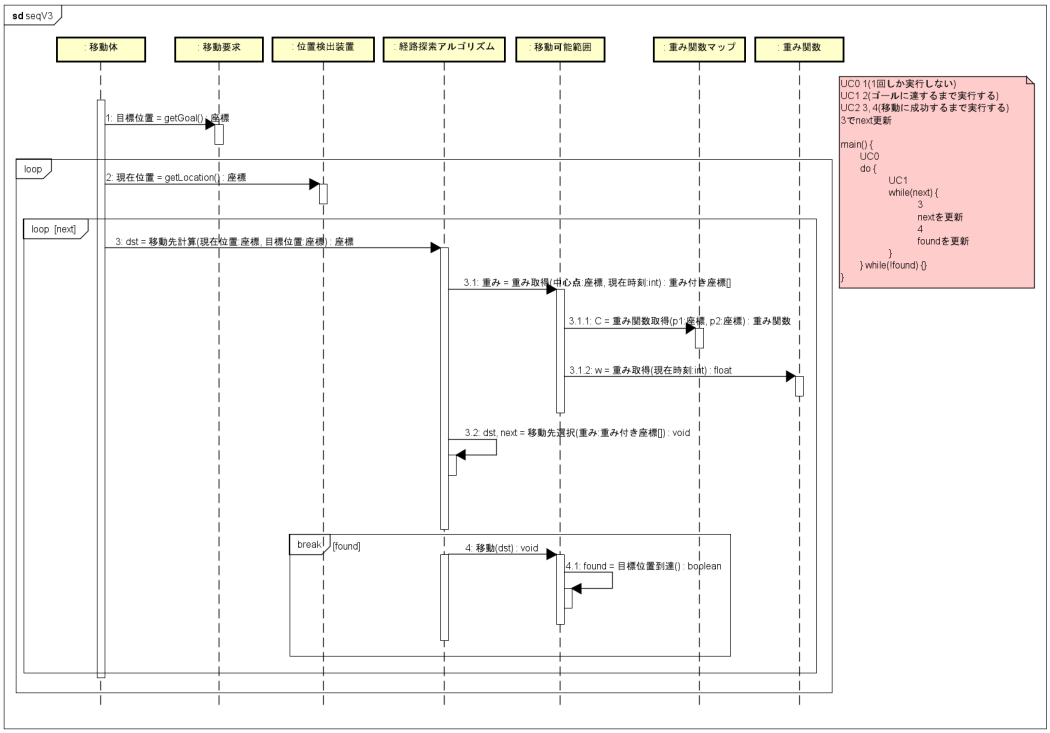


図 3.6: シーケンス図 V3

### 3.4.3 複雑度評価

シーケンス図 3.6 の複雑度計測をする。

**UC0**

1:目標位置 = getGoal():座標
　他オブジェクトとの交信のため 1

**UC1**

2:現在位置 = getLocation():座標
　他オブジェクトとの交信のため 1

**UC2**

3:dst, next = 移動先計算 (現在位置:座標, 目標位置:座標):座標
　他オブジェクトとの交信のため 1

3.1:重み = 重み取得 (中心点:座標):重み付き座標 []
　他オブジェクトとの交信のため 1

3.1.1:C = 重み関数取得 (p1:座標, p2:座標):重み関数
　他オブジェクトとの交信のため 1

3.1.2:W = 重み取得 (現在時刻:int):float
　他オブジェクトとの交信のため 1

3.2:dst, next = 移動先選択 (重み:重み付き座標 []):座標
　自オブジェクトとの交信のため 0

4. 移動 (dst):void
　他オブジェクトとの交信のため 1

4.1found = 目的地到達 ():boolean
　自オブジェクトとの交信のため 0

V1、V2、V3 の複雑度計測結果をまとめた表が表 3.3 である。V3 は V1 と比べ UC2 の複雑度が 5 倍になった。

表 3.3: V3 の複雑度指標の計測結果

| | V1 | V2 | V3 |
|---|---|---|---|
| UC0 | 1 | 1 | 1 |
| UC1 | 1 | 1 | 1 |
| UC2 | 1 | 3 | 5 |

### 3.4.4 複雑度評価の改善

これまでに用いた複雑度指標は引数が考慮されておらず、他のオブジェクトと引数の個数が多いメッセージ交信を行っても複雑度は 1 と数えられてしまうため、この指標は実態に即していない。バージョン 2 において重みを表す引数の個数が増えたことにより、正常値、不正値の判定を行い、例外処理をするため実行パスが増加した。よって、RFC に引数を考慮した指標を新たに定める。引数に関しては以下のように取り扱う。

- 引数がプリミティブ型及び `String` のものは除外
- 静的に決定できる値を返す物は除外
- 静的に決定できない値（例えば、その値を計算するメソッドを持ったオブジェクト）を返す場合は、戻り値となったオブジェクトの呼び出しに必要な通信の個数を数える

引数を考慮した RFC を用い、V1, V2, V3 の各シーケンス図の複雑度を計測した結果が表 3.4 である。V2 は V1 と比べると 7 倍に、V3 は 11 倍となった。

表 3.4: 改善された複雑度指標の計測結果

| | V1 | V2 | V3 |
|---|---|---|---|
| UC0 | 1 | 1 | 1 |
| UC1 | 1 | 1 | 1 |
| UC2 | 2 | 7 | 11 |

引数を考慮した RFC では V3 において著しい増加が見られるが、例外がある場合にこの指標はどうなるかを確認したい。

## 3.5　バージョン4

### 3.5.1　機能要求の変更

| | |
|---|---|
| 機能要求 | 経路探索に失敗した場合にも、パラメータを修正することにより、経路を再探索する機能を追加する |

### 3.5.2　設計の変更

クラス「経路探索アルゴリズム」に例外からの回復を行うメソッド「探索条件変更」を追加。このメソッドは、最初のスタート地点からの経路探索において例外が発生した場合に限り起動され、スタート地点を変更した上で、新たに経路探索を試みる。新しいスタート地点からの探索が成功した場合、終了する。

クラス図を図 3.7 に示す。

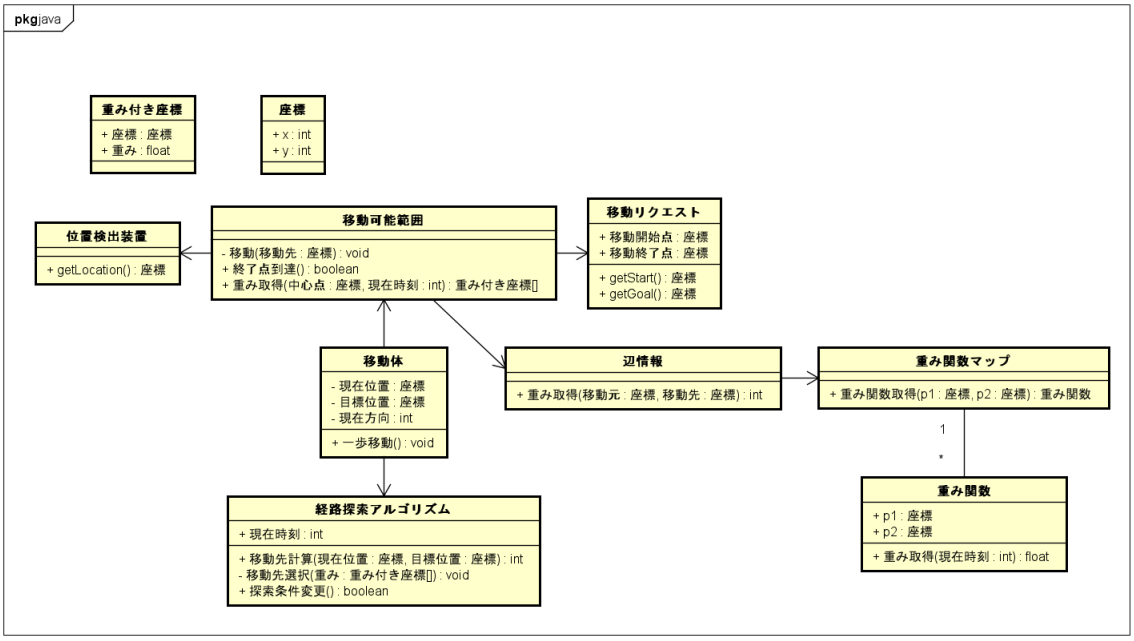


図 3.7: クラス図 V4

オプションの「SearchFailException」を追加する。これは、V3 の経路探索アルゴリズムにおいてメソッド「探索条件変更」が行われるため、シーケンス図 3.8 のようになる。

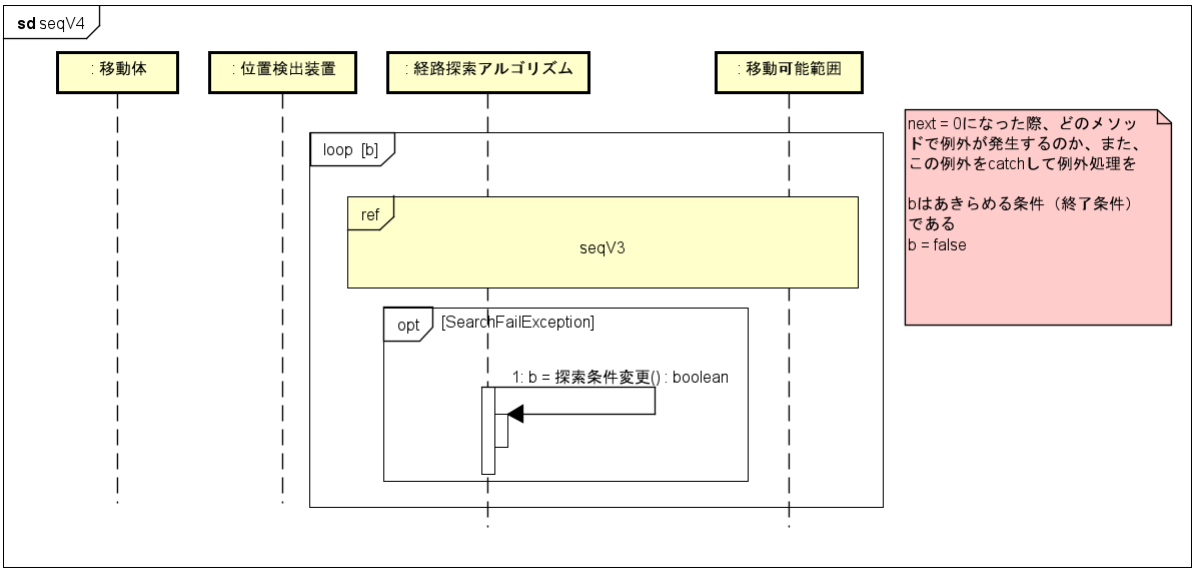


図 3.8: シーケンス図 V4

### 3.5.3　改善された評価指標の有効性確認

図 3.8 における複合フラグメント (opt) の例外処理「SearchFailException」は、以下に再掲したの図 3.9 のメッセージ 3.1, 3.2, 4 の 3 か所で発生する。

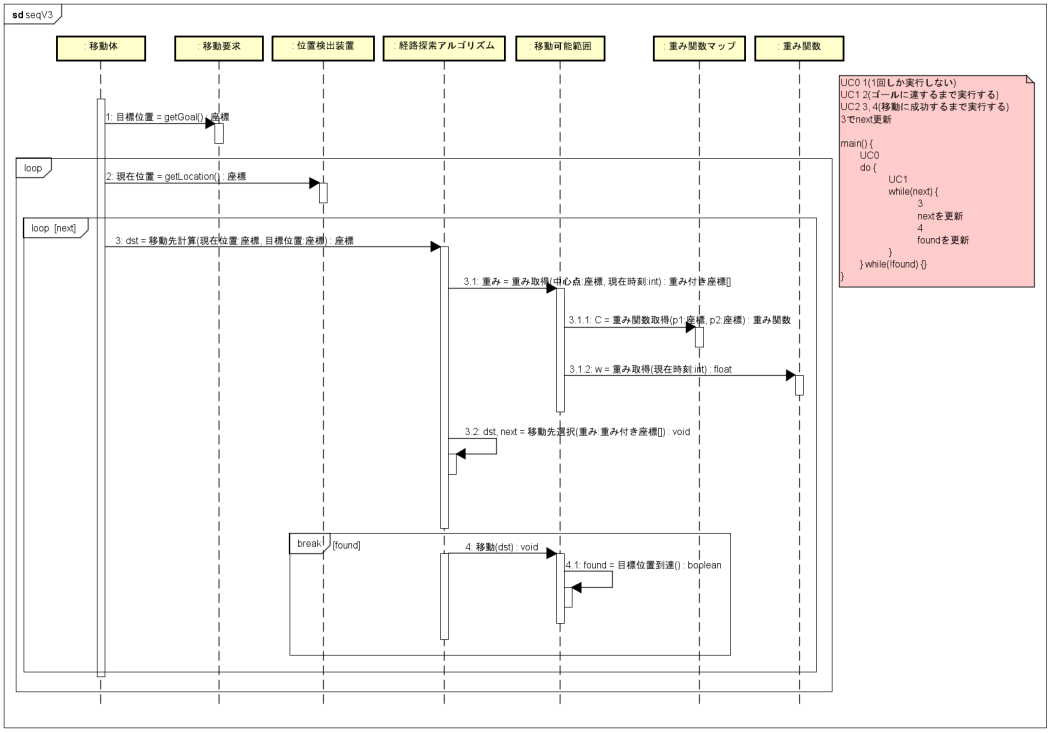


図 3.9: シーケンス図 V3(再掲)

例外発生時の実行パスは以下の 3 通りである。

1. メッセージ 3 → メッセージ 3.1(例外発生) → 例外処理
2. メッセージ 3 → メッセージ 3.1 → メッセージ 3.2(例外発生) → 例外処理
3. メッセージ 3 → メッセージ 3.1 → メッセージ 3.2 → メッセージ 4(例外発生) → 例外処理

それぞれの実行パスの複雑度計測をする。

**実行パス 1**

3:dst, next = 移動先計算 (現在位置:座標, 目標位置:座標):座標
　他オブジェクトとの交信のため 1、引数が 2 つあるため合計 3

3.1:重み = 重み取得 (中心点:座標):重み付き座標 []
　他オブジェクトとの交信のため 1、引数が 1 つあるため合計 2

ゆえに、実行パス 1 の UC2 における複雑度は合計 5 である。

**実行パス 2**

3.1.1:C = 重み関数取得 (p1:座標, p2:座標):重み関数
　他オブジェクトとの交信のため 1、引数が 2 つあるため合計 3

3.1.2:W = 重み取得 (現在時刻:int):float
　他オブジェクトとの交信のため 1、引数がプリミティブ型であるため合計 1

3.2:dst, next = 移動先選択 (重み:重み付き座標 []):座標
　自オブジェクトとの交信のため 0

ゆえに、実行パス 2 の UC2 における複雑度は実行パス 1 と合わせて 9 である。

**実行パス 3**

4:移動 (dst):void
　他オブジェクトとの交信のため 1、引数が 1 つあるため合計 2

4.1:found = 目的地到達 ():boolean
　自オブジェクトとの交信のため 0

ゆえに、実行パス 3 の UC2 における複雑度は実行パス 2 と合わせて 11 である。

したがって、例外発生時の UC2 の複雑度は実行パスそれぞれの複雑度を合計して、25 である。

シーケンス図 3.8 の複雑度を計測した結果が表 3.5 である。例外発生を伴う実行パスの増加により、複雑度が急増した。これは、計測方法に依存せず、問題が複雑になっているのが原因である。

表 3.5: 複雑度指標の計測結果 (V4 を含む)

| | V1 | V2 | V3 | V4 |
|---|---|---|---|---|
| UC0 | 1 | 1 | 1 | 1 |
| UC1 | 1 | 1 | 1 | 1 |
| UC2 | 2 | 7 | 11 | 25 |
| m3.1 | | | | 5 |
| m3.2 | | | | 9 |
| m4 | | | | 11 |

### 3.5.4 指標の妥当性確認

複雑度指標を変更して再度計測を行う。用いる複雑度指標を以下に示す。

**WMC(Weighted methods per class) を用いた計測**
計測対象のクラスが持つメソッドの複雑度の合計である。計測対象クラス C の各メソッドを $M_1, M_2, \cdots, M_n$ とし、これらのメソッドの複雑度をそれぞれ $c_1, c_2, \cdots, c_n$ とすると、$WMC(C) = \sum_{k=1}^{n} C_k$ である。本計測では、各メソッドの複雑さの値を 1 とし、各バージョンのシーケンス図に出現するメソッド数を数え上げる。ただし、自オブジェクトとメッセージ交信を行うメソッドの複雑さの値は 0 とする。

複雑度指標 WMC を用い、シーケンス図 3.8 の複雑度を計測した結果が表 3.6 である。V4 の複雑度が V3 と比較して複雑度が 6 増加した。

表 3.6: WMC の計測結果

| | V1 | V2 | V3 | V4 |
|---|---|---|---|---|
| UC0 | 1 | 1 | 1 | 1 |
| UC1 | 1 | 1 | 1 | 1 |
| UC2 | 1 | 3 | 5 | 11 |
| m3.1 | | | | 2 |
| m3.2 | | | | 4 |
| m4 | | | | 5 |

### 引数とその構造体を考慮した WMC による計測

WMC に対し、各メソッドの引数の個数を数え上げる。ここでは引数が構造体である場合、その中身の値を考慮する。つまり 1 つの構造体の中身の値が 2 つある場合、複雑さを 2 とする。

シーケンス図 3.8 の複雑度を計測した結果が表 3.7 である。V4 の複雑度が V3 と比較して複雑度が 25 増加した。例外発生に伴う実行パスにおいて、引数を持つメッセージが多いことが急増の要因である。

表 3.7: 引数とその構造体を考慮した WMC の計測結果

| | V1 | V2 | V3 | V4 |
|---|---|---|---|---|
| UC0 | 1 | 1 | 1 | 1 |
| UC1 | 1 | 1 | 1 | 1 |
| UC2 | 2 | 10 | 18 | 43 |
| m3.1 | | | | 9 |
| m3.2 | | | | 16 |
| m4 | | | | 18 |

表 3.7 の計測結果から、V3 のシーケンス図 3.10 の m3.1、m3.1.1、m3.1.2 が複雑度急増の要因となっていることがわかる。

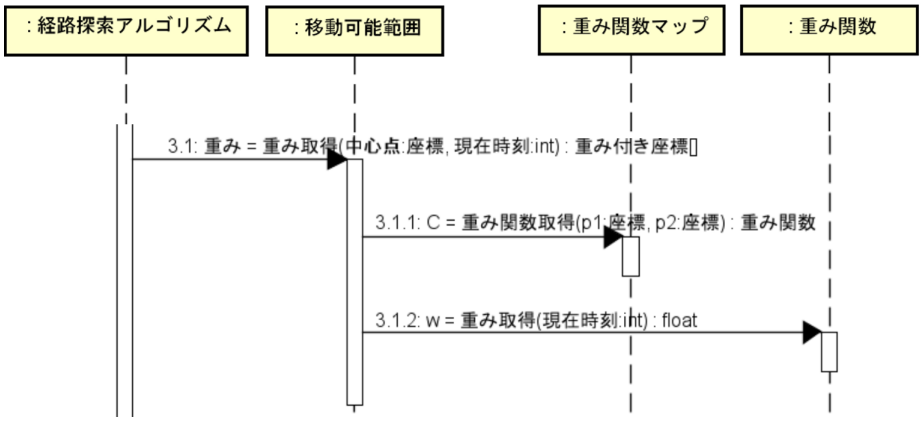


図 3.10: V3 のメッセージ

複雑度指標によって、複雑度が増加している要因が明らかになったので、次章でこれを軽減するための手法を提案する。

# 第4章　複雑度抑制の手法

## 4.1　原因分析と方針

第3章では各バージョンのシーケンス図に対して複雑度指標の計測を行った。この結果、例外発生に伴う実行パスの増加により複雑度が急増することが明らかになった。本章では、この実行パスに含まれるメッセージ数を削減するための手法を適用し、複雑度指標の計測をする。また、適用前後でどの程度複雑度を抑制されたかを示す。

## 4.2　抑制の対象と手法

複雑度急増の要因となっているメッセージ交信に関わるオブジェクトをグループ化し、その間のメッセージ交信だけを抽出したシーケンス図を生成する。複雑度急増の要因となる図4.1の赤破線で囲った箇所をグループ化し、1つのオブジェクトとして扱うことでシーケンス図におけるオブジェクト数を減らすことができる。

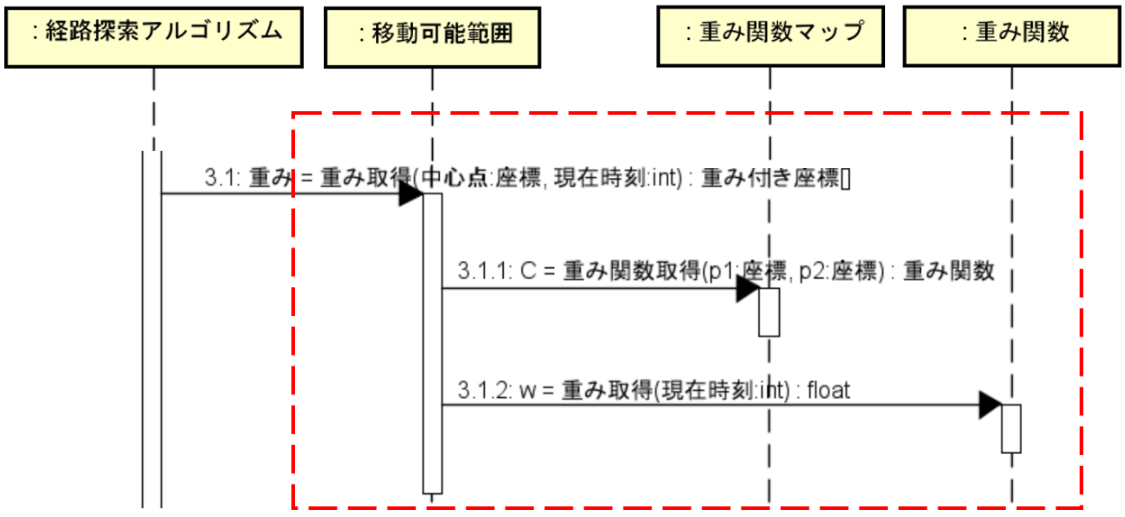


図 4.1: 適用前のメッセージ

オブジェクトをグループ化する手法を以下のように定義する。

- 1 つのグループのうち、代表となるオブジェクトを決める
- グループ外部とメッセージ交信をするオブジェクトを代表とする
- グループ自体を 1 つのオブジェクトとして扱い、階層化を可能とする
- 代表となるオブジェクトに対するメソッド呼び出しは変更しない

「移動可能範囲」オブジェクトを代表とし、「重み関数マップ」と「重み関数」を含めた 3 つをグループ化する。これによりオブジェクトの数が減り、メッセージ 3.1.1 およびメッセージ 3.1.2 が計測対象のシーケンス図から削除され、メッセージ交信の数が減少する。

この手法を図 4.1 適用した結果が、図 4.2 である。

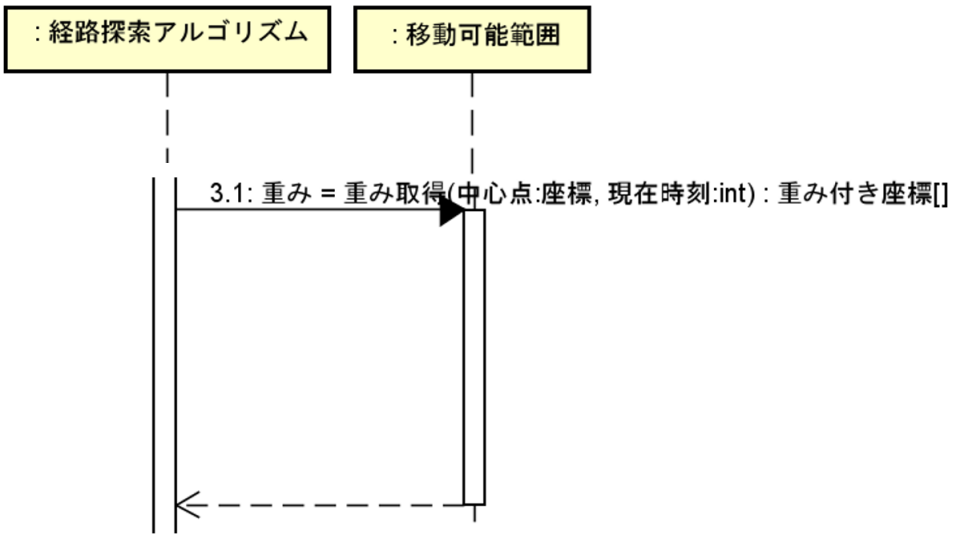


図 4.2: 適用後のメッセージ

## 4.3 抑制結果

第 4 章で用いた複雑度指標のうち、計測結果が最も増加する「引数とその構造体を考慮した WMC」により計測する。

変更を加えた V4 のシーケンス図の複雑度を計測した結果を表 4.1 に示す。

表 4.1: 複雑度指標の計測結果の比較

| | 適用前 | 適用後 |
|---|---|---|
| UC0 | 1 | 1 |
| UC1 | 1 | 1 |
| UC2 | 43 | 29 |
| m3.1 | 9 | 9 |
| m3.2 | 16 | 9 |
| m4 | 18 | 11 |

この結果から、適用により複雑度指標が 14 減少した。したがって、この手法はシーケンス図における複雑度の抑制に有効である。

# 第5章　おわりに

## 5.1　まとめ

本研究では MDD における機能追加に対し、多数の例外シーケンスを考慮するべきソフトウェアの複雑度を抑制する手法を提案した。MDD は UML のモデルからソースコードのひな型を自動的に生成する手法であり、モデル開発におけるモデルとソースのコードの一貫性保持が容易であるが、UML 自体が複雑になり保守性が悪化し、ソースコードにも悪影響を及ぼすという問題がある。

CK メトリクスの RFC と WMC を用いてシーケンス図の複雑度を測定し、機能追加による複雑度の増加要因が例外発生を伴う実行パスの増加であることが明らかになった。

実行パスに含まれるメッセージ数を削減し、複雑度を抑制する手段としてオブジェクトをグループ化するという手段をシーケンス図に適用したところ複雑度を抑制することができた。以上のことから、例外シーケンスを考慮したシーケンス図の複雑度を抑制することで MDD におけるモデルの保守性を保つことが可能となり、ソースコードの生成に貢献できることが期待できる。

## 5.2　今後の課題

1 つのモデルケースに対して複雑度を測定し抑制手段が有効であったが、これ以外のモデルに対しても有効な手段であるかを検証する必要がある。

# 謝辞

本研究を進めるにあたり、主指導教員である鈴木正人准教授には親切丁寧なご指導を賜りましたことに深く感謝いたします。私の力量が足りず研究活動が難航していた時期もありましたが、最後まで諦めずに本論文を書き上げることができたのは鈴木先生が研究の議論やアドバイスをして下さったからです。本当にありがとうございました。

# 参考文献


[1] 青田　健太郎．ソフトウェア品質を向上させるためのメトリクス， Winter Workshop 2016 in Zushi (2016)

[2] 津田直彦, 鷲崎弘宜, 深澤良彰．上流工程の UML クラス図を入力としたソフトウェアの保守性測定メトリクススイート，情報処理学会第 75 回全国大会 (2013)

[3] 米山　哲平. モデル駆動開発における適応保守の際の複雑度を軽減する複数のメトリクスに基づく開発手法に関する研究, 修士論文, 北陸先端科学技術大学院大学 (2016)

[4] Shyam R. Chidamber, Chris F. Kemerer．A Metrics Suite for Object Oriented Design，IEEE Transactions Vol.20 No.6 (1994)

[5] David S. Frankel．MDA モデル駆動アーキテクチャ，株式会社エスアイビー・アクセス (2003)

[6] David S. Frankel, John Parodi．MDA マニフェスト—巨匠たちの論点：エッセンス、及び現状と未来，株式会社エスアイビー・アクセス (2005)

[7] Object Management Group．UML2.0 仕様書 2.1 対応，オーム社 (2006)

[8] テクノロジックアート．UML2 ハンドブック，翔泳社 (2004)

[9] Lawrence Putnam, Ware Myers．初めて学ぶソフトウェアメトリクス，日経 BP 社 (2005)

[10] Mike Cohn．アジャイルな見積もりと計画づくり—価値あるソフトウェアを育てる概念と技法，マイナビ (2009)

[11] Hassan Gomaa． Designing Concurrent, Distributed, and Real-Time Applications with UML，Addison-Wesley Professional Education (2000)

[12] 佐藤 義男.PMBOK 第 4 版による IT プロジェクトマネジメント実践法，日本印刷株式会社 (2009)