Vol.2016-SE-192 No.13
Vol.2016-EMB-41 No.13
2016/6/3

設計モデルのリファクタリング手法の適用領域分割による組み合わせ

榊原 由季[1]　満田 成紀[2]　福安 直樹[2]　松延 拓生[2]　鯵坂 恒夫[2]

概要：ソフトウェアの品質を向上する技術の１つにリファクタリングがある。汎用的なプログラミング言語を対象としたリファクタリング手法は数多く提案されているが，上流の開発フェーズに着目した手法の提案はまだ少ない。本研究の目的は，設計レベルの仕様書を対象として，より適用範囲の広いリファクタリング手法を考案することである。そこで，クラス図からデザインパターンを用いたリファクタリングが可能な箇所を見つける手法と，シーケンス図のメトリクスからリファクタリングすべき箇所を見つける手法とを組み合わせることで，複数の観点に対応した設計レベルのリファクタリング手法を提案する。ソフトウェアシステム全体の中でそれぞれの手法が有効に機能する範囲をとらえ，対象領域を分けて適用し，適用後に結果を組み合わせる。本手法を実際の開発事例に適用したところ，適用可能性がドメインに依存し，組込みシステムには適用しやすいと考察できた．

Combination of Model Refactoring Methods by Dividing Their Target Area

Yuki Sakakibara[1]　Naruki Mitsuda[2]　Naoki Fukuyasu[2]　Takuo Matsunobe[2]　Tsuneo Ajisaka[2]

Abstract: Refactoring is a method for improving software quality. There are many methods to refactor program code. But a few methods to refactor software model have been proposed. In this paper, we propose a method combination to refactor UML model. That is a method which considers two aspects, structure and behavior, by combining design pattern based class diagram refactoring and sequence diagram refactoring. Firstly, we recognize class sets which each method is applied. Secondly, we divide the class sets and apply each method for the classes. Finally, we combine the results. Through a case study, we have considered that applicability of our method depends on the application domain of the target software, especially, embedded system is suitable for our method.

1. はじめに

ソフトウェア品質とは，ソフトウェアが固有の能力として備えている性質や特性のことである．ソフトウェアの良し悪しを判断する材料となるもので，一般的にその性質や特性が明示的，または暗黙的な期待水準に達していれば「品質がよい」，期待水準を超えていれば「品質が高い」と評価される．ソフトウェアの品質にはさまざまな側面がある．ISO/IEC 25010 では，システム／ソフトウェア製品品質モデルとして，利用時のソフトウェア品質，外部ソフトウェア品質，内部ソフトウェア品質の品質特性を規定している．

ソフトウェアの品質を向上させる技術の 1 つにリファクタリングがある．リファクタリングとは，ソフトウェアの外部的振る舞いは保ったまま，その内部構造を改善していく作業である [1]．リファクタリングはソフトウェアの保守性や拡張性といった品質を高めるだけでなく，第三者がソフトウェアのコードを確認する際に，より理解しやすくするために行われる．

[1] 和歌山大学大学院システム工学研究科
Sakaedani930, Wakayama 640–8510, Japan
[2] 和歌山大学システム工学部
Sakaedani930, Wakayama 640–8510, Japan

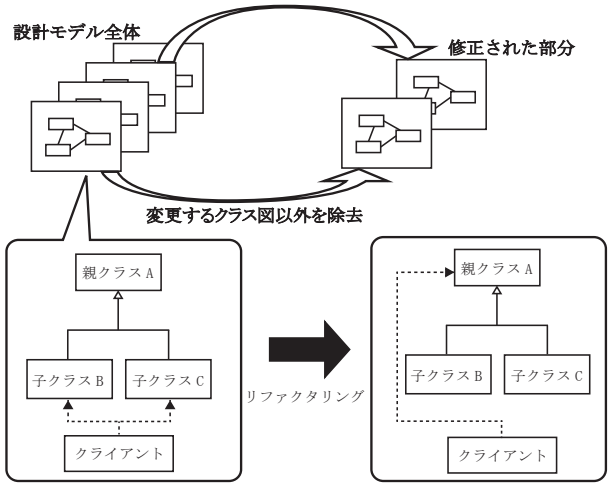

図 1 パターンベースリファクタリング手法の概要 [4]

本来リファクタリングはソースコードに対して行われるものであったが，モデル中心の開発手法が発展したことで，モデルベースのリファクタリングが新たに定義されている．複雑なクラス間の関連を簡潔にすることにより，データの流れが開発者にとって理解しやすくなる．

ソースコードベースのリファクタリングに対して，モデルベースのリファクタリングは設計段階で実行できるという利点がある．ソースコードを記述する前の段階で修正がしやすく，可読性の高いモデルを作成することで開発の手戻りを少なくすることができる．また，リファクタリングの流れを UML モデルで確認できるため，開発者にとってリファクタリング箇所の確認が容易となる [2][3].

2. 設計モデルのリファクタリング

2.1 パターンベースのリファクタリング

文献 [4] では，パターンベースのリファクタリング手法が提案されている．このリファクタリング手法の概要を図 1 に示す．UML モデルを解析し，デザインパターンが適用できる箇所を検出する．UML を解析した結果，特定の構造を持つ場合はデザインパターンを適用することができる．デザインパターン毎に適用条件を決めておき，条件に合った UML モデルを修正する．この文献では Observer パターンを対象にツールを実装している．デザインパターンの適用条件を満たす UML モデルを修正する．デザインパターンが適用可能である箇所を全体の UML モデルから抽出する．これによって，開発者は小規模な UML モデルからデザインパターン適用可能箇所を確認することができる．

2.2 シーケンスベースのリファクタリング

文献 [5] では，シーケンスベースのリファクタリング手法が提案されている．この手法は，次のことを前提としている．

(1) 相互作用フラグメントとは，あるライフラインが送信するメッセージ，あるいは関わる複合フラグメント，相互作用の利用のことを指す．
(2) 委譲操作とは，メッセージの送信を 1 度しか行わず，送信先が他のオブジェクトである操作と定義する．
(3) ゲッタは外部から見たシステムの振る舞いに影響を与えない．

表 1 Bad Smell とリファクタリング手法の対応表 [5]

Bad Smell	リファクタリング手法
長すぎる操作	相互作用の抽出 操作の抽出と相互作用の抽出
メッセージの連鎖	委譲の隠蔽
重複した相互作用フラグメント	相互作用の抽出 操作の抽出と相互作用の抽出
仲介人	仲介人の削除

この手法では，Fowler らがソースコードに対して提案した Bad Smell をシーケンス図に適用し，シーケンス図中の Bad Smell を自動検出する．それを取り除くためにモデルベースリファクタリングを行う．この文献では，自動検出して効果が期待できるものとして次の 4 種類の Bad Smell を対象にアルゴリズムが定義されている．

(1) 長すぎる操作
(2) メッセージの連鎖
(3) 重複した相互作用フラグメント
(4) 仲介人

表 1 に，文献で提案されている Bad Smell と，それを取り除くためのリファクタリング手法の対応関係を示す．

3. リファクタリング手法の組み合わせ

2 章で紹介したシーケンスベースとパターンベースの 2 つのリファクタリング手法を組み合わせる方法を考えた．初めに，文献 [5] においてシーケンスベース手法を適用している飛行船制御システムを対象にして考えた．

3.1 シーケンス図中の Bad Smell

飛行船制御システムのシーケンス図では，次のような Bad Smell が検出されている [5].

(1) 長すぎるメソッド
Blimp クラスの操作 startautomaticFight がメッセージを 11 回送信している．

(2) メッセージの連鎖
BlimpSystem クラスの操作 setUSSensors と setFlightPath が他のオブジェクトからオブジェクトを 1 つだけ取得している．

(3) 重複した相互作用フラグメント
相互作用 startTestFlight と Fligh 中の，操作 getSensorInfo，setSensorInfo のメッセージ送信が重複している．

(4) 仲介人

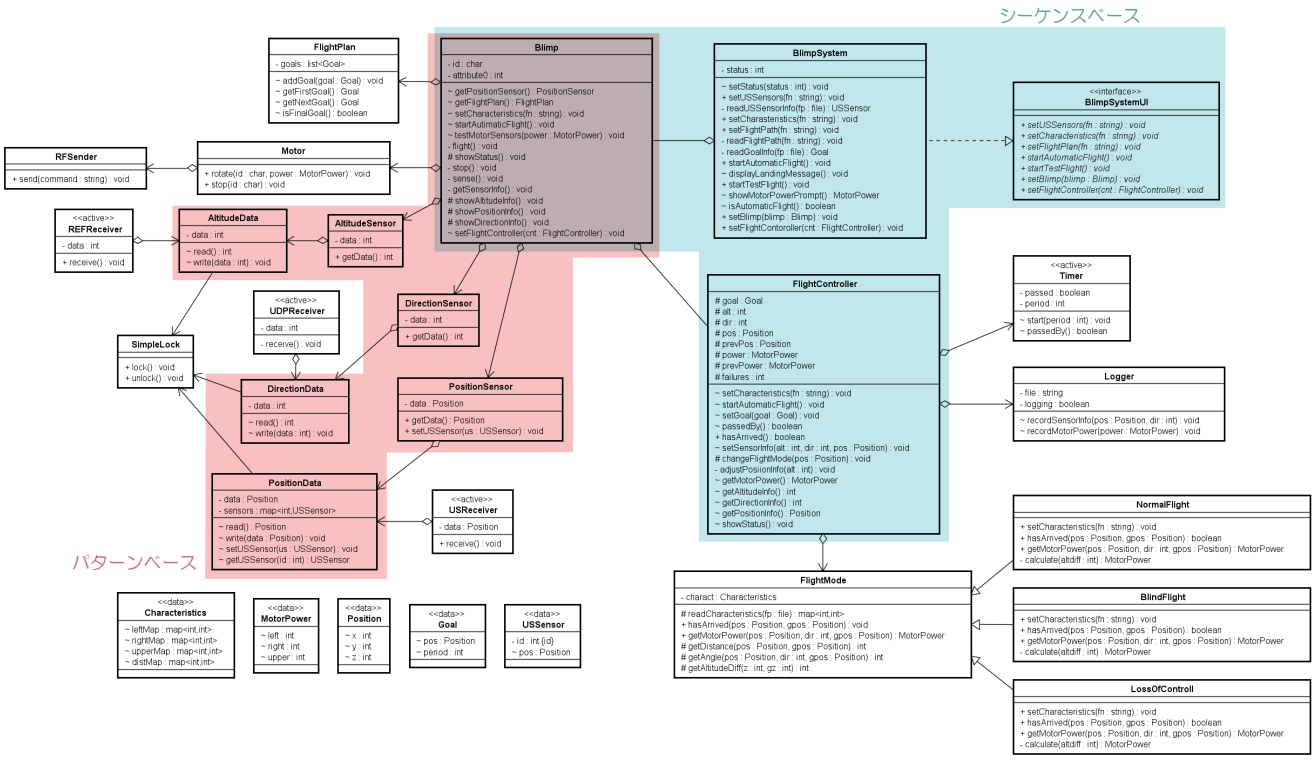

図 2 適用対象領域 [飛行船制御システム]

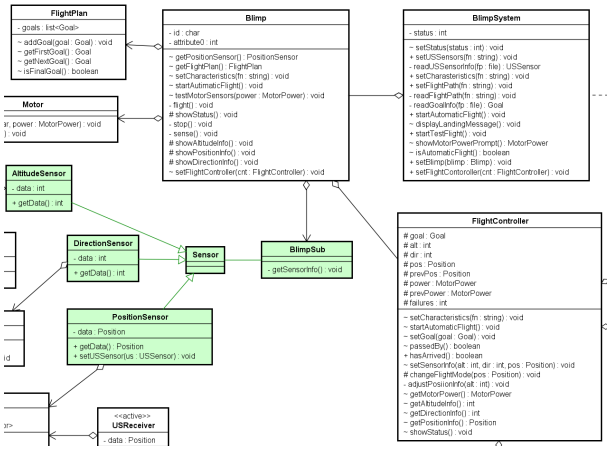

図 3 修正後のクラス図

Blimp クラスと FlightController クラスの操作数は 13 であり，それぞれ委譲操作を 5 つ持っている．

3.2 パターンベース手法の適用

同じ飛行船制御システムに対して，パターンベース手法を適用した．文献 [4] では Observer パターンを対象としていたため，このシステムのクラス図において，Observer パターンの適用可能箇所を調査した．Observer パターンとは，あるオブジェクトが状態を変えたとき，それに依存する全てのオブジェクトに自動的に知らされ，さらに，それらが更新されるようオブジェクト間に一対多の依存関係を定義するパターンである?．このシステムにおいて，状態が変化し，その際に変化が通知される部分はセンサーデータである．よって，センサーデータに関連したクラスがパターンベース手法の適用対象になると考えられる．これに該当するのは，図 2 に示す，Blimp クラス，AltitudeSensor クラス，DirectionSensor クラス，PositionSensor クラス，AltitudeData クラス，DirectionData クラス，PositionData クラスである．また，AltitudeSensor，DirectionSensor，PositionSensor の 3 つの Sensor クラスの役割が似ている．よって，これらのクラスを抽象化し，1 つのクラスにまとめることができる．

シーケンスベース手法で検出された箇所と，パターンベース手法が適用可能である箇所をクラス図上で確認すると図 2 のようになった．Blimp クラス，BlimpSystem クラス，FlightController クラスがシーケンスベース手法の適用対象領域である．それに対し，Blimp クラス，AltitudeSensor クラス，DirectionSensor クラス，PositionSensor クラス，AltitudeData クラス，DirectionData クラス，PositionData クラスがパターンベース手法の適用対象領域である．Blimp クラスは 2 つの手法の領域に含まれているが，その他のクラスは領域が分かれている．Blimp クラスを 2 つに分けると，それぞれの手法が適用対象とする領域を分割することができる．分割した領域にそれぞれの手法を適用し，その結果を組み合わせるということを考えた．

3.3 適用対象領域の分割

シーケンスベース手法とパターンベース手法の適用対象

領域を分割するため，2 つの領域が重なっている Blimp クラスを分断した．新しく BlimpSub クラスを作り，Blimp クラスのメソッドのうち Sensor クラスと関連のあるメソッドを抽出し，BlimpSub クラスに持たせた．この結果，Blimp クラスはシーケンスベース手法の，BlimpSub クラスはパターンベース手法の対象領域となり，2 つの手法の適用対象領域を分けることができた．

Observer パターンの適用条件となるモデルは，ConcreteObserver クラスが Subject クラスを利用しており，さらに Subject クラスも ConcreteObserveer クラスを利用しているようなモデルである．つまり，Subject クラスと ConcreteObserver クラスが相互に関連している場合に Observer パターンが適用できる．そこで，飛行船制御システムにおけるパターンベース手法の適用対象領域を，Observer パターン適用条件のモデルの形になるよう修正した．修正したクラス図が図 3 である．BlimpSub クラスは ConcreteObserver クラスに，AltitudeSensor クラス，DirectionSensor クラス，PositionSensor クラスは ConcreteSubject クラスに対応している．また，AltitudeSensor クラス，DirectionSensor クラス，PositionSensor クラスを抽象化した Sensor クラスを新たに作った．これは Subject クラスに対応している．Blimp クラスを分断し，適用対象領域を分割したことにより，前処理を行う際に変更を加えなければならない箇所が減少している．

3.4 適用領域分割による組み合わせ

図 4 は手法概要である．システムのシーケンス図にシーケンスベース手法を，UML モデルにパターンベース手法を適用し，それぞれの手法が適用可能である箇所を検出する．クラス図上で，それらのリファクタリングの適用対象領域を確認する．2 つの手法の適用対象領域が重なるクラスがあればそのクラスを分断し，手法ごとに適用対象領域を分ける．適用対象領域をそれぞれの手法でリファクタリングし，その結果を組み合わせる．

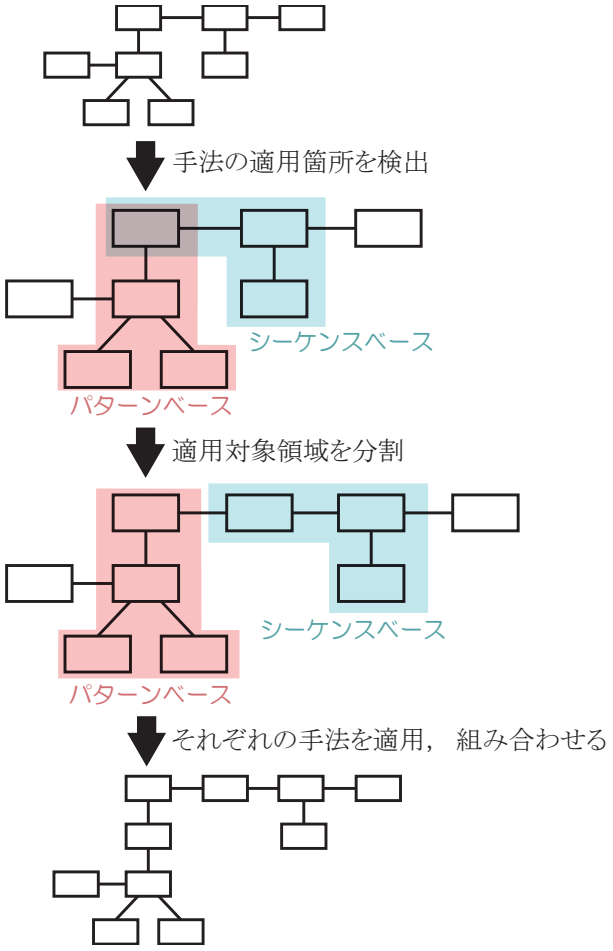

図 4 手法概要

4. 異なるドメインへの適用

3 章と同様に，教務支援システムを対象に 2 つの手法を適用した．適用対象とした教務支援ステムはシラバス，時間割管理や履修登録などの機能を持っている．

4.1 パターンベース

文献 [4] において，パターンベース手法の適用対象とされていたのがこの教務支援システムであったため，その結果を参考にした．図 6 より，KyouikuKatei クラス，Kamoku クラス間，JugyouKamoku クラス，Gakusei クラス間，SeisekiHanteiYouken クラス，KamokuKubun クラス間の 3 箇所が相互関係を持っている．よって，Observer パターン適用条件に当てはまるためパターンベース手法の適用対象となる．

4.2 シーケンスベース

本来は設計モデルのシーケンス図を対象に Bad Smell を検出するが，このシステムのシーケンス図が存在しなかったため，ソースコードからメトリクス測定ツール [*1] を用いて Bad Smell を検出した．その結果，SyllabusServiceImpl クラス内の exportCSVRow メソッドのサイクロマチック複雑度が 15 であった．それに対し，システム全体のサイクロマチック複雑度の平均は 1.426 であり，exportCSVRow メソッドのサイクロマチック複雑度が高いことがわかる．そこで，exportCSVRow メソッドのソースコードを確認し，シーケンス図を作成した．図 5 が作成したシーケンス図である．

このシーケンス図からは，次のような Bad Smell が検出される．

(1) 長すぎるメソッド

SyllabusServiceImpl クラスの ExportCSVRow メソッドから他のオブジェクトに送信されているメッセージ数は 63 である．これは長すぎるメソッドである．

(2) メッセージの連鎖

[*1] Metrics 1.3.6, `http://metrics.sourceforge.net`

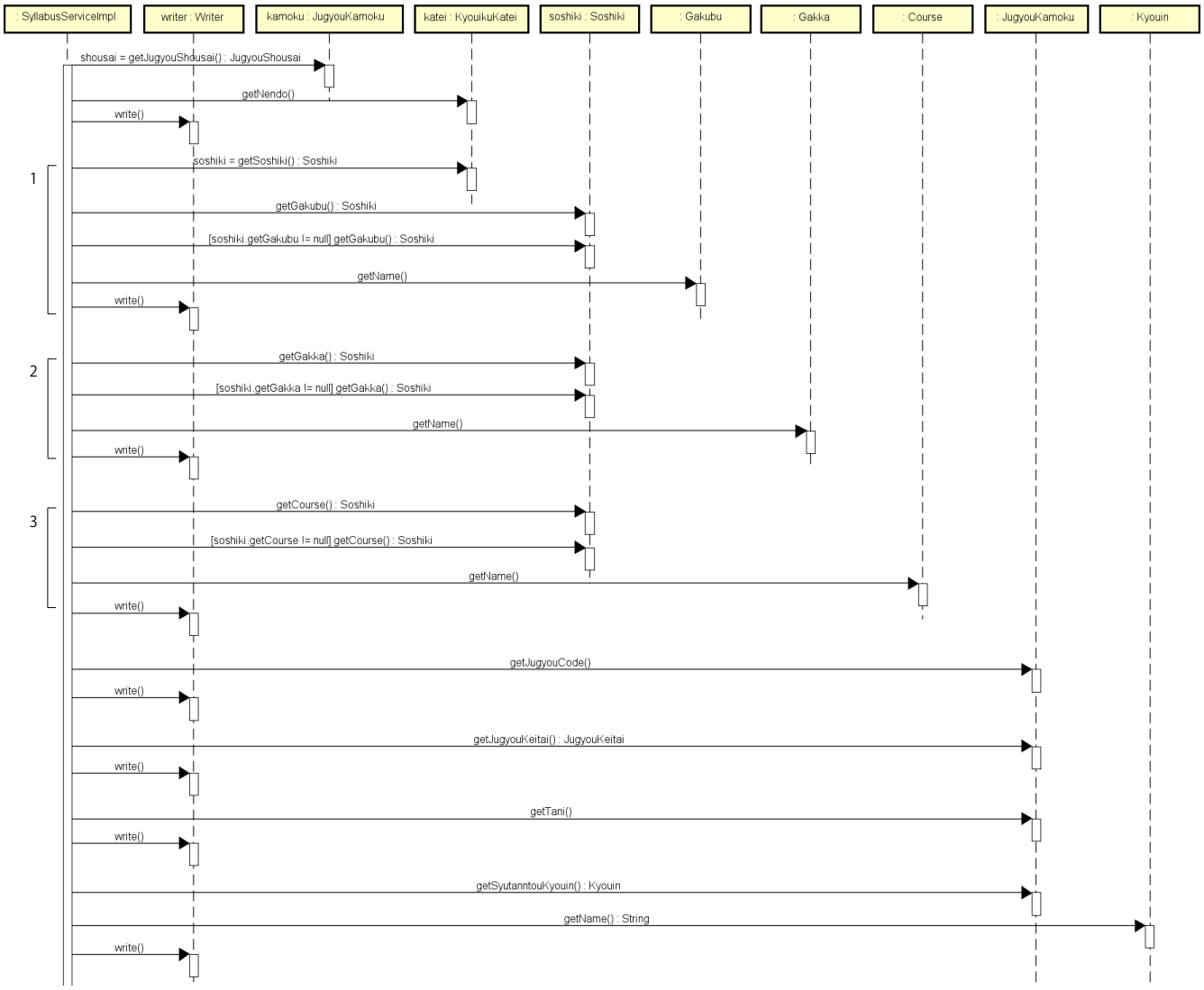

図 5 exportCSVRow メソッドのシーケンス図

図 5 で，SyllabusServiceImpl クラスが KyouikuKatei クラスの katei インスタンスに getSoshiki メッセージを送信し，Soshiki クラスの soshiki インスタンスを取得している．さらに，取得したインスタンスに getGakubu メッセージを送信している．KyouikuKatei クラスは SyllabusServiceImpl クラスにインスタンスを渡すという処理しか行っていない．これは，他のオブジェクトからオブジェクトを 1 つだけ取得しているシーケンスのため，メッセージの連鎖である．同様に，メッセージの連鎖と見られるシーケンスが 13 箇所ある．

(3) 重複した相互作用フラグメント

図 5 中で, exportCSVRow メソッドが JugyouKamoku のインスタンスに getSyutanntouKyouin メッセージを送信し，Kyouin クラスを取得している．取得したクラスに getName メッセージを送信し，Write クラスに write メッセージを送信してその結果を出力している．メッセージは異なるが，同じようなシーケンスとなっている部分が 8 箇所ある．また，図 5 に示す 1〜3 のシーケンスは，取得したインスタンスの存在確認をしていることを除くと同じようなシーケンスとなっている．これらの部分が重複した相互作用フラグメントである．

(4) 仲介人

このシーケンス図中には仲介人と考えられるシーケンスは存在しなかった．

4.3 対象領域

シーケンスベース手法で Bad Smell が検出される箇所と，パターンベース手法で検出された箇所をクラス図上で確認すると図 6 のようになった．Soshiki クラス, KyouikuKatei クラス，Kamoku クラス，Kyouin クラス，JugyouShousai クラス，JugyouKamoku クラス，Kyoushitu クラス，Rishu クラスがシーケンスベース手法の適用対象領域である．それに対し，KyouikuKatei クラス，Kamoku クラス， JugyouKamoku クラス，Gakusei クラス，SeisekiHanteiYouken

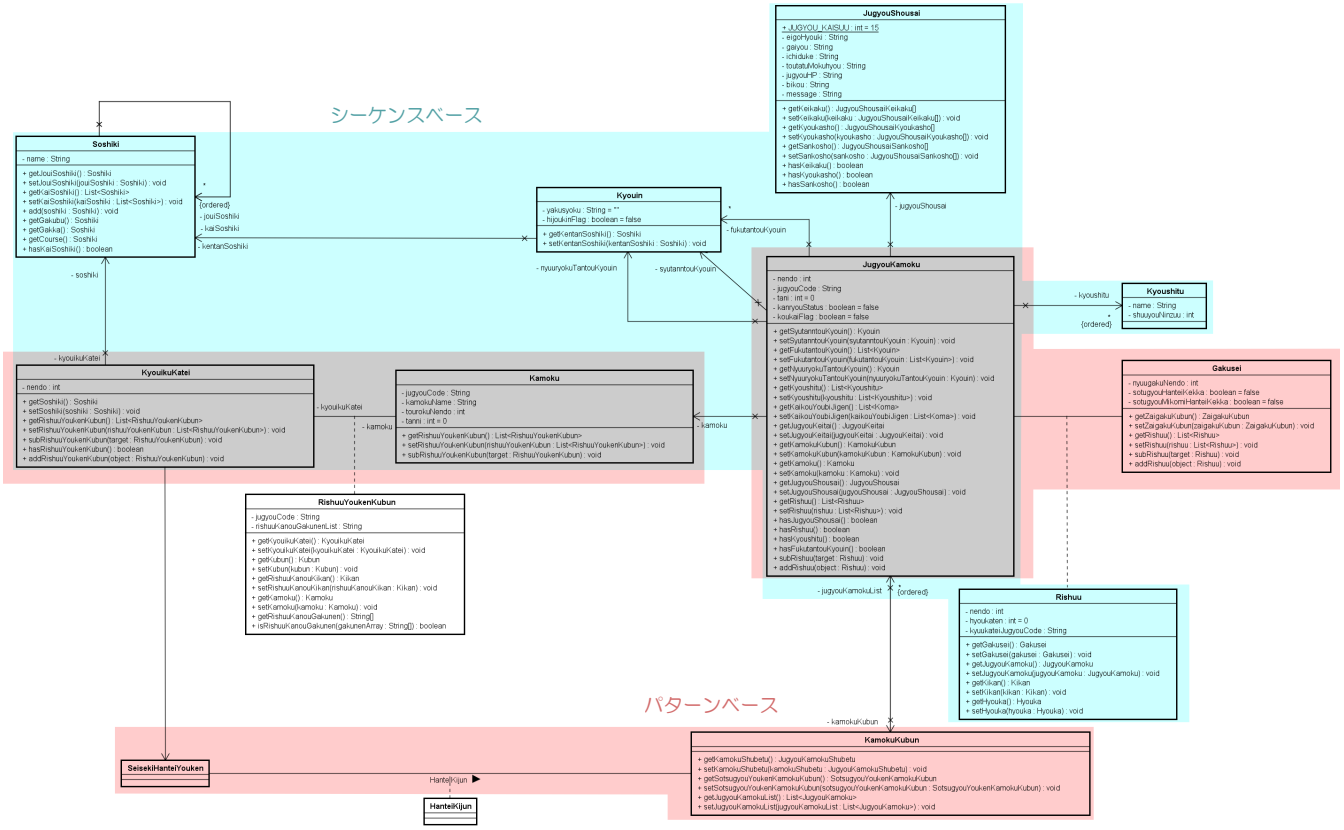

図 6 適用対象領域 [教務支援システム]

クラス，KamokuKubun クラスがパターンベース手法の適用対象領域である．

このシステムでは KyouikuKatei クラス，Kamoku クラス，JugyouKamoku クラスの 3 つのクラスが 2 つの手法の対象領域に含まれている．また，KyouikuKatei クラスと SeisekiHanteiYouken クラス，Kamoku クラスと JugyouKamoku クラス，JugyouKamoku クラスと KamokuKubun クラスがそれぞれ相互に依存関係を持っている．対象領域の重なるクラスが複数あり，さらにそれらのクラスが相互に依存関係を持っているため，3 章のように 2 つの手法の適用対象領域を分割することは困難である．

5. 考察

5.1 ドメインによる適用可能性

飛行船制御システムを対象に提案手法を適用した場合，シーケンスベース手法とパターンベース手法の適用対象領域をうまく分割できた．しかし，教務支援システムを対象に適用した場合，2 つの適用対象領域が重なるクラスが複数あり，それらのクラスが相互に依存関係を持っていたため，領域の分割は困難であった．

飛行船制御システムは組込みシステムに分類される．一般的に組込みシステムは，システム全体を制御する部分と，センサーやアクチュエーターを制御するドライバー部分に分けることができる．一方，教務支援システムは情報システムに分類される．一般的に情報システムはデータ構造を記述しているだけであり，システムを分けて考えることが難しい．このようなことが適用対象領域の分割のしやすさに関わっていると推測できる．

よって，提案手法の適用可能性はドメイン特性に依存していると考えられる．情報システムであっても，より詳細にドメインを分類すれば，分割しやすい特性を持つドメインが見つかるかもしれない．今回は 2 つのシステムにしか適用していないため，他のシステムにも適用して検証する必要がある．

5.2 複数クラスの分割

教務支援システムに提案手法を適用したところ，適用対象領域の重なるクラスが複数あった．さらに，それらのクラス間で相互に依存関係があったため領域を分割することは困難であった．

相互に依存関係にある複数クラスを分断する例を図 7 に示す．図 7 中のクラス B とクラス C が 2 つの手法の適用対象領域に重なるクラスである．これらのクラスをそれぞれ分断すると，一般的には図 7 中の 1 に示すような形になると思われる．この場合，分断したクラス間が相互に依存関係を持ってしまっており，クラス構造が複雑化する．このように分断してしまうと，元の構造よりも複雑なクラス構造になるためリファクタリングにならない．しかし，図

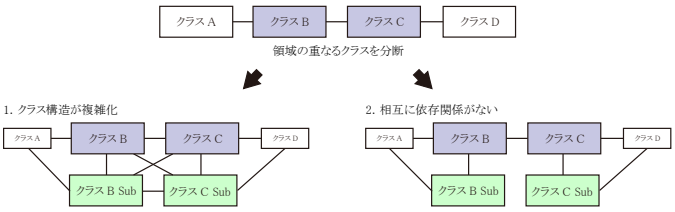

図 7 複数クラスの分断

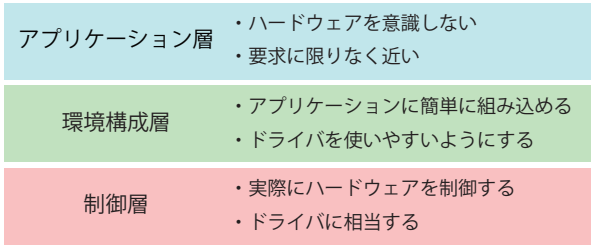

図 8 組込みシステム向けアーキテクチャの 3 層構造 [6]

中の 2 に示すような形に分断することができれば，適用対象領域を分割することができる．この場合，分断したクラス間に相互の依存関係がなく，クラス構造が複雑化しない．このような分断となれば，2 つの手法の適用対象領域を分割してリファクタリングを行うことは有効と言える．

図 7 中の 2 に示すようなクラスの分断ができるように，あらかじめメソッドを作り変えておくことが考えられる．この際，分割されたクラス間で，仲介人の Bad Smell となるシーケンスが現れることが予想される．しかし，領域を分割してしまうためシーケンスベースの手法では検出できないという問題がある．

5.3 組込みシステム向けアーキテクチャとの関係

リファクタリング手法ごとに適切な適用領域に分割する際には，対象とするシステムのアーキテクチャに注目することで，分割する部分が明確になることが考えられる．文献 [6] では，組込みシステム向けアーキテクチャが提案されている．このアーキテクチャは図 8 に示すように，アプリケーション層，環境構成層，制御層の 3 層構造となっている．最上層にユーザ視点の要求であるアプリケーション機能の実現をするアプリケーション層，最下層に実際にハードウェアを制御する制御層，中間層にアプリケーション機能からドライバを簡便に利用できるようにする環境構成層がある．

このアーキテクチャを飛行船制御システムに取り入れると，アプリケーション層にシーケンスベース手法の対象領域に含まれるクラスが集まる．一方，パターンベース手法の対象領域に含まれるクラスは環境構成層と制御層に集まる．適用領域分割のために分断された Blimp クラスと BlimpSub クラスに注目すると，それぞれアプリケーション層と環境構成層とに分かれることになる．したがって，本研究で行った適用領域分割は，アーキテクチャに適合させる行為とみなせることになる．

このことから，利用されている様々なシステムアーキテクチャにを対象に，リファクタリング手法の適用領域との対応関係を整理しておくことが考えられる．システム開発時にアーキテクチャを導入することで，対応関係からモデルベースリファクタリング手法の適用領域が明らかになり，複数の手法の組み合わせを容易に行うことが可能となる．

6. おわりに

本研究では，設計レベルの仕様書を対象とした，より適用範囲の広いリファクタリング手法の考案を目的として，シーケンスベースのリファクタリング手法とパターンベースのリファクタリング手法を組み合わせる手法を提案した．ソフトウェアシステム全体の中で，それぞれの手法が適用可能である範囲を検出し，対象領域を分けて適用し，適用後に組み合わせる．提案した手法を飛行船制御システムと教務支援システムに適用した．

飛行船制御システムに適用した場合，2 つの手法の適用対象領域の重なるクラスが Blimp クラス 1 つであり，うまく領域を分割することができた．一方，教務関連システムに適用した場合，適用対象領域の重なるクラスが KyouikuKatei クラス，Kamoku クラス，JugyouKamoku クラスと 3 つあり，これらのクラス間で相互に依存関係を持っていたため，領域を分割することは困難であった．

本研究で提案した手法の適用可能性はドメイン特性に依存すると考えられるが，2 つのシステムにしか適用していないため，他のシステムに適用して検証することが必要である．

参考文献

[1] M. Fowler, Refactoring: Improding The Design of Existing Code, Addison Wesley, 1999. （児玉公信, 平澤章, 友野晶夫, 梅沢真史 訳，リファクタリング―プログラムの体質改善テクニック，ピアソン・エデュケーション，2000.）
[2] D. Astels, Refactoring with UML, Proc. of International Conference on eXtreme Programing and Flexible Process in Software Engineering 2002, pp.67–70, 2002.
[3] M. Boger, and T. Sturm, Refactoring Browser for UML, Proc. of NetObjectDays 2002, pp.366–377, 2002
[4] 増田敬史，吉田則裕，浜口優，井上克郎，UML モデルを対象としたリファクタリング候補検出の試み，電子情報通信学科技術研究報告，Vol.108，No.64，pp.25–30，2008.
[5] 石川敦啓, 上田賀一, シーケンス図に対するモデルリファクタリングの試み, 情報処理学会研究報告, Vol.2014-SE-183, No.9，2014.
[6] 海老原健一，満田成紀，福安直樹，鯵坂恒夫，事例分析に基づく組込みシステムに適したソフトウェアアーキテクチャの提案，情報処理学会研究報告，Vol.2010-SE-168, No.14, pp.1–7, 2010.