O-002

デメテルの法則に基づくクラス設計支援

― Eclipse plug-in 機能としての実装 ―

A software Tool to support Object Class Design Based on the Law of Demeter in Object Oriented Programing: An Implementation by using Eclipse Plug-in Facilities.

千葉 亮太† 　橋浦 弘明† 　古宮 誠一†
Ryota Chiba 　Hiroaki Hashiura 　Seiichi Komiya

1．まえがき

今日の情報化社会において、ソフトウェア開発は大規模化、複雑化してきている．そのため、ソフトウェア開発には多くの知識が必要となってきている．特に、近年よく使われているオブジェクト指向には OOSE、OMT などの代表的な設計法があるが、これらの設計法の共通の欠点として、クラスの設計が難しいという問題がある．また、オブジェクト指向プログラミング言語や、モデリング言語を知識として習得した技術者が、その背景にある考え方を知らずに技術を適用する結果として、オブジェクト指向技術が本来もつ、堅牢性、拡張性などの利点が得られなくなる．

そのため、企業ではソフトウェア開発の経験者、ソフトウェアに関する知識やスキルを持つ人材の育成を大学に求めている．そこで、本学では高度情報演習 2B と言う授業を通して、オブジェクト指向による実践的なソフトウェア設計・開発をプロジェクト形式で学ばせている．

ソフトウェア開発に関する深い知識や高いスキルを持つ人材を育成するためには、良いソフトウェアを設計するための方法(ソフトウェア設計法)を習得させる必要がある．そのためには、ソフトウェア設計法を支援するためのツールを開発する必要がある．

ソフトウェア設計法にはさまざまな方法が存在する．ソフトウェアの品質は ISO/IEC 9126-1 によって定義されている．本研究では、それらの定義の中からソフトウェアの保守性について着目した．

そこで、本研究ではソフトウェア設計案の良さを測る指標として結合度を用いる．結合度はモジュール間の関連の強さのことで、モジュール間の結合度が弱いほど良いモジュールであるといえる．この中より、モジュール間の相互依存関係に注目して、メッセージの宛先に制限を加えることにより、結合度を減少させる．結合度を下げる手法としてデメテルの法則を利用し、依存関係の検出を行うツールを開発することで、依存関係を検出して、直す必要があるということを教える．

最後に本稿の構成を以下に示す．2 では本研究の関連研究を示し、本研究の位置づけを示す．3 ではデメテルの法則と、それを本研究で拡張したものを示す．4 では開発したツールの概要と、実装方法を述べ、5 ではその開発したツールで取得した結果を考察する．6 では本稿のまとめを述べる．

† 芝浦工業大学大学院 工学研究科

2．関連研究

Mayers はソフトウェアの保守性を高めるためには、モジュール分割が必要であると言っている．モジュール分割を行ううちの 1 つの手法の中に、モジュールの独立性を考えて分割する手法がある．Myers [1]はプログラムモジュールの独立性を測る尺度として、モジュール凝集度（モジュール強度とも呼ばれる）とモジュール結合度を提唱している．

これらを定量的に評価するためにメトリクスという指標で計測することを Shyam ら[2]は提案している．

これらのメトリクスを IDE 上で調べるためのツールとしては、「Eclipse Metrics Plugin [3]」や「Metrics plugin for Eclipse[4]」等がある．Eclipse Metrics Plugin は、パッケージ内、もしくは Type ごとの結合度を数値として表すことが可能である．Metrics plugin for Eclipse はパッケージ間にどのような依存性があるかを視覚的に表示することが可能である．これらのツールを使うことにより、開発中においてもリアルタイムにメトリクスを調べることが可能になる．

しかしながら、これらのメトリクスの数値を適切な数値に持って行くことが必要であるが、これらのツールではどのようにすれば数値が最適になるのかは示されず、そのための効果的な手段や方法も提示されない．そのため、適切な数値にするためには、開発者の経験や、勘によるところが多かった．

本研究では、ソースコードからパッケージ内のどの部分が実際に結合をしているかを提示することにより、クラスの設計を支援できるツールを開発したい．

3．デメテルの法則

3.1 デメテルの法則

デメテルの法則は、オブジェクト指向ソフトウェアにおいて良い設計案を生み出すためのルールとして、1987 年の秋にノースイースタン大学の Lieberherr ら[5]によって提案された．

デメテルの法則は、図 1 のように、「基準となるオブジェクトに隣接するオブジェクトとしか通信してはならず、離れたところにあるオブジェクトとは直接通信してはならない」というものである．

これは、Larman[6]によって示された"Low Coupling Pattern (疎結合性パターン)"に他ならない．即ち、デメテルの法則は、必要のない結合を明らかにする仕組みであると言い換えることができる．

なお、Lieberherr らは、基準となるオブジェクトと直接通信してよいオブジェクトのことを“friend”と名付けて

いる．この friend と呼ばれるオブジェクトは、文献 5 によって C++プログラム向けに定義されている．

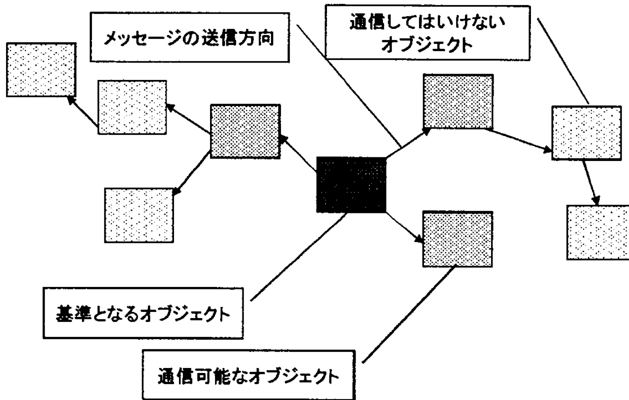

図 1　デメテルの法則

3.2　Java 言語向けに拡張したデメテルの法則

このようにデメテルの法則は、この法則に従ってプログラムを設計すると、オブジェクトの独立性が高くなり、Myers らによって定義された結合度が下がるように設定されている．“friend”という概念は、オブジェクト指向言語の C++を想定して 1988 年に定義された（この当時は Java という言語は無かった）．そこで Java でも適用できるようにするために friend を次のように定義する．

あるオブジェクトのどんなメソッドも、下記に示すいずれかのメソッドだけを呼び出すべきである．

1.オブジェクト自身
2.オブジェクト自身へのパラメータとして渡されたオブジェクト
3.オブジェクト自身が属性として保持しているオブジェクト
4.オブジェクト自身のメソッド内で生成したオブジェクト
5.JAVA の組み込みクラス・基本型

上記のいずれかの条件を満たすオブジェクト間に限って、相互のメッセージ交換を許す(クラス間の相互依存関係の制限に加えない)ことにした．

特に図 2 は 5 の定義を表したものである．これは Java で用いられている組み込みクラスや基本型を、OS の一部分であると判断し、依存関係には含まないものとするため、friend とした．

3.3　デメテルの法則の適用範囲

デメテルの法則はクラス間の相互依存関係に注目して、メッセージの宛先を制限することで、クラス間の結合度を低くする狙いがある．クラスという最も細かい粒度でデメテルの法則を適用すると、ソフトウェア開発の規模が大きくなるにつれて、各オブジェクトにとって friend の関係にあるオブジェクトの数(結合度の数値)が飛躍的に大きくなる．このため、friend の関係にあるオブジェクト同士でも、メッセージ交換が本当に必要なオブジェクトなのか、メッセージ交換を取り止めても良いオブジェクトなのかの判断が付かなくなる．そこで本研究では、クラスではなく、複数のクラスをまとめたパッケージを単位として採用した．

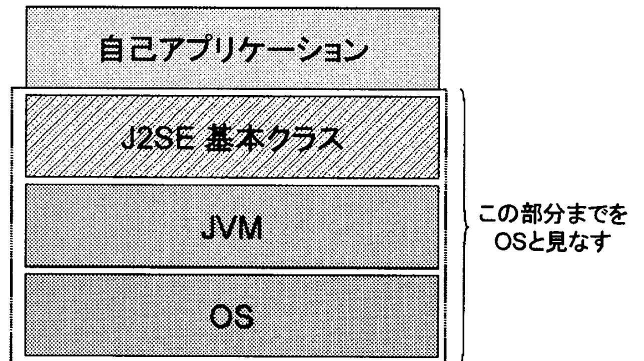

図 2　Java 基本クラスの位置づけ

4.　ツールの開発

4.1　ツールの必要性

我々が提案するパッケージ間における結合度の測定方法は、プログラムに出現するデータ型のすべての種類に対して、同一パッケージ内の同一データ型から発行されるメッセージを 1 つと数えることで、パッケージ間で存在している依存関係の数をパッケージごとに数える方法である．この手法で依存関係を数え上げると、ソースプログラムの規模が小さければ人手で数え上げることも可能であるが、プログラムが大きくなってくるとパッケージの数も増えるので、すべての種類のデータ型を対象に依存関係の数を数え上げるのは時間がかかってしまう．そのため、パッケージの中で定義されているすべてのデータ型をツールによって自動抽出し、その抽出結果に基づいて、パッケージごと、かつ、データ型の種類ごとに、同一視してもよいかどうか判断し、パッケージ間における依存関係を自動的に検出することが可能であれば、検出にかかる時間を減らすことが可能である．

また、これらの依存関係を見つけるのは誰でもできるのではなく、検出・識別のために、モジュールの独立性についての知識が必要となり、我々が支援しようとするプログラミング初学者では判断が難しい．従って、依存性の検出・識別を教授者側が行うのは、演習を行うグループの数だけプログラムがあるので、手作業では不可能である．

4.2　ツールの概要

Eclipse[7]とはフリーの統合環境(IDE) である．統合開発環境とはソフトウェアを開発する上で必要なエディタ、コンパイラ、デバッガなどのプログラミング必要なツールを統一されたインターフェースでまとめ、統合的に開発が行える環境のことである．また、Eclipse は、多くの企業において Java の開発で用いられているという実績がある．

このような背景から、我々は、Eclipse のプラグインとしてツールの実装を行うことにした．本ツールは図 3 のような形で提供される．ツールに対してソースコードがおいてあるディレクトリを指定し、検索を行うパッケージ名を指定すると、検索対象となったソースファイルの解析を行い

3.2 で定義した friend の定義に従って違反しているクラスを検索する.

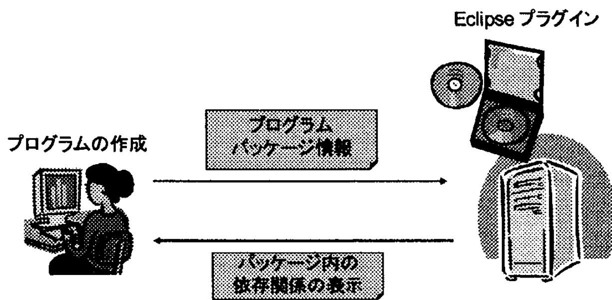

図 3　ツールの概要

4.3　ソースの解析方法

ソースコードをモデル化して操作するには Java エレメント API を使用する方法と DOM/AST API を利用する方法がある. しかし、Java エレメント API を利用すると、ソースコードの中で定義されているメソッド、コンストラクタのシグネチャの情報を取得することはできるが、メソッド内部で記述されている処理コードについての情報を取得することはできない. それに対し org.eclipse.jdt.core.dom パッケージに含まれている DOM/AST API は Java ソースコードを Java エレメントよりも詳細にモデル化しているためメソッド内部の情報も取得できる. また、AST は不完全、もしくは正しくないソースでも受け入れるため、作成途中のソースコードでも解析することが可能となる. そこで、我々は DOM/AST API を用いてソースコードを解析することにした.

ソースコードを解析する手順は図 4 の通りである

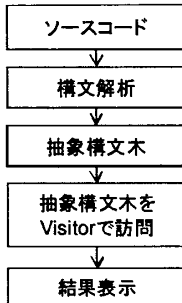

図 4　ソースの解析順序

ソースコードをツールに入力すると、DOM/AST が構文解析を行い、抽象構文木(AST: Abstract syntax Tree)を作成する. この作成された AST を Visitor パターン[8]を用いて依存解析を行いたいパッケージ中に存在するクラスの完全修飾名を取得することができる. この完全修飾名から、3.2 で定義された対象となるクラス名のみを抽出することで、friend を検出することが可能になる. その friend クラスの結果を用いて、パッケージ外へ通信を行っているクラスの検出を行う.

5.　実験

5.1　実験目的

パッケージ間の依存関係を調べるためには、3.2 章で定義した friend を検出する必要がある. そこで本実験では与えられたソースコードの解析を行い、指定されたパッケージ内における friend の完全修飾名を検出することが可能であるか実験を行う. その結果を用いて、パッケージ間における通信をしているクラスの検出を行った.

また、オープンソースで開発されているプロダクトを例にとり、パッケージ間の依存数を示すだけでは依存関係を解決することが難しいことを示し、実際に依存している部分を示すことで、本ツールが問題点を解決していることを示す.

5.2　実験内容

はじめに図 5 のようなクラス図で示されるソースコードを作成し、そのソースコードを開発ツールで解析を行い、実際にパッケージ間における依存が検出できるかの確認を行う. (実験 1)

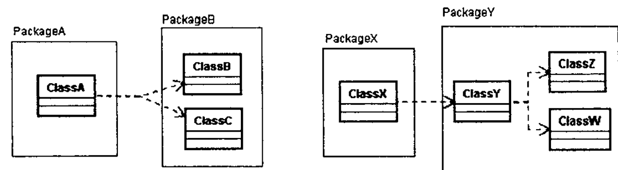

図 5　実験 1 のクラス構成

次に、オープンソースプロダクトである Apache Tomcat[9]の Version 5.5.20 のソースコードの中から、org.apache.catalina.*の各パッケージに対してこのツールを適用した. (実験 2)

5.3 実験結果と考察

実験 1 の結果を表 1 に示す.

表 1　実験 1 のツールの検出結果

依存関係元	依存関係先
PackageA.ClassA	PackageB.ClassB
	PackageB.ClassC
PackageX.ClassX	PackageY.ClassY

図 5 において、PackageA から PackegeB に対して、2 つの依存があることがわかる. 本ツールでも同様に確認することが可能である. PackageX から、PackageY も同様に検出できており、PackageY の内部で依存関係がある部分は検出されていない. このようにパッケージ間における依存を検出できていることが確認でき、Java 言語向けに拡張した friend の定義に基づいて検出することが可能であることがわかった.

この結果に基づいて実験 2 のソースコードを解析してパッケージ間の依存数を測定した. その結果が表 2 である. 表 2 の結果から、どのパッケージにも必ず外部に依存する部分が存在する. つまり、パッケージ間に対する依存が存在していることを示している. 従って、この外部の依存数の結果を表示するだけでは、どの依存関係が適切な関係であるか、不適切な関係であるかといったことが、ソースコードを読まなければわからない. また、パッケージによっては、その依存数が膨大である箇所があり、その部分を手作業で探すことは手間であるといえる. そこで、ソースコードの解析を実際に行い、パッケージ間のどのクラス間に依存があるかを示すことが必要であるといえる.

今回の実験では例として、org.apache.catalina.users のツールの出力結果を元に、パッケージ間の依存関係を表したものが図6である.

このように、どのパッケージ中のクラスが、どこのパッケージのクラスに依存関係があるかを出力することが可能であることがわかる．この結果を利用することで、パッケージ間における依存関係を明らかにすることが可能となり、ソースコードを読むことなく依存関係を把握することができ、どのクラスを再設計する必要があるということの情報を与えることができる.

表2　実験2における外部依存数

パッケージ名	外部依存数
org.apache.catalina.ant	7
org.apache.catalina.authenticator	27
org.apache.catalina.connector	48
org.apache.catalina.core	103
org.apache.catalina.deploy	1
org.apache.catalina.launcher	2
org.apache.catalina.loader	13
org.apache.catalina.mbeans	52
org.apache.catalina.realm	25
org.apache.catalina.security	5
org.apache.catalina.servlets	21
org.apache.catalina.session	24
org.apache.catalina.ssi	14
org.apache.catalina.startup	39
org.apache.catalina.users	8
org.apache.catalina.util	19
org.apache.catalina.valves	26

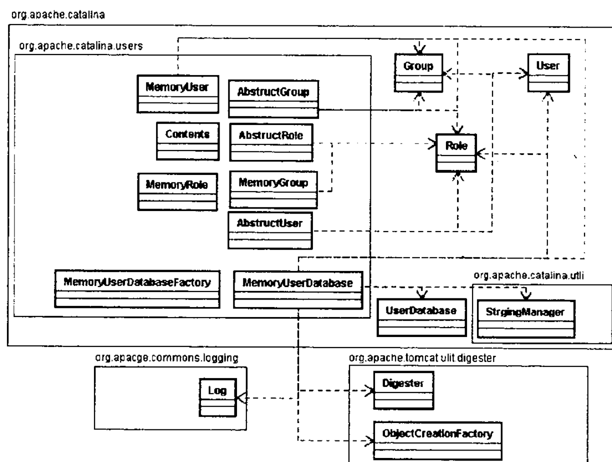

図6　users のパッケージ依存関係

6. まとめと今後の課題

我々はソフトウェア設計を考える上では、モジュール分割をきちんと行われていることが必要であり、その指標の1つであるモジュールの独立性について着目した．デメテルの法則は C++の時代に考案された物であったので、Java 言語向けに適用が可能となるように friend の定義の拡張を行った．さらに、デメテルの法則の適用単位をクラスではなく、パッケージとすることで、検出の複雑さを押さえ適用を容易にする方法を提案した.

この定義に基づいてパッケージ間の依存を発見するために Eclipse のプラグイン機能として実装を行い、パッケージ間の依存を実際に発見することが可能となった.

しかし、このツールを導入するに当たっては、パッケージ分割がある程度適切に行えるようにならないと利用することが不可能である．そのため、プログラミング初学者に対して適用する場合、パッケージ分割をどのようにする必要があるか事前に教えておかなければならない．また、依存が検出された場合、どのようにこの問題を解決しなければならないか事前に教えておかなければ、適切なソフトウェア設計にすることができない.

また、本研究ではパッケージ間についてデメテルの法則を適用したが、クラス間も同様にデメテルの法則を利用する場合、対象となるオブジェクトの数が多くなってしまうので、オブジェクト間のメッセージの役割を識別し、メッセージを削るルールを作成しないと、メッセージ関係を把握することができず、適切な結合度にすることが難しいと考えられる.

参考文献

[1] G.J Myers, “Composite / Structured Design,” Van Nostrand. Reinhold,1978.[Glenford J. Myers, 国友義久,伊藤武夫訳,編．ソフトウェアの複合/構造化設計 近代科学社,1979]

[2] Shyam R. Chidamber and Chris F. Kemerer, “A Metrics Suite for Object Oriented Design,” IEEE Transactions on Software Engineering, Vol.20, No.6, pp.476-493, June. 1994.

[3] "Eclipse Metrics Plugin," http://www.stateofflow.com/projects/16/eclipsemetrics (2007/4/18 現在)

[4] "Metrics plugin for Eclipse," http://metrics.sourceforge.net/ (2007/4/18 現在)

[5] K. Lieberherr , I. Holland , A. Riel, “Object-oriented programming: an objective sense of style,” Conference proceedings on Object-oriented programming systems, languages and applications, p.323-334, September 25-30, 1988, San Diego, California, United States

[6] Larman, C., “Applying UML and patterns: an introduction to object-oriented analysis and design,” Prentice Hall,1998. [依田光江訳、実践 UML: パターンによるオブジェクト指向開発ガイド,ピアソン・エデュケーション,1998]

[7] "Eclipse,"http://www.eclipse.org/ (2007/4/18 現在)

[8] Gamma, E., Helm, R., Johnson, R., and Vlissides,J., “Design Patterns,” Addison-Wesley, 1994.

[9] Apache Foundation, “ Apache Tomcat, ” http://tomcat.apache.org/ (2007/4/18 現在)