社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2008-5, KBSE2008-5 (2008-05)

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

増田 敬史† 吉田 則裕†† 浜口 優†† 井上 克郎††

† 大阪大学基礎工学部情報科学科 〒560-8531 大阪府豊中市待兼山町 1-3
†† 大阪大学大学院情報科学研究科 〒560-8531 大阪府豊中市待兼山町 1-3
E-mail: †kis-masd@ics.es.osaka-u.ac.jp, ††{n-yosida,y-hamagt,inoue}@ist.osaka-u.ac.jp

あらまし 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．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．本稿では，この問題を解決するために，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適用可能箇所を検出し，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．さらに，本手法を実現したツールを作成し，リファクタリング候補検出を行えることを確認した．

キーワード UML モデル，リファクタリング，デザインパターン

A Method to Identify Refactoring Opportunities in UML Models

Keishi MASUDA†, Norihiro YOSHIDA††, Yuu HAMAGUCHI††, and Katuro INOUE††

† School of Engineering Science, Osaka University
1-3, Machikaneyama-cho, Toyonaka-shi, 560-8531, Japan
†† Graduate School of Information Science and Technology, Osaka University
1-3, Machikaneyama-cho, Toyonaka-shi, 560-8531, Japan
E-mail: †kis-masd@ics.es.osaka-u.ac.jp, ††{n-yosida,y-hamagt,inoue}@ist.osaka-u.ac.jp

Abstract Unified Modeling Language (UML) is widely used in Object-Oriented software design. Recently, several studies have been done on refactoring for UML models. Refactoring is a well-known process to improve software design. However, it is difficult to identify refactoring opportunities manually in large-scale UML models. In this paper, we propose a refactoring support tool for UML models. The tool automatically identifies refactoring opportunities for applying design patterns that are general repeated solutions to common problems in software design. In addition, we present some case studies to show the usefulness of the proposed tool.

Key words UML Model, Refactoring, Design Pattern

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) 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に基づいてソフトウェアの設計モデルを可視化できる．

こうして作成された設計モデルは何度も改良されることで品質の高い設計モデルとなる．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[4] 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”と定義されている [4]．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 [1], [3] が提案されており，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 [11] 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し，もう一度実装するという手戻りを減らすことができる [1], [3].

リファクタリングの際に，過去に設計者が経験的に得た再利用可能な設計パターンであるデザインパターン [7] を利用する

ことで，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特定することは困難である．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．デザインパターン適用可能箇所の特定はUMLのXML (eXtensible Markup Language) 表現 [2] であるXMIを解析することで可能である．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．

提案手法では，まずUMLのXML表現であるXMIを解析する．次に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．UMLモデルとXMIファイルの相互変換はJUDE [6] などのUMLモデリングツールによって行う．リファクタリングを行う開発者は，ツールから出力された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．

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 [10] 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，作成したを使用することで開発者がリファクタリング候補の確認を行えることがわかった．

2. 準　　備

2.1 UMLを用いたソフトウェア開発

UMLはOMG (Object Management Group) [12] 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)．

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．これによ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．

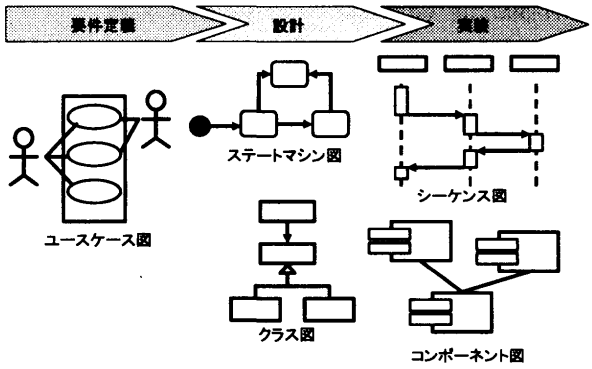


図1　UMLが提供するダイアグラムの一部

2.2 リファクタリング

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．

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．

2.2.1 ソースコードに対するリファクタリング

リファクタリングとはソフトウェアの実装を行った後でソースコードを洗練していき，設計を改善する作業と定義されている [4]．重複したコードをまとめたり，長すぎるメソッドを分割するなど様々なリファクタリングの方法が研究されている．

2.2.2 モデルに対するリファクタリング

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 [11]．図2にUMLモデルをリファクタリングすることで設計が洗練されていく様子を示す．複雑なクラス間の関連を簡潔にすることでデータの流れが開発者にとってわかり易くなる．

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易く，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 [1], [3]．

2.3 デザインパターン

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

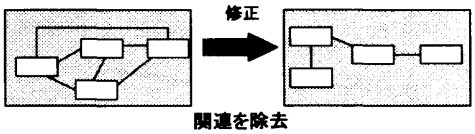


図 2　UML モデルに対するモデルベースリファクタリング

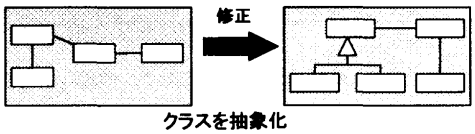


図 3　デザインパターンを利用したモデルベースリファクタリング

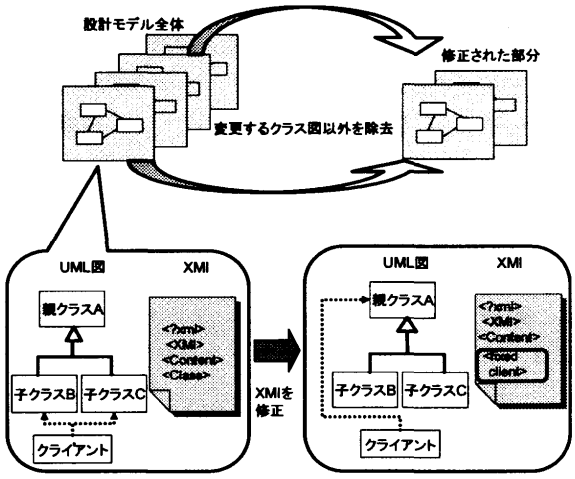


図 4　提案手法の概要

きる [5].

また近年モデルに対するリファクタリング時にデザインパターンを利用する手法が研究されている [7]. この背景には，一般に将来変更が必要にならない箇所にデザインパターンを適用してもかえって設計が複雑になるため，設計の初期段階よりも修正段階でデザインパターンを適用したほうが効率がよいという理由がある.

図 3 に図 2 でリファクタリングした UML モデルに対してさらにデザインパターンを用いたリファクタリングを行う様子を示す. クラスの抽象化を行い，2 つのクラスの共通部分を親クラスから継承させる. これにより後に同じような機能を持つクラスを追加することが容易になる.

3. 提案手法

3.1 概　　要

図 4 に提案手法の概要を示す. ソフトウェアの UML モデルを 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 モデルを取得する.

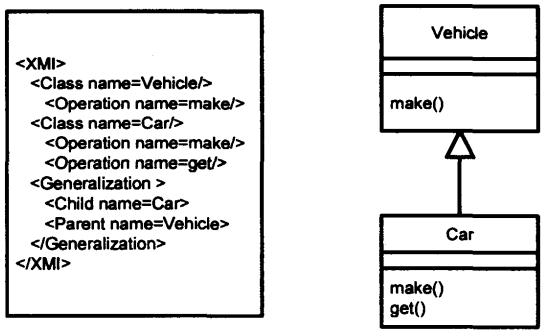


図 5　XMI ファイル (左) と対応する UML モデル (右)

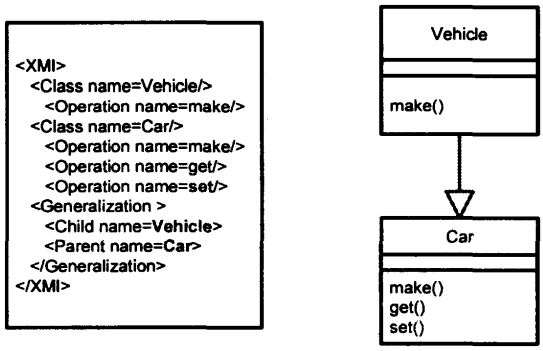


図 6　修正後の XMI ファイル (左) と修正が反映された UML モデル (右)

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 解析による UML モデルの情報取得と修正

ソフトウェアの UML モデルの XML 表現である 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 5 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.

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 Generalization 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 6 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，特定の構造を持つ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.

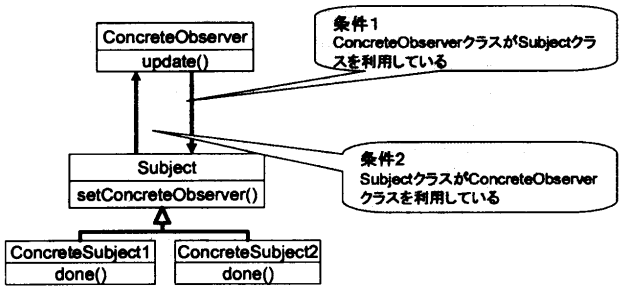


図 7　Observer パターンの適用条件

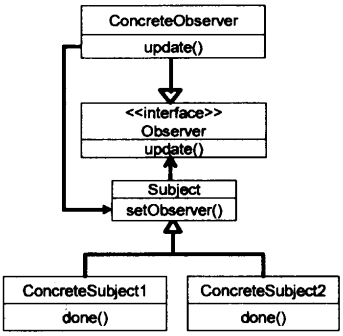


図 8　Observer パターン適用後の UML モデル

実装したツールでは，例えば図 5 の GenerationData 記述を利用して，クラス間の関連を解析し，UML モデルが適用条件に合致するかを判定する．図 7 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している場合，Observer パターンが適用できる．

3.2.3　デザインパターン適用後の UML モデルに修正

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．図 7 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 8 に示す．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．

3.3　デザインパターン適用可能箇所の抽出

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できる．

図 9 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．

4.　適 用 実 験

4.1　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

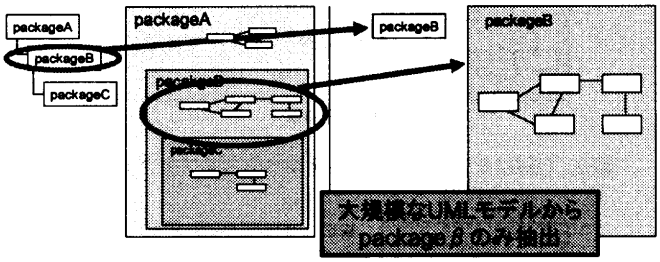


図 9　ソフトウェアの UML モデル (図左) と，修正した箇所の UML モデル (図右)

の目的は次の二つである．

- 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．

- 注目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．

4.2　実 験 対 象

4.2.1　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 モデルとオージス総研が開発したソフトウェアの UML モデルである．

まず DigitalClock モデルで適用実験を行う．DigitalClock モデルは Observer パターンを用いたリファクタリングができる典型的な UML モデルとして紹介されている [10]．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．

次にオージス総研 [8] が開発したソフトウェアの UML モデルに対して適用実験を行う．このソフトウェアは大学の教務関係業務システムである．業務システムの UML モデルに対して実装したツールを適用することで，ツールが実在する大規模な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．

4.2.2　UML モデルに対する注目パッケージ記述の抽出機能を評価

XMI ファイル全体から注目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 は多くの言語 (Java，C#，C++等) に対応したコ

表 1　実験対象のデータ

UML モデル	クラス数	パッケージ数
DigitalClock モデル	7 クラス	1 パッケージ
教務関係業務システムの UML モデル	104 クラス	12 パッケージ
Antlr3.0 の UML モデル	178 クラス	12 パッケージ

ンパイラ・コンパイラである．適用実験では ANTLR3.0 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．

4.3 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，このツールが小規模なモデルに対する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．

その後，業務関係業務システムの UML モデル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 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を，大規模なソフトウェアに対して適用することで開発者が小規模な範囲でリファクタリング候補の検討が行えるかを示す．

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．

4.4 実験結果

4.4.1 デザインパターン適用可能箇所の検出

図 10 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．ツール適用前の図では，現在の時刻のデータを持つ TimeSource クラスとその時刻をデジタルで表現する DigitalClockDriver クラスが相互に関連している．ツール適用後の図では，Observer クラスを導入することで，TimeSource クラスと DigitalClockDriver クラスの相互関連が除去されている．これにより，例えば現在の時刻をアナログで表現する AnalogClockDriver が追加する時も Observer クラスを継承することで可能になる．

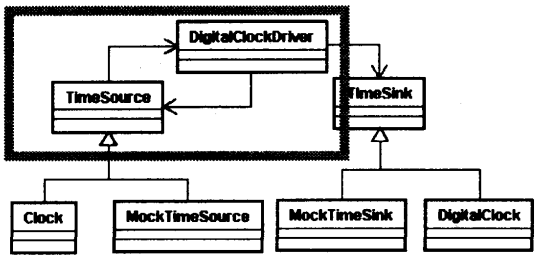
次に教務関係業務システムの UML モデルに対してツールを適用し，Observer パターンの適用可能箇所を検出した．検出された相互依存箇所は 3 通りであった．

そのうちの一つである 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 11 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．

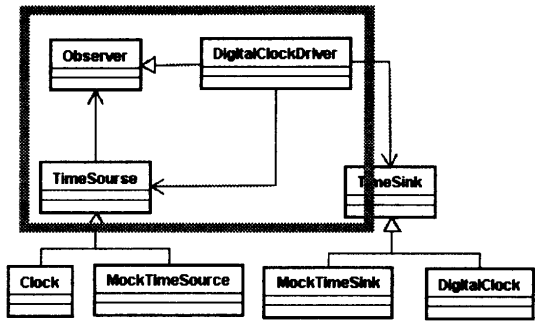
仕様変更のシナリオとしては例えば留学生のデータを持つ Ryugakusei クラスなどが追加される，などが挙げられる．この場合 JugyouKamoku クラスが Gakusei クラスに依存しているため，Ryugakusei クラスが JugyouKamoku クラスを利用できない．よって JugyouKamoku クラスの設計を変更する必要がある．

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 12 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．



(a) ツール適用前



(b) ツール適用後

図 10 DigitalClock モデルに対しツールを適用

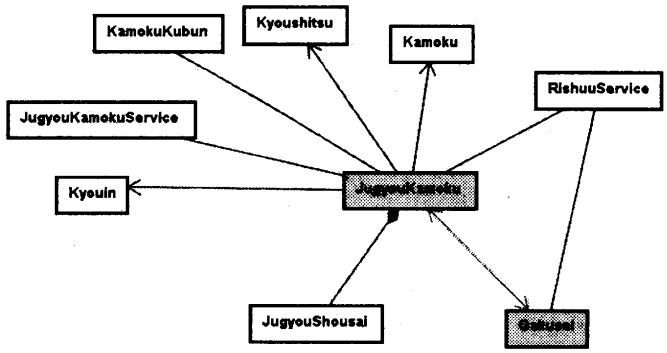


図 11 JugyouKamoku クラスと Gakusei クラス，それに関連するクラスを示す UML モデル

他の Observer パターン適用可能箇所についても調べたところ，同じように相互依存にあるクラスの間にそれらのデータを管理する Service クラスがあることがわかった．

4.4.2 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．

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

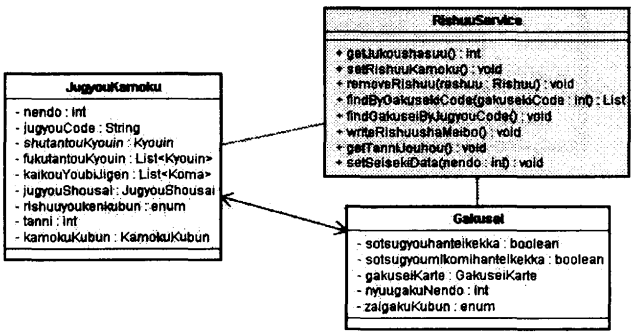


図 12 JugyouKamoku クラスと Gakusei クラス，RishuuService クラスを含めた詳細な UML モデル

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．

4.5 考　　察

オージス総研が開発した教務関係業務システムの UML モデルに対しての適用実験では，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．また ANTLR に対して注目したいパッケージのみを抽出することができた．これができることによりツールを使用して大規模なソフトウェアの UML モデルから，リファクタリング候補を検出し，開発者が小規模な範囲で設計を確認することも可能となると思われる．

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．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．よって実装したツールではパッケージ毎にデータを集めるのではなく，種類毎にデータを集めて管理していた．

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	77

5. む　す　び

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 モデルを解析し，デザインパターン適用条件に合致する箇所を検出し，デザインパターン適用後の UML モデルに変更する．次に，開発者が注目したい箇所を UML モデル全体から抽出する手法を提案した．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，小規模な UML モデルにツールを適用することで，デザインパターン適用可能箇所の検出ができることを確認した．次にオージス総研が開発した教務関係業務システムの UML モデルにツールを適用し，大規模なソフトウェアのリファクタリング候補を検出できることを確認した．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，小規模な 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，様々なデザインパターンでより多くの例題に適用し，手法の有用性を定量的に評価することなどが挙げられる．

謝辞

本研究は一部，文部科学省 “次世代 IT 基盤構築のための研究開発”の委託に基づいて行われた．また，日本学術振興会 科学研究費補助金 特別研究員奨励費 (課題番号:20・1964) の助成を得た．

文　　献

[1] D. Astels. Refactoring with UML. Proc. of International Conference on eXtreme Programming and Flexible Process in Software Engineering 2002, pp.67-70, 2002.
[2] eXtensible Markup Language (XML). http://www.w3c.org/XML.
[3] M. Boger, and T. Sturm. Refactoring Browser for UML. Proc. of NetObjectDays 2002, pp.366-377, 2002.
[4] M. Fowler. Refactoring : Improving the Design of Existing Code. Addision Wesley, 1999.
[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns Elements of Reusable Object-Oriented Software. Addison Wesley, 1999.
[6] JUDE. http://jude.change-vision.com/jude-web/index.html. ChangeVision.
[7] J. Kerievsky. Refactoring to Patterns. Addition Wesley, 2006.
[8] オージス総研. http://www.ogis-ri.jp/.
[9] P. Kruchten. The Rational Unified Process. Addison Wesley, 2000.
[10] R. C. Martin. Agile Software Development. Pearson Education, 2004.
[11] T. Mens, G. Taentzer, and D. Muller. Challenges in Model Refactoring. Technical Report from University of Mons-Hainaut, Belgium, 2005.
[12] Object Management Group. http://www.omg.org/.