特別研究報告

題目

UML モデルを対象とした
リファクタリング候補検出手法の提案と実現

指導教員

井上 克郎 教授

報告者

増田 敬史

平成 20 年 2 月 19 日

大阪大学 基礎工学部 情報科学科

平成 19 年度 特別研究報告

UML モデルを対象としたリファクタリング候補検出手法の提案と実現

増田 敬史

内容梗概

ソフトウェアの設計を行う際によく用いられる言語として UML（Unified Modeling Language）が挙げられる．また近年ソフトウェアの外部に対する振る舞いを変えずに内部構造を整理するリファクタリングを UML モデルに対して行う研究が盛んである．リファクタリングを行う箇所として，様々なプログラムで再利用できる汎用的な設計パターンであるデザインパターンの適用可能箇所が挙げられる．デザインパターンを利用することで、拡張性の高い設計を実現できる．

しかし大規模なソフトウェアでは，開発者が UML モデルからリファクタリング箇所を特定するのは困難である．よって，既存の UML モデルを解析して，リファクタリング候補を自動で検出できるような仕組みが必要と考えられる．

本研究では，ソフトウェアの UML モデルを解析することで，UML モデル上のデザインパターンを適用できる箇所を検出し，デザインパターン適用後の UML モデルを提示する手法を提案する．UML モデルの中からリファクタリング候補として検出された箇所のみが抽出される．リファクタリングを行う開発者は検出された箇所を UML モデル上で確認し，実際にその箇所にデザインパターンを適用するべきかを判断することができる．デザインパターン適用可能箇所の特定と修正は UML の XML (eXtensible Markup Language) 表現である XMI を解析することで可能である．文書構造化言語である XML は UML モデルを木構造で記述することができるため，XML パーサを利用することで XMI を解析する．

さらに，本手法を実現するツールを作成し，画面に時刻を表示するプログラムを表現した “Digital Clock”の UML モデルや，産業界で開発されたソフトウェアの UML モデル，最後にオープンソースソフトウェアの UML モデルに適用して，実装したツールがリファクタリング候補検出を行えることを確認した．

主な用語

UML モデル（Unified Modeling Language）
リファクタリング
デザインパターン
XML（eXtensible Markup Language）

目 次

1　まえがき

ソフトウェア開発者の間でソフトウェアの理解を共有するための言語として，UML（Unified Modeling Language） [15] が普及している．UML を利用することで，オブジェクト指向設計論 [2] に基づいてソフトウェアの設計モデルを可視化できる．UML で表された設計モデルを共有することで，開発者間の意味解釈の誤りを減らすことができる．

こうして作成された設計モデルは一人もしくは複数の開発者によって何度も改良されることで品質の高い設計モデルとなる．ソフトウェア開発には開発途中に起こる仕様変更や，運用の段階での機能拡張が避けられない．よってこうした設計の変更に柔軟に対応できるような設計モデルの作成が求められる．しかし初めて設計モデルが作成された段階では，後に変更の起こる箇所を予測することは困難なため，変更が起こる箇所が判明する度に設計モデルの改良が必要となる [10]．このように，設計変更の可能性のある箇所が判明した時点で逐次設計モデルを改良して，変更に対する柔軟性を持たせる作業をリファクタリング [7] と言う．

リファクタリングは，“ソフトウェアの外部的振る舞いを保ったまま，理解や修正が容易になるようにソースコードを修正して内部の構造を改善していく作業 [7]”と定義されている．このように元々リファクタリングとはソースコードに対して行うものであった．しかし最近の研究ではリファクタリングを設計モデルに対しても適用するという手法 [5, 3, 17] が提案され，こうしたリファクタリング手法はモデルベースリファクタリング [13] と言われる．設計段階でリファクタリングを行うことで，実装を行った後に設計を変更するという手戻りを減らすことができる．またリファクタリング作業の流れを 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 [5, 3]．モデルベースリファクタリングは，クラスの構造とクラス間の関係を静的に示す UML ダイアグラムの一つであるクラス図を変更することで行うことができる．

リファクタリングの際に，過去に設計者によって発見された設計ノウハウであるデザインパターン [10] を利用する手法がある．デザインパターンは様々なプログラムで再利用できる汎用的な設計パターンであり，デザインパターンを利用することによって，ソフトウェア開発において頻出する典型的な問題に対処できる．UML モデル上にデザインパターンを適用すべき箇所が存在するということは，その箇所の設計に何らかの欠点があり，リファクタリング候補となる可能性が高いことを示している．

開発者はデザインパターン適用可能箇所を検討することで実際にリファクタリングを行うかの判断を行うことができる．しかし，大規模なソフトウェア開発では，開発者がデザインパターン適用可能箇所を UML モデルの集合から自力で特定することは非常にコストがかかる．よって，UML モデルを解析し，デザインパターン適用可能箇所を自動で検出できるような仕組みが必要と考えられる．

本研究では，ソフトウェアの設計時に，クラス図を表す UML モデルで示されるクラスの構造とクラス間の関係を自動的に解析することによって，設計モデルのリファクタリング時に適用可能なデザインパターンを特定し，ソフトウェアの品質向上を支援することを目的とする．UML モデルの解析には，UML の XML（eXtensible Markup Language）表現 [4] である XMI を利用する．クラス図を表す UML モデルはパッケージ，クラス，変数，参照の関係といったデータを持つので，これらを木構造で表現できる．そのため，文章構造化言語である XML を用いて UML モデルのデータ構造を表すことができる．また XML 文書を扱うために様々なアプリケーションが提供されているので，XML を容易に扱うことが出来る．開発者は，UML モデルと XMI ファイルの変換機能を備えた UML モデリングツールを利用することで，UML モデルから XMI ファイルを取得し，XMI ファイルを解析・修正した結果を UML モデルに反映することが出来る．

提案手法では，

1. UML の XML 表現である XMI を解析する．
2. デザインパターンを適用した場合に修正した後のクラス図を示す XMI ファイルを出力する．

という二つの手順を行う．これにより設計モデル上にある修正すべき箇所を特定しやすくなる．リファクタリングを行う開発者は，出力された XMI ファイルを変換して得た UML モデルを見て，設計を変更する可能性のある箇所を確認することができる．例えば開発者は “クラスが追加される”といった設計変更のシナリオを考え，検出された箇所に実際にデザインパターンを適用するかを判断することができる．

さらに，本手法を実現するツールを作成し，画面上に時刻を表示するプログラムを表現した “Digital Clock” [12] モデルや，産業界で開発され実際に稼動しているソフトウェアの UML モデル，オープンソースソフトウェアの UML モデルに適用して，作成したツールの評価を行う．

以降， 2 節で研究の背景となるリファクタリングや，デザインパターンを利用したソフトウェアの品質向上の手法について述べ，それらを使用したソースコードベースのリファクタリングや，モデルベースのリファクタリングなどの既存研究について説明する．3 節では，UML モデル解析の提案手法について述べ， 4 節では提案手法を実装したシステムについて説明する．5 節でシステムを用いた評価実験を行い， 6 節で実験結果の考察を述べる．最後に 7 節でまとめと今後の課題について述べる．

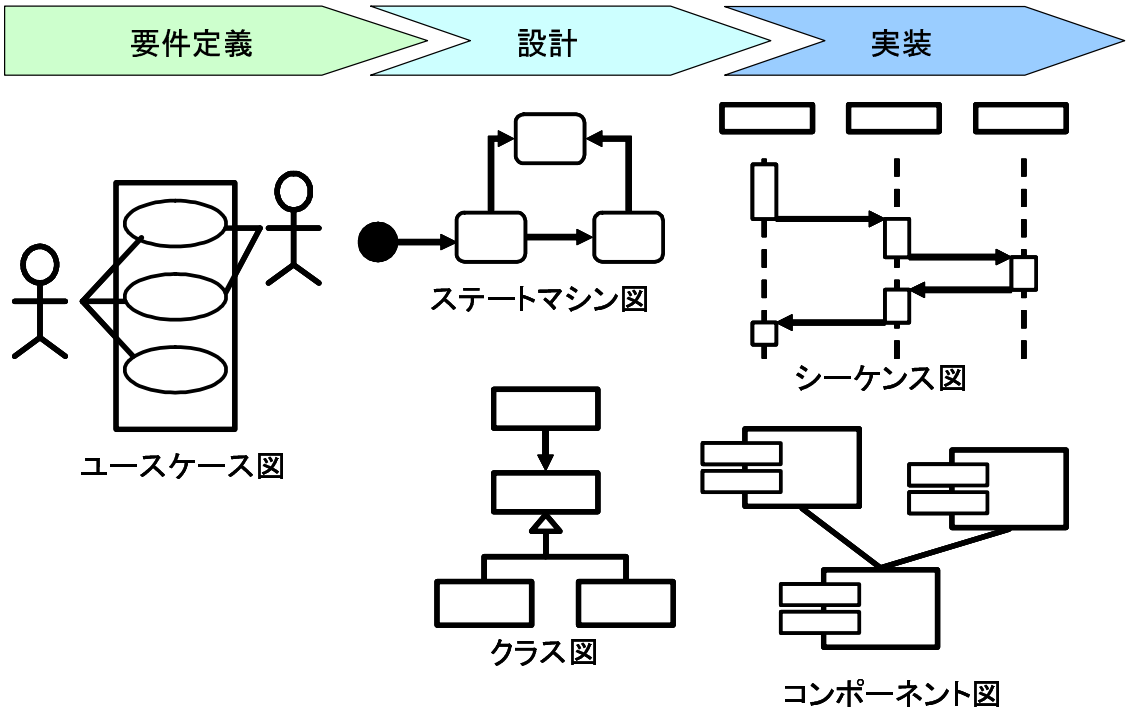


図 1: UML が提供するダイアグラムの一部

2　背景

UML を用いたソフトウェアの開発手法とソフトウェアのリファクタリング，リファクタリングに使用するデザインパターンについて述べ，それらに対する既存研究について説明する．

2.1　UML を用いたソフトウェア開発

本節では UML を用いたソフトウェア開発手法について説明する．モデリング言語である UML(Umified Modeling Language) [15] の普及に伴い，モデルを中心としてソフトウェアを開発するという考え方が広く認知されつつある．

UML は OMG(Object Management Group) [14] によってその仕様が標準化されているモデリング言語である．UML はソフトウェア開発の上流工程から下流工程まで利用でき，かつシステムの様々な側面をモデル化できるように，図 1 のような多くのダイアグラムを提供している (UML2.0 で 13 種類)．加えて，UML では各ダイアグラムに対する表記法とその意味が厳密に定められているため，UML ダイアグラムを用いて記述したモデルはその構成部品 (モデル要素) に至るまで詳細に意味が定義されている．

従来のソフトウェア開発では，仕様書や設計書のような成果物の多くがテキスト文書として自然言語で記述されていた．そのため，記述すべき内容や使用する語彙に関する規則があらかじめ決められていたとしても，曖昧さを取り除くことはできなかった．その結果，開発工程間や開発者間で成果物の解釈を誤ってしまうという問題が起こっていた．

しかし，UML の普及によって，各工程の成果物は意味定義が標準化されたモデルとして記述されるようになった．そのため，従来の自然言語に比べるとはるかに曖昧さを減らすことが可能となり，開発者に共通のコミュニケーション基盤をもたらした．結果，モデルの重要性は広く認識されるようになり，自然言語で記述された設計書やプログラミング言語で記述されたソースコードよりも，モデルを中心としてソフトウェア開発を行う手法が認められるようになった．つまり，要求定義から設計・実装工程にいたるまで，各工程の成果物は全て UML を用いてモデルとして記述し，最終的な実装段階でソースコードとしてプログラミングを行うという流れである．

UML を用いたソフトウェア開発では，後戻りを許さないウォータフォール型ではなく，互いに関連するモデルを繰り返し修正・洗練させていくことで，段階的にソフトウェアを作り上げていく RUP（Rational Unified Process） [11] のような開発プロセスを用いることが多い．

2.2 リファクタリング

ソフトウェアのリファクタリングについて説明し，リファクタリングの適用例を挙げる．

2.2.1 ソフトウェアの特性

書籍 “Refactoring” [7] の中でソフトウェアの 3 つの特性が定義されている． 3 つの特性とは

- 外部から見た動作

  特定の処理を実行する機能を持つこと．

- 修正容易性

  ソフトウェアの変更に対する柔軟性を持つこと．

- 可読性

  ソフトウェアを作成した開発者自身は当然として，ソフトウェアの詳細を知らない開発者にもソフトウェアの意図を伝えることができる構造を持つこと．

である．ソフトウェアは，ある目的を達成するために開発されるツールであるため，少なくとも一度開発されたソフトウェアには特定の処理を実行する機能があるといえる．

しかし修正容易性の低いソフトウェアは，機能の追加の必要が生じたときに，ソフトウェアが変更しやすいものであるかを保障しない．また可読性の低いソフトウェアは，開発担当者以外の技術者がそのソフトウェアの内部を見た時に，理解し易いものであるかを保障しない．

2.2.2 リファクタリングの必要性

リファクタリングとは，ソフトウェアの外部から見た動作，つまり特定の処理を実行する機能を変えずに，内部構造を整理してソフトウェアの修正容易性と可読性を高めることである．

一度作成したソフトウェアの設計を変更しなければならない場面は多い．ソフトウェアの開発途中の仕様変更，デバッグ，製品として運用されている段階での機能追加の要望が出たときなどである．よって，ソフトウェアは変更に対する柔軟性を持っていなければならない．一部に変更を加えたら，他の多くの部分も同時に変更しなければならない様な設計や，一部を変更したことで，他の部分が機能しなくなる様な設計は，見直すべきである．

また，設計をした開発者以外の人間が設計モデルやソースコードをみる場面も多い．開発途中にコードレビューを行って開発チーム全体に知識を浸透させる場合や，一度リリースした製品に対して他の開発者が変更を加える場合がある．よって，ソフトウェアは読み手にその機能を正確に伝える理解容易性を持っていなければならない．

2.2.3 モデルベースリファクタリング

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するにつれ，モデルベースのリファクタリングが新たに定義された [13]．設計モデルに対してリファクタリングを行うということである．モデルベースリファクタリングでは，ソフトウェアの設計品質を向上させる [13]．設計品質には

- 使い易さ（usability）

  ソフトウェアのインターフェースが優れていること．

- 読み易さ（readability）

  設計モデルの理解が容易であること．

- パフォーマンス（performance）

  実行速度が速いこと．

- 移植のし易さ（adaptability）

  プラットフォームの変更などに対応できること．

- 安全性（security）

  セキュリティホールが少ないこと．

などがある．リファクタリングは，設計モデルに model smell [13] と呼ばれる欠点を発見した時に行われる．model smell は

- 冗長さ（redundancies）
- 意味の曖昧さ（ambiguities）
- 設計の矛盾（inconsistencies）
- 設計の不完全さ（incompleteness）
- 脆弱さ（non-adherence）
- 設計への過度な注釈（abuse of the modelling notation）

などを持つモデルを指す．

モデルベースでリファクタリングすることのメリットがいくつかある [5, 3, 17]．一般にクラスの中の重複したコードは少ないほうがいいとされる．ソースコードを変更したときにその箇所と重複した箇所を全て書き換えなければならなくなる，などのデメリットがあるためである．巨大なクラスは，そのクラスがあまりに多くの仕事をしているために，たいていインスタンス変数を持ちすぎていたり，ソースコードが多すぎる場合があるが，このクラスは重複したソースコードの温床となっているのでなんらかのリファクタリングを行ったほうがいい．そういった関連したクラス群に比べて巨大なクラスを見つけるにはソースコードで見るよりもクラス図で見たほうが一目瞭然である．

2.2.4 モデルベースリファクタリングの例

リファクタリングの例を図 2 に示す．左図は，クラス A，B，C，D　が複雑に関連しあっている．この設計では，例えばクラス A に変更が生じた場合，関連しているクラス B，C，D も変更しなければならない可能性が高い．また関連の数が 6 つもあるため，データの流れが理解し難い．

これを解決する為に右図ではクラス M を追加して，他のクラスの調停者として使用している．クラス A，B，C，D はクラス M を仲介して他のクラスと関連付けられているので，

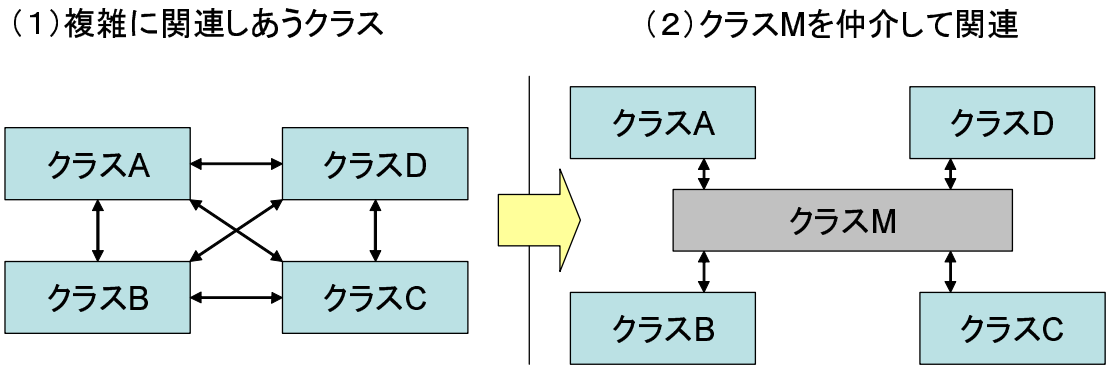


図 2: リファクタリングの例

もしもクラス A を変更した場合もクラス M を変更するだけでよく，クラス B，C，D を変更する必要はない．また関連の数が 4 つと少なくなったことで，データの流れも理解し易くなっている．

2.3 デザインパターン

デザインパターンについて説明し，デザインパターンの適用例を示し，最後にリファクタリングにおけるデザインパターンの利用について説明する

2.3.1 ソフトウェア開発におけるデザインパターン

書籍 “Design Patterns Elements of Reusable Object-Oriented Software” [8] においてデザインパターンの定義が以下のように述べられている．“デザインパターンは，オブジェクト指向システムにおいて重要でかつ繰り返し現れる設計を，それぞれ体系的に名前付けし，説明を加え，評価したものである．” “デザインパターンを使えば，成功した設計の再利用が簡単にできる．”

デザインパターンは汎用的な設計パターンである．デザインパターンを利用することによって，ソフトウェア開発において頻出する典型的な問題に対処するための設計パターンを作成できる．

また著書ではデザインパターンに関しての注意も述べられている．“デザインパターンを無秩序に適用すべきではない．デザインパターンを適用することよって柔軟性を実現できるが，同時にそれは，設計を複雑にし，あるいは性能を犠牲にすることにもつながる．”

デザインパターンを適用しても，その箇所にはそもそも後の修正が必要ない場合もある．よって初めから全ての箇所にデザインパターンを適用すればいいという話ではない．この考

えは後ほど説明するリファクタリングにおいてデザインパターンを利用するという手法に繋がる．

2.3.2 デザインパターンの例

図 3(a) では，統計データが持つ a，b，c それぞれの値を表形式，棒グラフ形式で表現している．いずれかの場所で値の変更が起きたら，他の全ての場所へ変更を通知して常に表示と値の一貫性を保つことが要求される．図 3(b) にこのプログラムを設計した UML モデルが示している．ToukeiData クラス，Table クラス，BarGraph クラスはお互いを知っており，Table クラスの表に変更があった時，Table クラスは他の 2 クラスに変更を伝え，反映させる．

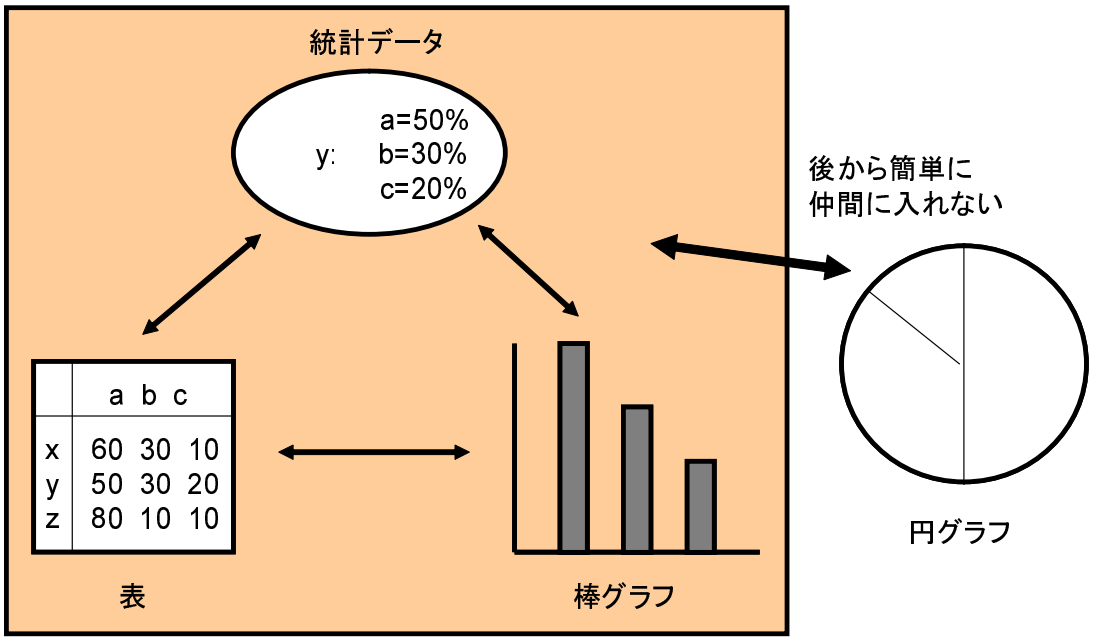
このプログラムに対して，円グラフ表示を行う CircleGraph クラスが追加された場合，他 3 クラス全てが CircleGraph を知り合いに追加しなければならない．今後さらに関係するクラスが増えた場合，ますます変更が困難になることがわかる．

そこでデザインパターンの一つである Observer パターンをこのプログラムに適用する．図 4 に，先ほどのプログラムに Observer パターンを適用したものを示す．統計データのみが全ての知り合い関係を示す知り合い表を持っている．表に変更があった場合は，まずそれを統計データに伝え，データを変更する．そして統計データが棒グラフに変更を伝え，反映させる．図 4(b) にこのプログラムを設計した UML モデルを示した．Table クラス，BarGraph クラスは Observer クラスを継承している．ToukeiData クラスは新たな Observer ポインタを配列（知り合い表）に追加していく．この時 Table クラス，BarGraph クラスはお互いを知っている必要は無い．この場合，円グラフ表示を行う CircleGraph クラスが追加されても，ToukeiData クラスが新たな Observer ポインタを配列に追加するだけで整合性は保たれる．

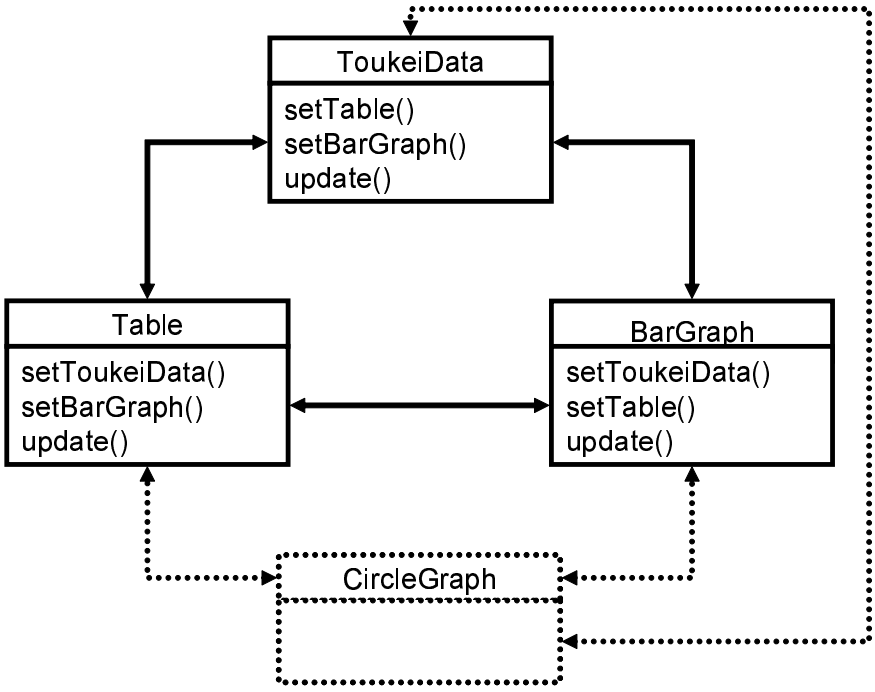
2.3.3 デザインパターンを用いたリファクタリング

パターン指向リファクタリング入門 [10] にデザインパターンについて以下のように述べてある．“デザインパターンを上流設計で使用したり，あまりに早い段階からコードに組み込むよりも，デザインパターンをリファクタリングの道標として利用することが多くの場合において望ましい．”

全ての箇所にできるだけデザインパターンを適用して初めから設計を必要以上に柔軟にしても，後に全ての箇所に変更が必要になることはない．変更が必要であるか定かではないのに，どんな変更にも耐えられるようにデザインパターンを適用することは無意味に設計を大きく，複雑なものにしてしまう．またそれまでに必要となる労力も，多くが無駄になってしまう．このことからデザインパターンは，設計に変更が必要であるとある程度わかった段階

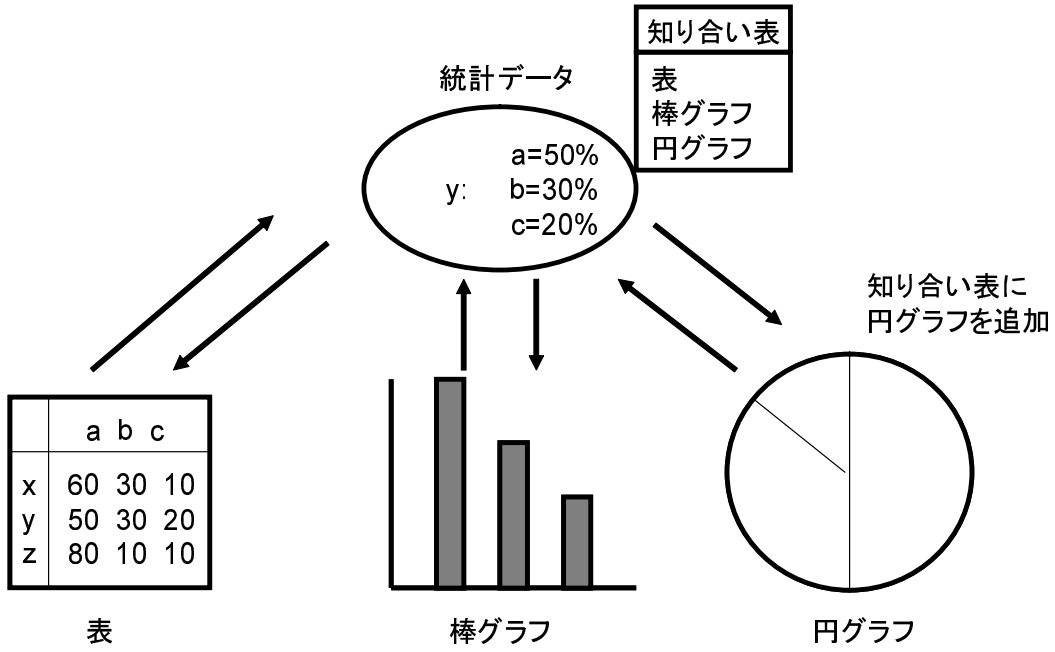


(a) プログラムの動作

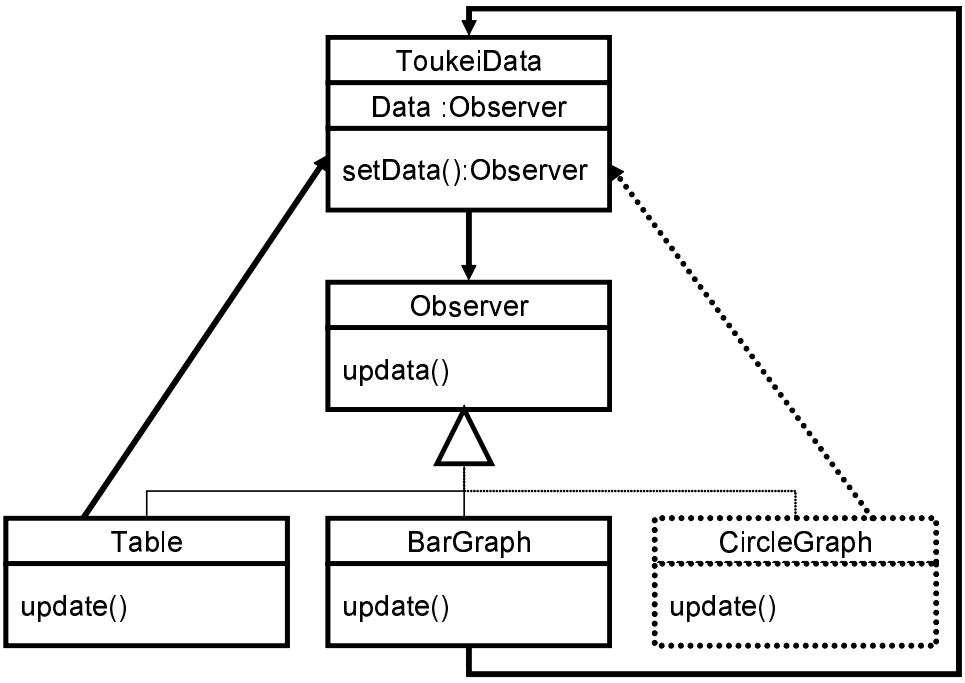


(b) UML モデル

図 3: Observer パターン適用背景



(a) プログラムの動作



(b) UML モデル

図 4: Observer パターン利用

で適用するべきであるという考えがある．つまりリファクタリングにおいてデザインパターンを利用するという手法である [10].

2.4 関連研究

ここではデザインパターンを利用したソースコードベースのリファクタリング手法として Rajesh らの手法 [16] について説明し，デザインパターンを利用したモデルベースのリファクタリング手法として Demirezen らの手法 [6] を説明する．

Rajesh らの手法は，ソースコードに対するリファクタリングにデザインパターンを利用する手法である．作成されたソースコードに対して，デザインパターンが適用可能な箇所を検出して，デザインパターンを適用したソースコードを出力するツール “JIAD” を開発した．本研究では設計モデルに対してリファクタリングを行うことで，ソースコードベースでリファクタリングを実行するよりも上流の段階で設計を洗練させることを目的としている．

Demirezen らの手法は，設計モデルに対するリファクタリングにデザインパターンを利用する手法である．作成された UML モデルに対して，デザインパターンが適用可能な箇所を検出して，デザインパターンを適用したクラス図を出力するツールを開発した．ユーザーは，リファクタリングを実行する UML モデルを設計モデルの中から選んで，デザインパターンが適用可能であるかの判断を下すことができる．本研究では，一つの UML モデルに限らず，設計モデル全体の UML モデルについて，デザインパターン適用可能箇所を検出することを目的としている．

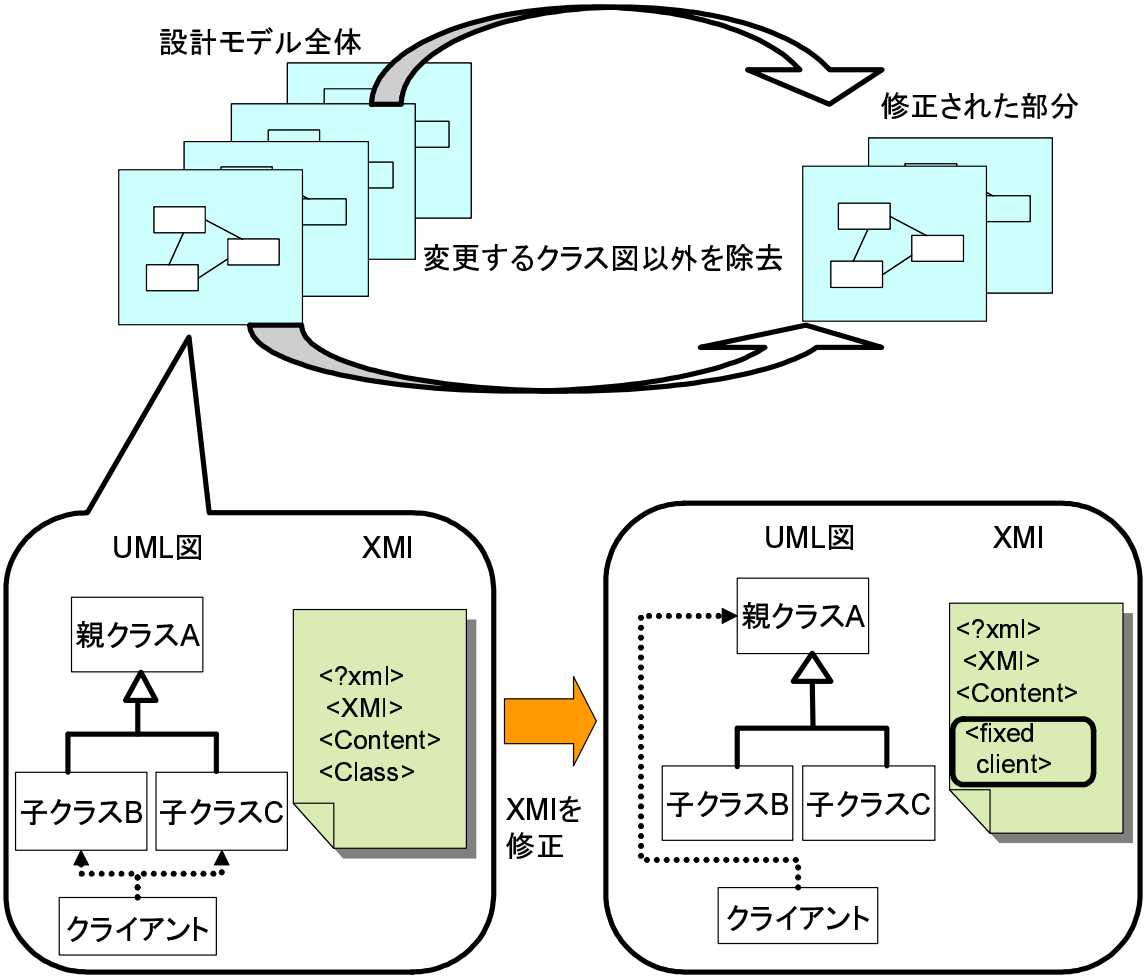


図 5: 提案手法の概要

3 提案手法

本節では，クラス図を示す UML モデルの解析と修正を行い，デザインパターン適用可能箇所の表示をする手法について述べる．まず概要を述べ，続いて手法の詳細を手順ごとに述べる．

3.1 概要

図 5 に提案手法の概要を示す．設計モデル全体の UML モデルの集合を XML 表現で表す XMI ファイルを入力とする．UML モデリングツールを用いて，UML モデルの集合から XMI ファイルを取得する．XMI ファイルを解析し，デザインパターンを適用できる箇所をデザインパターンを適用した記述に修正する．修正した箇所のみ XMI ファイル全体から抽

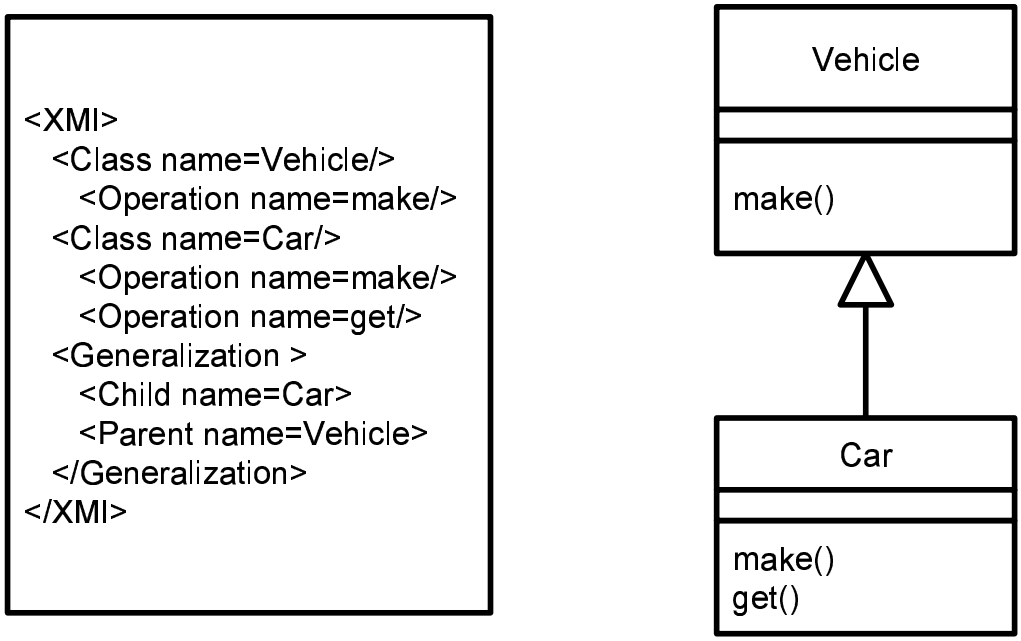


図 6: XMI ファイル (左) と対応する UML モデル (右)

出して出力する．UML モデリングツールを用いて，出力された XMI から UML モデルの集合を取得する．リファクタリングを行う開発者は，得られた UML モデルの集合を見てリファクタリングを行うか否かの判断を行う．

3.2 デザインパターン適用可能箇所の検出と修正

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して修正する手法を示す．

3.2.1 XMI ファイルを用いた UML モデルの解析と修正

UML モデルの集合を XML 表現で表した XMI ファイルを解析，修正することで，対応する UML モデルを修正する．図 6 に単純な UML モデルと，その UML モデルを表す XMI ファイルを示す．なお，XMI ファイルは理解し易くする為に簡略化して示している．二つはお互いに対応しており，一方を変更することで，他方も変更される．

Car クラスは Vehicle クラスを継承しており，Vehicle クラスは make というメソッドを，Car クラスは make，get というメソッドを持つ．XMI ファイルではクラスデータを “Class” タグで表し，その下の階層にメソッドを表す “Operation” タグを持つ．また親子関係は “Gen-

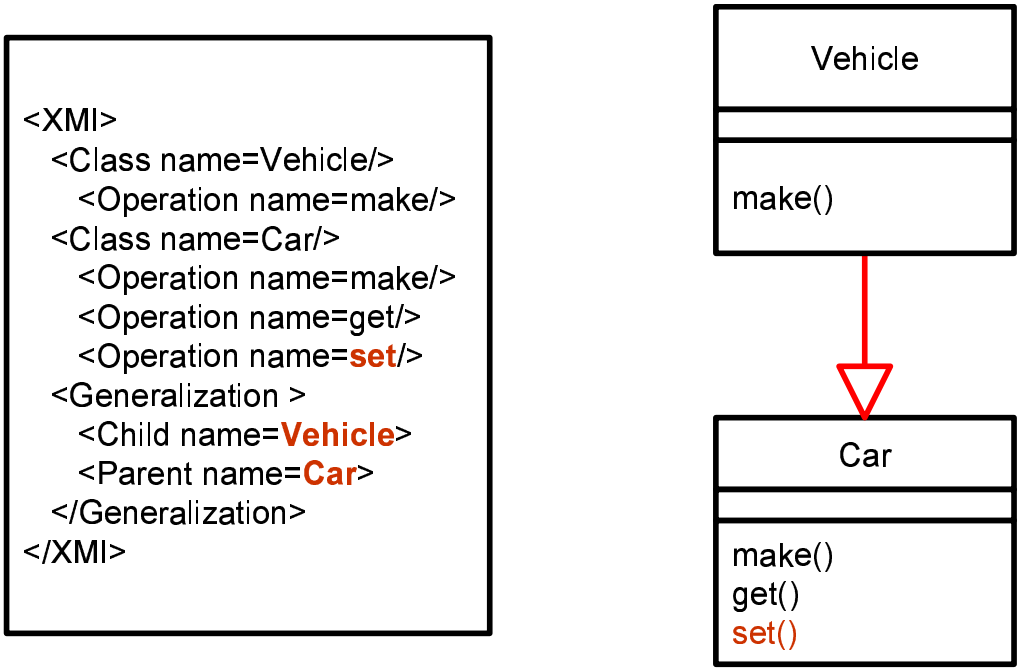


図 7: XMI ファイル (左) を修正することで，修正が反映された UML モデル (右)

eralization” タグで表し，その下の階層に子クラスを示すと “Child” タグと親クラスを示す “Parent” タグを持つ．

図 7 に XMI ファイルを修正することで，修正が反映された UML モデルを示す．XMI ファイルの変更箇所を赤字で示している．Car クラスに set メソッドを追加し，Vehicle クラスと Car クラスの親子関係が逆転している．この変更に対応して，左の UML モデルも修正されている．このようにして UML モデルを修正することができる．

3.2.2 デザインパターン適用条件

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 モデルを解析した結果，特定構造を持っている場合はデザインパターンが適用できる．デザインパターン毎に適用条件を決めておき，条件にあった UML モデルを修正する．図 8 に，2.3.2 章で紹介した Observer パターンを適用する UML モデルの条件を示す．Subject クラスと ConcreteObserver クラスが相互に関連，もしくは依存している場合，Observer パターンが適用できる．

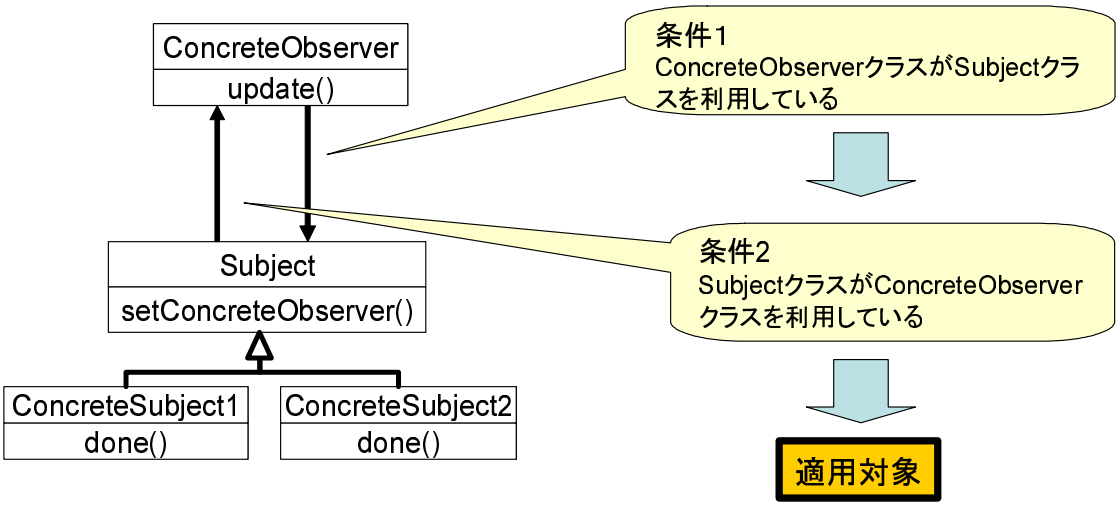


図 8: Observer パターンの適用条件

3.2.3 デザインパターンを適用した UML モデルの修正

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．XMI ファイルに Observer インターフェースを追加し，ConcreteObserver クラスに Observer インターフェースを実装させる．また Subject クラスは Observer インターフェースを利用する．こうすることで Observer インターフェースを実装している他のクラスも Subject クラスを利用できる．図 8 の UML モデルに Observer パターンを適用して修正した UML モデルを図 9 に示す．

3.3 デザインパターン適用可能箇所の表示

デザインパターンが適用できる箇所を全体の XMI ファイルから抽出する．これにより開発者は少ない UML モデルの集合から，デザインパターン適用可能箇所を確認できる．また変更したクラスに注釈を付けることでさらに確認を容易にする．

図 10 に適用可能箇所を全体から抜き出し，デザインパターンが適用できる箇所のみ表示している様子を示す．

全体の XMI ファイルには，PackageA，B，C の記述が含まれていた．修正した箇所が PackageB 中の内容であった場合，PackageB の記述のみを全体の記述から抽出する．PackageB 中のクラス関係が変更されており，修正されたクラスには “Chenged” という注釈が付けられている．

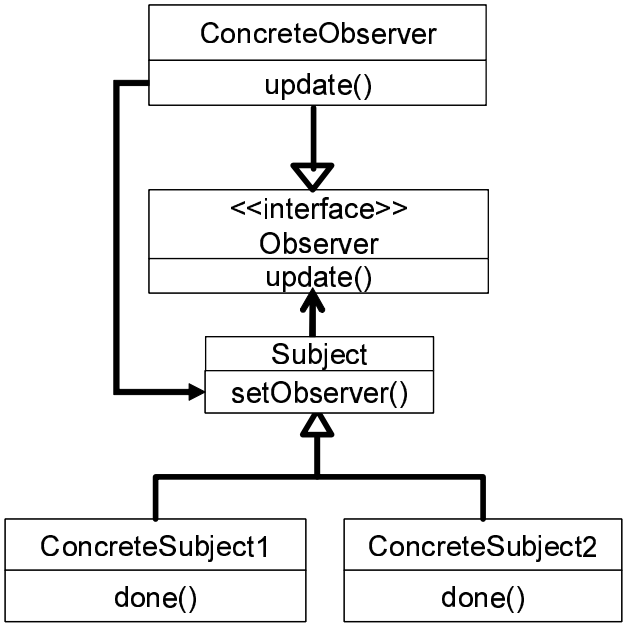


図 9: Observer パターン適用後の UML モデル

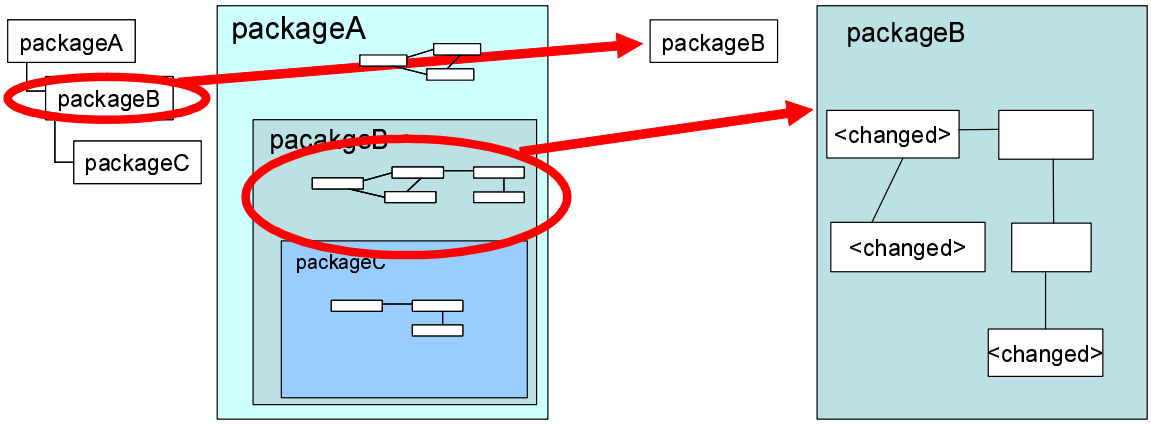


図 10: 修正を加える前の UML モデルの集合 (図左) から，修正を加えた箇所のみを抽出した UML モデル (図右)

4 実装

本節では，作成した XMI 解析ツールの実装について説明する．

4.1 ツールの概要

ツールはXMIファイルの解析･修正を行う機能と，XMIファイル全体から修正箇所の抽出を行う機能の二つを持つ．UMLモデルの集合とXMIの相互変換をするために，“JUDE” [9]というUMLモデリングツールを用いた．開発者は，リファクタリングを行うUMLモデルの集合からJUDEを用いてXMIファイルを取得し，取得したXMIファイルをツールに入力する．ツールは入力されたXMIファイルを解析し，修正後のXMIファイルを出力する．開発者は出力されたXMIファイルからJUDEを用いて修正されたUMLモデルの集合を得ることができる．

XMIファイルの解析・修正を行う機能には，提案手法のうち4.2節で述べたUMLモデルの解析と修正を行う手順が実装されている．またXMIファイル全体から修正箇所の抽出を行う機能には，4.3節で述べた全体のUMLモデル集合のうち修正したUMLモデル集合のみを抽出する手順が実装されている．

4.2 XMIファイルの解析・修正機能

XMIファイルに記述されているUMLモデルのデータを解析し，デザインパターン適用可能箇所を修正する．図11にXMI解析の為に取得するデータを示す．

入力 XMIファイル

出力 修正済みXMIファイル

4.2.1 デザインパターン適用条件に一致する箇所の修正

UMLモデルの修正作業はXMIファイルを書き換えることで行う．

- クラス同士の親子関係を追加，削除する時

  パッケージデータ直下の継承関係データを変更する．

- クラスにメソッドを追加，削除する時

  クラスデータ直下のメソッドデータを変更する．

- クラスのフィールドを追加，削除する時

  クラスデータ直下のフィールドデータを変更する．

- クラスを追加，削除する時

  パッケージデータ直下のクラスデータを変更する．

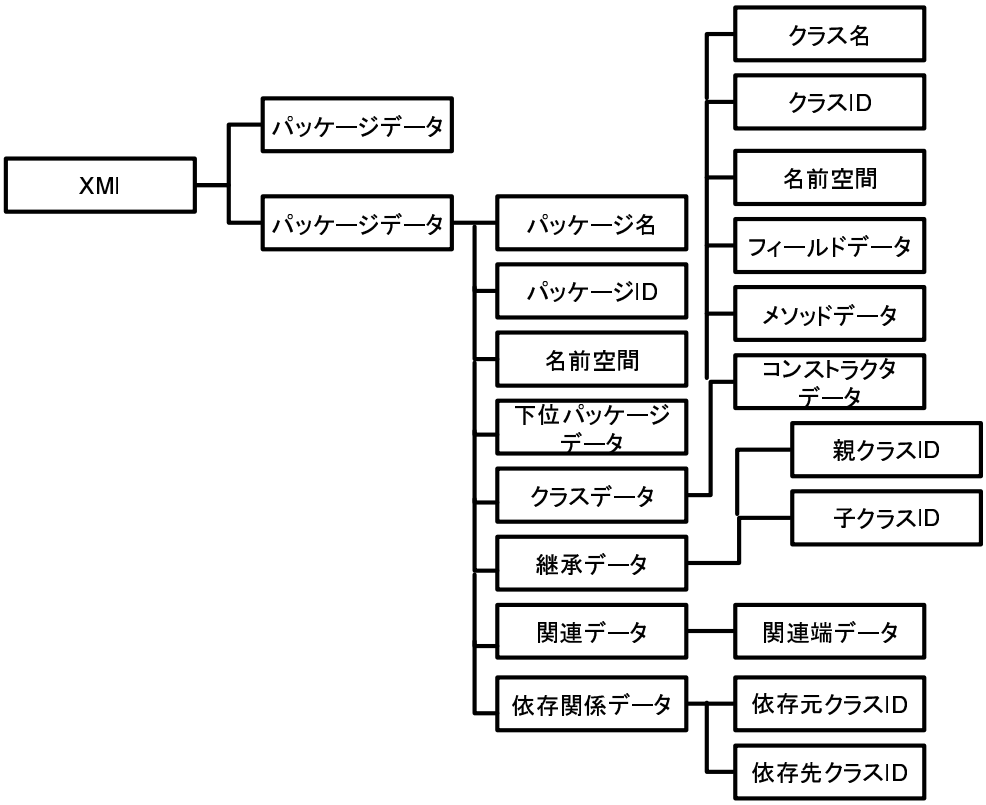


図 11: 取得した UML モデルデータ

- 修正したクラスに注釈を付ける時

  修正したクラスデータにステレオタイプ記述を追加する.

4.3 XMI ファイル全体から修正箇所の抽出を行う機能

入力 修正済み XMI ファイル

出力 修正箇所抽出済み XMI ファイル

XMI ファイル全体から，修正したクラスを含むパッケージと，関連するパッケージのみを抽出する.

4.3.1 抽出する XMI 記述

全体の XMI 記述から，以下の記述を抽出する.

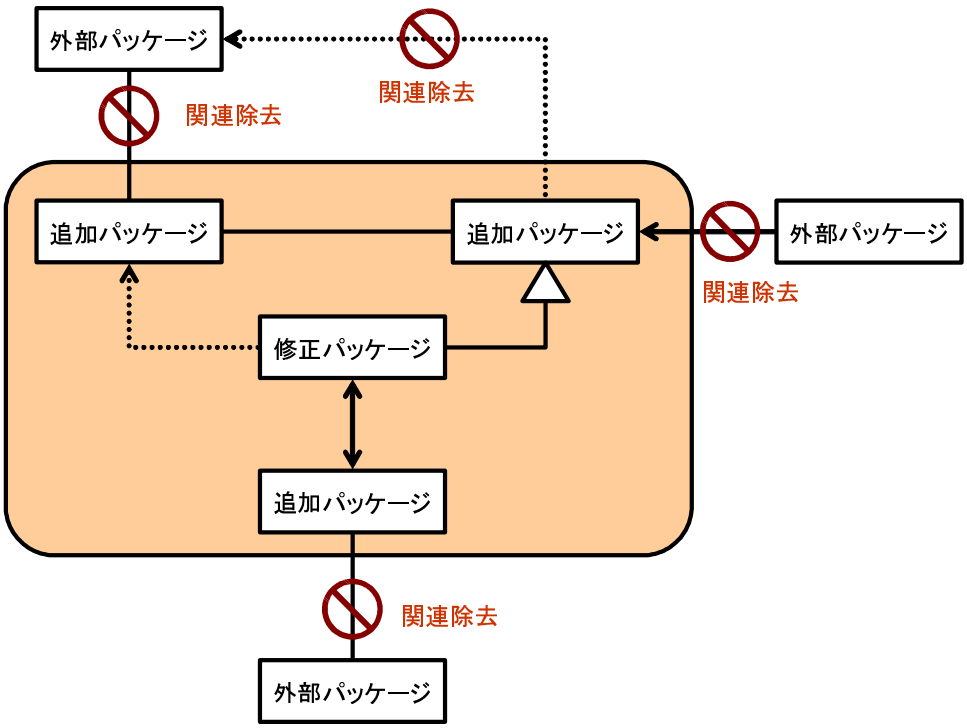


図 12: 抽出するパッケージと除去する参照関係

1. 修正パッケージ記述

   修正したクラスを含むパッケージ記述

2. 追加パッケージ記述

   修正パッケージ記述に他パッケージのクラスとの継承関係，依存関係，関連が記述されている場合，該当パッケージ記述を追加する．

4.3.2 パッケージ間参照記述の除去

修正パッケージ，追加パッケージ以外のパッケージとの参照記述を除去することで，パッケージ間の参照関係を修正パッケージと追加パッケージの二段階に押さえることができる．追加パッケージ記述にある外パッケージに対応する関連データ，依存関係データ，継承関係データを除去する．

図 12 に抽出するパッケージと除去する参照関係を示す．修正パッケージと追加パッケージ間，追加パッケージ同士の参照関係は保持しつつ，追加パッケージとその他の外パッケージとの参照関係は全て除去している．

5　適用実験

5.1　実験目的

提案手法の有効性を確かめるため，適用実験を行った．実験の目的は次の二つである．

- UML モデルに対するリファクタリング候補検出機能を評価

  ツールを使用することで，UML モデル中のデザインパターン適用可能箇所を検出できることを確認し，この機能を利用してリファクタリング候補の検討が出来るかを調べる．

- UML モデルに対する関連パッケージ記述の抽出機能を評価

  ツールを使用することで，UML モデルの集合から注目したいパッケージ記述のみを抽出できることを確認する．

5.2　実験対象

5.2.1　UML モデルに対するリファクタリング候補検出機能を評価

XMI ファイルの解析，修正機能の実験対象は “DigitalClock”モデルと産業界で開発されたソフトウェアの UML モデル（ModelA とする）である．

DigitalClock モデル（7 クラス，1 パッケージ）は書籍 “Agile Software Development” [12] に紹介されているモデルで，Observer パターンを用いたリファクタリングが適用できる典型的な UML モデルである．リファクタリング適用前と後の UML モデルが示されているため，このモデルに対して実装したツールを適用することで，ツールがデザインパターン適用可能箇所を検出することができるかを調べる．

ModelA（104 クラス，12 パッケージ）は大学の組織編制管理システムである．ModelA に対して実装したツールを適用することで，ツールが実在する設計モデルに対して，デザインパターン適用可能箇所を検出できるかを調べる．また ModelA は，Struts というフレームワークを用いて開発されている．フレームワークとは，ソフトウェアを開発する際に頻繁に必要とされる汎用的な機能をまとめて提供し，ソフトウェアの土台として機能するクラスの集合である．

5.2.2　UML モデルに対する関連パッケージ記述の抽出機能を評価

XMI ファイル全体から関連パッケージ記述の抽出を行う機能の実験対象はオープンソースソフトウェアである ANTLR のソースコードを JUDE を利用して変換して得た UML モデルである．

ANTLR [1] は多くの言語（Java，C#，C++等）に対応したコンパイラ・コンパイラである．適用実験では ANTLR3.0（6.9 万行，178 クラス，12 パッケージ）を対象とした．実在するソースコードから JUDE のリバースエンジニアリング機能を用いて，UML モデルを取得し，その UML モデル全体から指定したパッケージを抽出できるかを調べる．

5.3 実験内容

まず DigitalClock モデルにツールを適用し，Observer パターンの適用可能箇所を検出することで，このツールが実際にデザインパターン適用可能箇所の検出に使用できることを示す．

図 13 に，DigitalClock モデルを示す．書籍では，TimeSource クラスと ClockDriver クラスは相互に利用しているため，ClickDriver 以外のクラスが TimeSource を利用することができない．そこでこのモデルに Observer パターンを適用し，ClockDriver のインターフェースとして Observer クラスを追加すべきだと紹介されている．

その後，ModelA に対してツールを適用し，検出した Observer パターン適用可能箇所に対して Observer パターンを実際に適用すべきかという議論をする．これにより作成したツールが，実際に稼動しているソフトウェアのデザインパターン適用可能箇所の検出に使用できることを示す．

最後に，ANTLR の UML モデルから関連パッケージの抽出を実行する．“test”パッケージを除いた 11 のパッケージから “stringtemplate”と “analysis”の二つのパッケージの抽出を行う．これにより作成したツールが，UML モデル全体からの関連パッケージ抽出に使用できることを示す．

5.4 実験結果

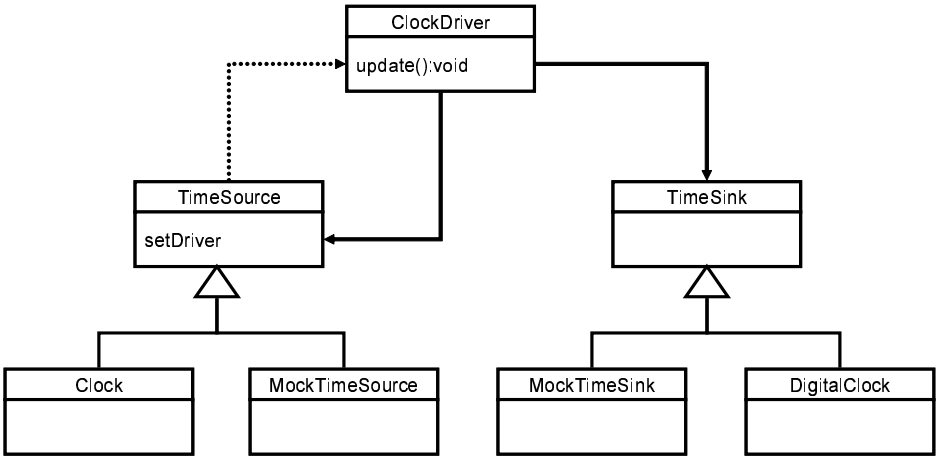
5.4.1 デザインパターン適用可能箇所の検出

図 14 に DigitalClock モデルに対し，ツールを適用することで，Observer パターンの適用可能箇所を検出できた様子を示す．図 14(b) に Observer パターン適用後の UML モデルが示されている．図 13(b) で示した UML モデルと同じように，ClockDriver が Observer を継承している．これにより，ツールを用いて UML モデルを解析することで，実際にデザインパターンの適用可能箇所を検出できることが確認できた．

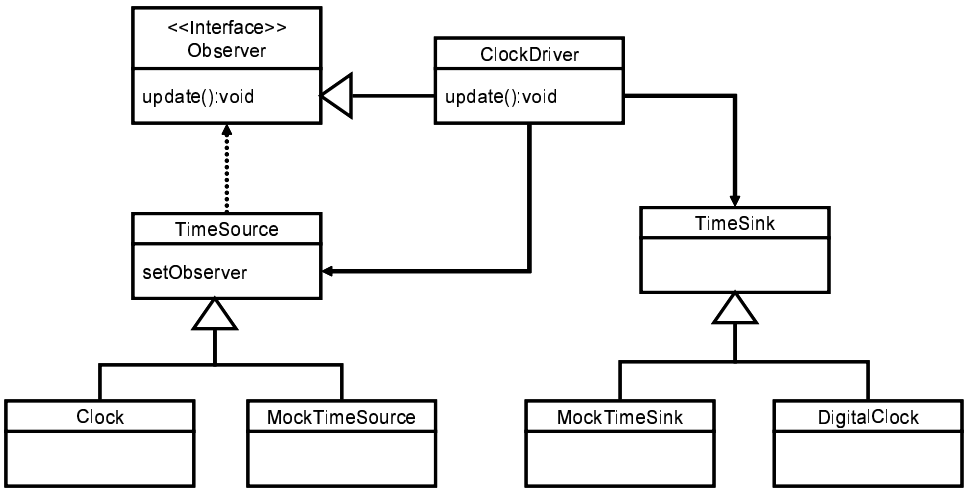
次に ModelA に対してツールを適用し，Observer パターンの適用可能箇所を検出した．

検出された相互依存箇所を以下に示す．

- Action クラスと ActionMapping クラス間



(a) デザインパターン適用前

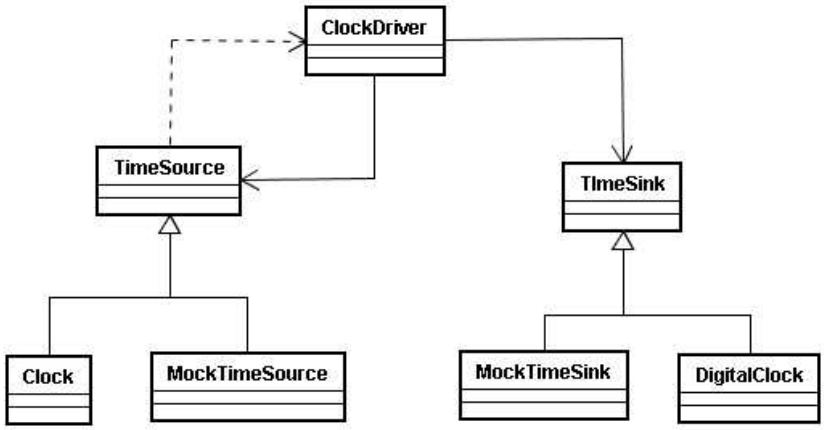


(b) デザインパターン適用後

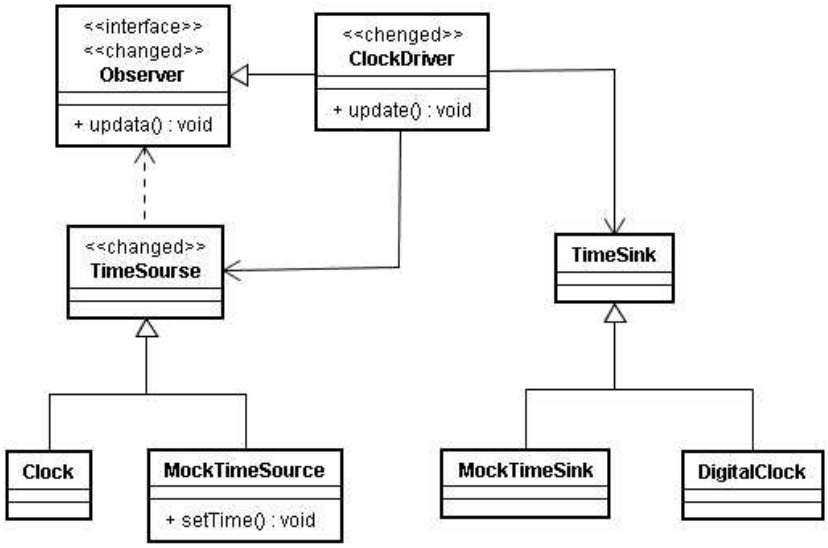
図 13: DigitalClock モデルの Observer パターン適用例

- Action クラスと ActionForward クラス間
- JugyouKamoku クラスと Gakusei クラス間
- Kamoku クラスと Kyouikukatei クラス間

そこで，検出された箇所に実際に Observer パターンを適用すべきか，という議論を行った．まず，Action クラスと ActionMapping クラス，ActionForward クラスの間の依存関係については，これらのクラスは，ModelA が利用している “struts”フレームワークの構成要素の一部であることがわかった．フレームワークの設計はあらかじめ定義されており，開発者はこれらを利用して他のクラスを実装するため，ソフトウェアはフレームワークに依存し



(a) デザインパターン適用前



(b) デザインパターン適用後

図 14: DigitalClock モデルに対しツールを適用

ている，よって Action クラスらの変更は行うべきではないと結論付けた．

次に JugyouKamoku クラスと Gakusei クラス間の依存関係について議論を行った．図 15 に JugyouKamoku クラスと Gakusei クラスに関連しているクラスも含めた UML モデルを示す．

仕様変更のシナリオとしては例えば留学生のデータを持つ “Ryugakusei” クラスなどが追加される，などが挙げられる．この場合 JugyouKamoku クラスが Gakusei クラスに依存しているため，Ryugakusei クラスが JugyouKamoku クラスを利用できない．よって JugyouKamoku

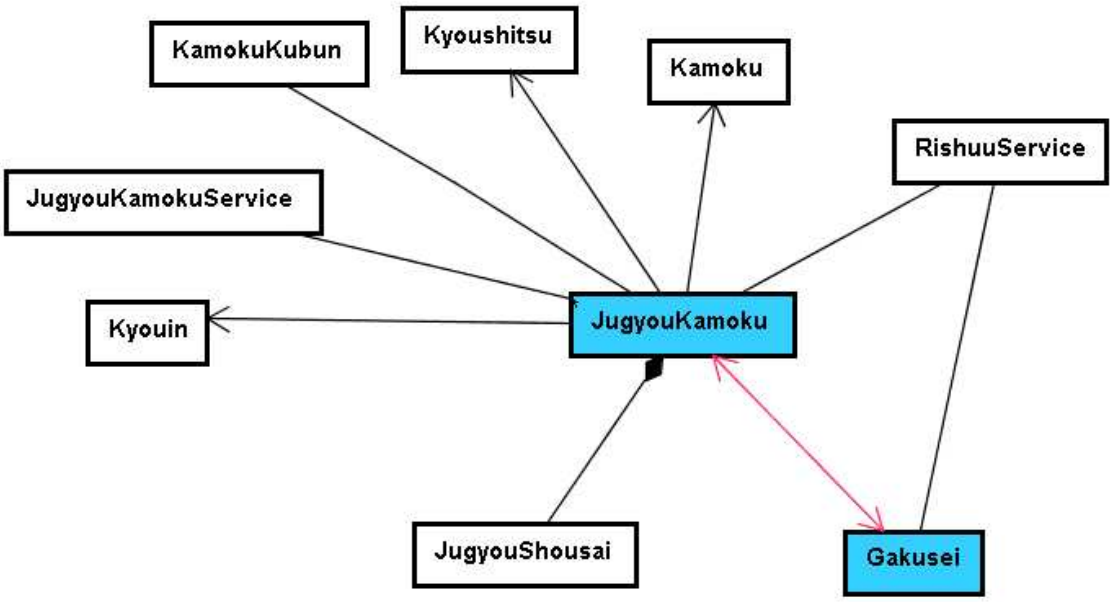


図 15: JugyouKamoku クラスと Gakusei クラス，それに関連するクラスを示す UML モデル

クラスの設計を変更する必要がある．

しかし，仕様書を見ながらさらに細かくクラス関係を確認したところ，RishuuService クラスがあることによって，上で挙げた設計の変更を行う理由が無くなることがわかった．図 16 に，RishuuService クラス，JugyouKamoku クラスと Gakusei クラスの詳細な UML モデルを示す．

RishuuService クラスが JugyouKamoku クラスと Gakusei クラス間のデータのやり取りを行っているため，例え Ryugakusei クラスが追加されても，RishuuService クラスを変更すればよい．よって JugyouKamoku クラスと Gakusei クラスの間に Observer パターンを適用する必要は無い．

最後の Kamoku クラスと Kyouikukatei クラスについても，KyouikukateiService クラスが Kamoku クラスと Kyouikukatei クラスのデータのやり取りを行うため，同じく Observer パターンを適用する必要はない．

こうして，三箇所全てにおいて，リファクタリング候補の検証を行った．ツールを使用して検出された箇所を確認することで，その箇所はリファクタリングの必要が無い，柔軟な設計を持つことが確認できた．

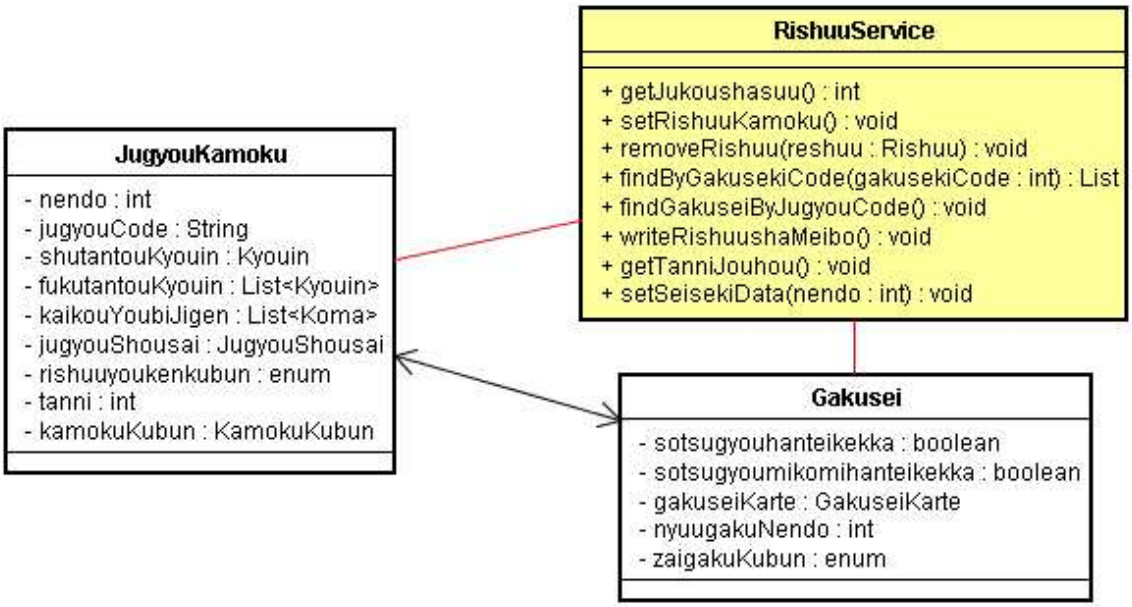


図 16: JugyouKamoku クラスと Gakusei クラス，RishuuService クラスを含めた詳細な UML モデル

5.4.2 関連パッケージの抽出

ANTLR のソースコードから取り出した UML モデルに対して，ツールを適用し，関連パッケージの抽出を試みた．ANTLR にはテストケースで使用するクラスが含まれる．しかし今回はリファクタリング支援を行うツールなのでテストに使用するクラスは全て除いた．ANTLR はテストクラスを除いて総パッケージ数 11，クラス数 148 である．

表 1 に，ANTLR に含まれる test パッケージを除いた全てのパッケージ名とパッケージに含まれるクラス数を示す．

ANTLR 全体から “stringtemplate” パッケージと “analysis” パッケージを指定し，抽出する．表 2 に，抽出したパッケージとそれに含まれるクラス数を示す．“tool” パッケージ，“codegen” パッケージ，“misc” パッケージは指定したパッケージに関連のあるパッケージとして同様に抽出された．

stiringtemplate パッケージに関連している tool パッケージと，analysis パッケージに関連している codegen パッケージ，misc パッケージが追加されたことにより，指定したパッケージの情報を損なっていない．これにより，実装したツールが UML モデルから指定したパッケージを抽出できることがわかった．

表 1: ANTLR に含まれる全パッケージ（test パッケージ除く）とそれぞれのクラス数

パッケージ名	クラス数
antlr	12
collection	1
impl	1
analysis	16
codegen	11
misc	8
runtime	27
debag	15
tree	17
stringtemplate	3
tool	37
パッケージ数	総クラス数
11	148

表 2: ANTLR から stringtemplate パッケージと analysis パッケージを抽出

抽出パッケージ名	クラス数
stringtemplate	3
analysis	17
追加パッケージ名	クラス数
tool	37
codegen	12
misc	8
パッケージ数	合計クラス数
5	89

5.5 考察

ModelA についての適用実験では，リファクタリングをすべき箇所は特に見つからなかった．しかし，ツールを使用して Observer パターンが適用可能な箇所を検出することにより，検出した箇所のクラス関係を議論して，リファクタリングの必要の無い柔軟な設計を持つことがわかった．よって，ツールを使用することでリファクタリングを行う際の支援ができたと思われる．ANTLR に対して指定したパッケージのみを抽出するという実験はうまくいった．これができることにより例えばクラス数が１０００以上もあるようなシステムについても，リファクタリング候補を検出し，小さな範囲で設計を確認することも可能となると思われる．

今回 JUDE を用いて ANTLR のソースコードファイルから UML モデルを取得したが，ソースコード中に記述されているクラス間の依存関係が UML モデルに反映されてないという理由で，ANTLR に対してデザインパターンの検出を行うことができなかった．こうした場合は，UML モデルに反映されているデータを利用してリファクタリング候補を検出し，その箇所についてソースコードを見ながら確認し，さらに候補を絞っていくという方法が考えられる．

ツールを作成するに当たってもっとも注意すべき点は，JUDE が出力する XMI ファイルの文法についてのことであった．パッケージ記述や，クラス記述などのデータを取得する際に，XMI ファイルのどの位置にそれらの情報が記述されているかをチェックしていかなければならない．例えばパッケージ間にまたがるクラス間の関連や関連端の記述がどちらのパッケージデータの中に記述されているかは４通りに分かれている．A パッケージ記述の中に A パッケージ，B パッケージの関連記述が含まれる．B パッケージ記述の中に A パッケージ，B パッケージの関連記述が含まれる．A パッケージ記述の中に A パッケージの関連記述，B パッケージ記述の中に B パッケージの関連記述が含まれる．A パッケージ，B パッケージ以外のパッケージの中に A パッケージ，B パッケージの関連記述が含まれる．このような構造で XMI が記述されているため，パッケージ毎にデータを集めるのではなく，データの種類毎にデータを集めて管理していた．

6 まとめ

本稿では，デザインパターン適用可能箇所を指摘することで，UML モデルに対するリファクタリング候補検出を支援する手法の提案を行った．まず，UML モデルを XML 形式で表した XMI ファイルを解析し，デザインパターン適用条件に当てはまる箇所を検出し，XMI ファイルを書き換えてリファクタリング後の UML モデルを示す．次に，XMI ファイルから指定したパッケージ記述のある箇所を抽出することで，UML モデル全体から注目したい箇所だけを取り出す手法を提案した．リファクタリングを行う開発者は提示されたリファクタリング候補について，そのクラス図を仕様書などを用いて細かい設計まで検討することで実際にリファクタリングを行うかの判断をすることができる．

提案手法の妥当性を確認するための適用実験を行った．実験ではまず，小さな UML モデルにツールを適用することで，XMI ファイルを用いたリファクタリング候補の検出が可能であることを確認した．次に産業界で開発された UML モデルにツールを適用し，実際に稼動しているシステムのリファクタリング候補を検出することを確認した．またオープンソースソフトウェアである ANTLR にツールを適用することで，ツールを用いて検出したリファクタリング箇所が開発者にわかりやすいように，小さな範囲の UML モデルに限定できることが確認できた．

今後の課題として，あるデザインパターンを適用して設計が変更された箇所にさらに他のデザインパターンの適用を指摘すること，UML モデルを用いた有効なデザインパターンを提示すること，パッケージに含まれるクラス数が莫大な数に上るような UML モデルに対してリファクタリング候補をうまく限定することなどが挙げられる．

また本研究では，UML モデルを対象にしたリファクタリング支援を行ったが，UML モデルはシステムの一面を表しているに過ぎず，最終的にはソースコードベースでのリファクタリングが必要になることから，ソースコードベースリファクタリングを支援するツールとの連携が不可欠だと考えている．

謝辞

本研究の全過程において，常に適切な御指導及び御助言を賜わりました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上克郎 教授に心より深く感謝致します．

本研究において，常に適切な御指導及び御助言を賜わりました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下誠 准教授に深く感謝致します．

本研究において，適切な御指導及び御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 石尾隆 助教に深く感謝致します．

本研究において，適切な御指導及び御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 肥後芳樹 助教に深く感謝致します．

本論文を作成するにあたり，逐次適切な御指導，御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 吉田則裕 氏，浜口優 氏，島田隆次 氏，三宅達也 氏，宮崎宏海 氏に深く感謝致します．

最後に，その他様々な御指導，御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上研究室の皆様に深く感謝致します．

参考文献

[1] ANTLR. http://www.antlr.org.

[2] 青木 淳:オブジェクト指向分析設計入門. ソフト・リサーチ・センター, 1993.

[3] D. Astels. Refactoring with UML. Proc. of International Conference on eXtreme Programming and Flexible Process in Software Engineering 2002, pp.67-70, 2002.

[4] eXtensible Markup Language（XML）. http://www.w3c.org/XML.

[5] M. Boger, and T. Sturm. Refactoring Browser for UML. Proc. of NetObjectDays 2002, pp.366-377, 2002.

[6] Z. Demirezen, and N. Y. Topaloglu. A Refactoring Tool for Design Patterns with Model Transformations. Proc. of NWUML, 2006.

[7] M. Fowler. Refactoring : Improving the Design of Existing Code. Addision Wesley, 1999.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns Elements of Reusable Object-Oriented Software. Addison Wesley, 1999.

[9] JUDE. http://jude.change-vision.com/jude-web/index.html. ChangeVision.

[10] J. Kerievsky. Refactoring to Patterns. Addition Wesley, 2006.

[11] P. Kruchten. The Rational Unified Process. Addison Wesley, 2000.

[12] R. C. Martin. Agile Software Development. Pearson Education, 2004.

[13] T. Mens, G. Taentzer, and D. Muller. Challenges in Model Refactoring. Technical Report from University of Mons-Hainaut, Belgium, 2005.

[14] Object Management Group. http://www.omg.org/.

[15] Object Management Group. UML Superstructure Specification Version 2.0, 2005.

[16] J. Rajesh, and D. Janakiram. JIAD:A Tool to infer Design Patterns in Refactoring. Proc. of PPDP 2004, pp.24-26, 2004.

[17] G. Sunye, D. Pollet, Y. L. Traon, and J. M. Jezequel. Refactoring UML Models. Proc. of UML 2001, pp.134-148, 2001.