バージョン間の利用関係の変化を提示するシステムの試作

2009SE094 亀井雄佑　2009SE118 木下裕太郎　2009SE147 前原一幾
指導教員：横森励士

1　はじめに

ソフトウェアはさまざまな変化への対応や不具合の修正，機能追加の要求に対応するために，長期に渡って保守活動が行われているものが数多く存在する．このようなソフトウェアがどのように構成されているかを理解するためには，長年の保守活動でどのような変更が行われて来たかという情報が重要な情報として活用されている．さらに，直接の開発者だけでなく，開発の参加者全体が広く進捗情報を共有するために，最近の開発環境においては EPM などの開発データの自動収集・分析システムなどが活用されている．しかし，ソフトウェア開発における状況を的確に理解するためには，進歩情報だけでなく様々な観点からの分析結果の共有が必要となる．

本研究では，ソフトウェア部品間の利用関係に着目し，それらが保守活動を通じてどのように変化していったかを示すことを目的として作成したツールである，URV(Use Relation Viewer) を提案する．URV は，ソフトウェアの複数のバージョン間のファイルを分析して，利用関係の差分を取得し，その変化を示すツールである．保守活動が進行するにつれて，ソフトウェア部品の数と部品間の利用関係の数は増大し，複雑化していくので，それらの利用関係がどのように増加していくかの概要を理解することの重要度は増していくと考える．URV を活用することで変更が施された箇所を大きな粒度で把握でき，ソフトウェアの全体像の変化を把握しやすくなることを，URV を用いた実際のシステムへの適用例を題材に示す．

2　背景技術

2.1　ソフトウェア部品と利用関係

一般にソフトウェア部品とは，その内容をカプセル化したうえで，環境においてそれらを交換可能な形で実現したシステムモジュールの一部である [1]，[3]．本研究では，開発者が再利用を行う単位としてクラスをソフトウェア部品とみなす．ソフトウェアは複数のソフトウェア部品で構成されると考えることができ，継承，変数の宣言，インスタンスの作成，メソッドの呼び出し，フィールド参照など「ある部品が他の部品を利用する」「他の部品からその部品が呼び出される」などといった関係を定義することができる．

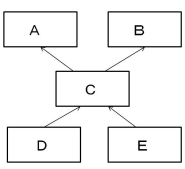

図 1　コンポーネントグラフ

本研究では，それらの関係をコンポーネントグラフとして表現する．コンポーネントグラフでは部品を頂点，部品間の利用関係を有向辺として定義する [4]．図 1 において部品 C が部品 A と部品 B を使用していて，部品 D と部品 E が，部品 C を使用していることを示している．

2.2　Classycle

Classycle とは，Java を対象としたクラスやパッケージ間の利用関係を分析するツールである．コンパイルされた後の class ファイル中の記述を対象に，各クラスがどのクラスを利用しているか，どのクラスに利用されているかという情報を取得し，XML ファイルで分析結果を出力する [2]．

XML ファイルは HTML ファイルに変換して出力される．変換された HTML ファイルは，表 1 のように表示される．指定のクラスがどのクラスを利用しているか，どのクラスに利用されているかという情報は，それぞれの数をクリックすることで表 2 のように表示される．

Classes and Packages

Click on ↻ or ↻ to go to the cycle to which the class/package belongs.
Click on ■ behind a number and a window will pop up showing more details.

Class/Package	Size	Used by	Uses internal	Uses external	Layer	Source (s)
P net.sf.freecol ↻	3	17 ■	13 ■	7 ■	1	targets
P net.sf.freecol.client ↻	13	8 ■	13 ■	9 ■	1	targets
C net.sf.freecol.client.ClientOptions ↻	8720	29 ■	16 ■	7 ■	4	targets
C net.sf.freecol.client.ClientOptions$1 ↻	1100	1 ■	2 ■	3 ■	4	targets
C net.sf.freecol.client.ClientOptions$2 ↻	1092	1 ■	2 ■	3 ■	4	targets
C net.sf.freecol.client.ClientOptions$3 ↻	1107	1 ■	2 ■	2 ■	4	targets
C net.sf.freecol.client.ClientOptions$4 ↻	1106	1 ■	2 ■	2 ■	4	targets
C net.sf.freecol.client.ClientOptions$5 ↻	1203	1 ■	3 ■	2 ■	4	targets
C net.sf.freecol.client.ClientOptions$6 ↻	2151	1 ■	7 ■	3 ■	4	targets

クラス名　利用されている個数　利用している個数

表 1　Freecol ver.0.7.2 の分析結果

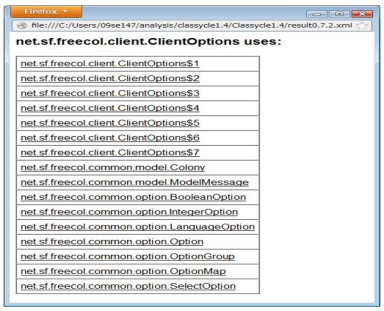

表 2　ClientOptions の利用先

2.3　EPM

EPM とはリアルタイムでのプロジェクト管理を目的とした開発データの自動収集・分析システムである．開発支援システムから開発履歴データを収集しプロジェクト管理を行うために有益な分析結果をユーザに提供する [5]．ソフトウェア開発で広く使用されている，構成管理シス

テム，メール管理システム，障害管理システムからデータを収集し，格納されたプロセスデータをユーザ（管理者・開発者）の要求に応じて分析結果を表示し，結果をグラフや表として表示することができる．これにより直接の開発者だけでなく，開発の参加者全体が広く進捗情報を共有するのに役立つ．

3　研究について

3.1　既存のツールの問題点

EPM では開発履歴データを収集しているが，実際に活用しているのは LOC などの限られた情報で，ソフトウェア部品間の関係の変化などには着目していない．これらの情報は開発を通じて複雑化しており，グラフや図を利用して把握することでシステムをより理解しやすくできると考えられるので，これらの関係の変化を分析して表示する仕組みが必要である．

3.2　研究の目的

本研究では，ソフトウェア部品間の利用関係の変化に着目して，それらがどのように変化していったかを調査して表示するツール URV(Use Relation Viewer) を提案する．長期に渡る保守活動で，ソフトウェア部品の数は増大し，それに伴い利用関係の数も増大し複雑化する．構造が複雑化したソフトウェアを理解することは容易ではなく，どの部分の利用関係が複雑で，またどの部分が変更にコストがかかるか，という情報を提供することは重要であると考えられる．本ツールを用いることで，バージョン間のソフトウェア部品間の利用関係がどのように変化したかという情報が視覚的に理解でき，開発者がそれらの情報を共有し，多くの参加者がソフトウェア全体像の変化を把握しやすくなると考えられる．

4　ツールの説明

4.1　ツールの構成

複数あるバージョンのそれぞれのバージョンを Classycle によって分析した結果をもとに，それらの情報をまとめて読み取り，利用関係の変化を分析する．分析したデータを元に，利用関係の変化をいくつかの機能を用いて提示する．

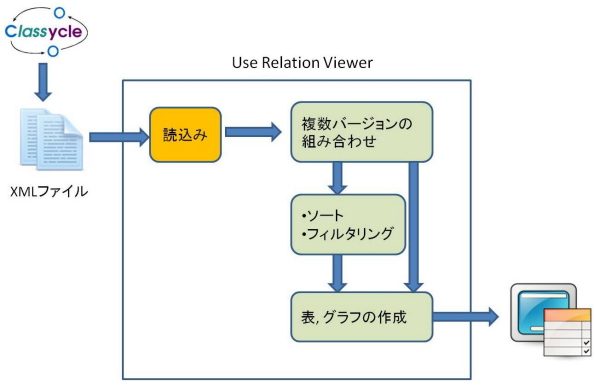

図 2　URV の構成

4.2　入力部

複数のバージョンを持つソフトウェアに対して，それぞれのバージョンを Classycle によって分析した結果として生成される XML ファイルを XML パーサーを用い 1 つずつ読み取る．

```
<classes[numberOfExternalClasses]>
 <class[name][sources][type][innerClass][size][usedBy]
        [usesInternal][usesExternal][layer][cycle]>
   <classRef[name][type]/>
   :
 </class>
   :
</classes>
```

図 3　classes タグの構成

XML ファイルに記述されている利用関係についての情報が格納されている class タグを分析する．図 3 のように，classes タグにはソフトウェア内に存在する class ファイルが class タグによって示されている．class タグでは，クラス名が属性 name に格納されており，利用関係についての情報として，他のクラスに利用されている個数が属性 usedBy に格納されており．他のクラスを利用している個数が属性 usesInternal に，ソフトウェア外部のクラスを利用しているクラスの個数が属性 usesExternal に格納されている．またそれぞれのクラスがどのクラスを利用しているのか，利用されているのかという情報が，classRef タグで示される．

入力部ではこれらの情報を読み取り，利用関係の変化を分析する．

4.3　出力のレイアウト

4.3.1　表の表示

それぞれのバージョンに対応する XML ファイルを複数指定し，表示したい要素を選択することで，クラスの利用関係の表が表示される．この表では，クラス単位で利用関係の個数が表示されるが，そのバージョンに存在しないクラスは，空白として表示される．ソートをして指定したバージョンの要素の数を昇順，降順に並び替えることができ，クラス名を名前順に並び替えることもできる．また，分析した表から見つけたいクラス名を指定し検索して，表 7 のように抽出することもできる．

4.3.2　グラフの表示

図 4 のように，指定した個数のクラスの利用関係の変化を，全バージョンについて折れ線グラフを用いて視覚的に提示できる．

4.3.3　利用関係の差分表示機能

2 つのバージョンを指定し，各クラスの利用関係の個数の差分を表 4 のように表示できる．数が増えたクラスは正の数，減ったクラスは負の数で表示されるが，絶対値で表示することもできる．

4.3.4　利用先，利用元表示機能

クラスを 1 つ指定し，クラスのバージョン間での利用関係を, クラス単位で表示できる．表 5 のように，利用関係が存在する場合には「●」を表示し，そうでない場合には空白にする．

4.3.5　フレームワーク利用 (外部利用関係) の分析機能

XML ファイル内の class タグの属性 usesExternal では，外部のクラスを利用している個数が格納されており，classRef タグにより利用先のクラスが格納されているので，これらを用いてクラスが利用しているフレームワークを検索することができる．利用先のパッケージ名を指定し検索して，クラス単位で何個利用しているかを表示する．

5　URV の適用例

適用例を通じて，実際のソフトウェアを対象にした分析について説明する．

実際に Nutch という Web 検索ソフトウェアプロジェクトについて，バージョン 0.8，0.9，1.0，1.1，1.2，1.3 の利用関係の変化を調べる．ツール上ではそれぞれ ver.1 から ver.6 に対応する．

ClassName	ver.1	ver.2	ver.3	ver.4	ver.5	ver.6
org.apache.nutch.util.NutchConfiguration	50	54	80	57	82	57
org.apache.nutch.crawl.CrawlDatum	46	60	80	84	87	84
org.apache.nutch.parse.ParseData	44	47	53	45	58	45
org.apache.nutch.protocol.Content	44	46	52	50	58	50
org.apache.nutch.parse.Parse	44	45	48	46	52	46
org.apache.nutch.metadata.Metadata	39	41	51	46	55	46
:	:	:	:	:	:	:

表 3　被利用クラス数を降順で表したクラスの表

まずは，各クラスが利用されている関係を調べる．ソート機能を利用し，バージョン 0.8 で最も利用されているクラスに着目する．表 3 のように，このバージョンでは NutchConfiguration クラスが一番利用されている．

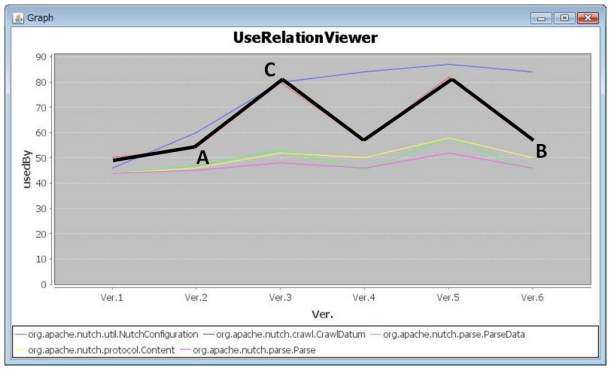

図 4　表 3 の上位 5 つのクラスの変化を表したグラフ

また，グラフを利用してバージョン間の変化の大きいクラスを視覚的に発見する．図 4 のように，NutchConfiguration クラスは，バージョン 0.9(図 4 中の A) から 1.3(図 4 中の B) の間で大きな変動がみられる．バージョンが更新される度に，増減を繰り返している．

ClassName	ver.1	ver.2	Different
org.apache.nutch.util.NutchConfiguration	80	54	26
org.apache.nutch.crawl.CrawlDatum	80	60	20
org.apache.nutch.util.NutchJob	32	17	15
org.apache.nutch.metadata.Metadata	51	41	10
org.apache.nutch.searcher.Hit	18	8	10
org.apache.nutch.searcher.HitDetails	28	18	10
:	:	:	:

表 4　バージョン 0.9 と 1.0 の差分を表した表

実際に差分ソート機能を使って，バージョン 0.9(図 4 中の A) と 1.0(図 4 中の C) の差を調べる．表 4 のように，やはり NutchConfiguration クラスが一番増加していて，26 個増加していた．

ClassName	ver.1	ver.2	ver.3	ver.4	ver.5	ver.6
org.apache.nutch.fetcher.Fetcher2		●				
org.apache.nutch.net.urlnormalizer.regex.		●	●	●	●	●
org.apache.nutch.crawl.AdaptiveFetchSch			●	●	●	●
org.apache.nutch.indexer.field.AnchorField			●		●	
org.apache.nutch.indexer.field.BasicFields			●		●	
org.apache.nutch.indexer.field.CustomFiel			●		●	
:	:	:		:	:	:

表 5　利用元クラスの有無を表した表

実際に何が利用されるようになって，何が利用されなくなったかは利用先，利用元表示機能で確認することができる．表 5 から，Fetcher2 クラスのように利用しなくなったクラスも存在することが確認できる．

ClassName	ver.1	ver.2	ver.3	ver.4	ver.5	ver.6
org.apache.nutch.segment.SegmentReade	55	54	57	58	57	58
org.apache.nutch.crawl.CrawlDbReader	39	41	46	46	46	46
org.apache.nutch.searcher.OpenSearchS	38	39	39		39	
org.apache.nutch.crawl.LinkDb	37	38	39	41	41	41
org.apache.nutch.searcher.DistributedSea	36	37				
org.apache.nutch.parse.js.JSParseFilter	36	36	36	36	36	36
:	:	:	:	:	:	:

表 6　利用クラス数を降順で表したクラスの表

次は，NutchConfiguration クラスが外部のクラスを利用している関係を調べる．利用されている側を分析した場合と同じようにソートすると，表 6 のように NutchConfiguration クラスは上位には出現せず，利用する関係はあまり多くないことが分かる．

ClassName	ver.1	ver.2	ver.3	ver.4	ver.5	ver.6
org.apache.nutch.util.NutchConfiguration	10	7	7	9	7	9

表 7　NutchConfiguration クラスの検索結果

特定のクラスに絞った調査をする時に，表 7 のようにクラス名の検索をして，NutchConfiguration クラスを瞬

時に見つけ出すことができる．表 7 から利用されている関係では大きく変動があったが，利用する関係ではあまり変動がないことがわかる．

ClassName	ver.1	ver.2	ver.3	ver.4	ver.5	ver.6
org.apache.nutch.util.NutchConfiguration	2	2	2	1	2	1
org.apache.nutch.tools.DmozParser$XML	0	0	0	0	0	0
org.apache.nutch.segment.SegmentMerge	6	9	12	10	10	10
org.apache.nutch.html.Entities	0	0	0		0	
org.apache.nutch.fetcher.Fetcher	9	11	13	16	16	16
org.apache.nutch.ontology.Ontology	0	0	0		0	
:	:	:	:	:	:	:

表 8　フレームワークの検索結果

また，ソフトウェア外部のクラスを利用しているクラスの中には，フレームワークを利用しているクラスも存在する．Nutch は Hadoop というフレームワークを多く利用していて，実際どのクラスが何個利用しているのかは，表 8 のようにフレームワーク利用の分析機能を使って確認することができる．この結果から NutchCinfiguration クラスはどのバージョンでも Hadoop を利用していることが確認できる．

6　考察

6.1　プロトタイプからの機能追加について

本研究においては，プロトタイプを作成した後で，実際に URV を用いて分析を行ってもらった．その中であがってきた要望として，

- クラス名を名前順でソートする機能
- XML ファイルを開いた時，前回開いたフォルダを最初に表示する機能
- フレームワークで検索する機能

があがった．特定のフレームワークとアプリケーションの間の関係を調べたいときに，バージョンごとにアプリケーションのクラスの利用関係を見て，そのフレームワークを使用しているかを調べる必要があるので，フレームワークの分析を瞬時に行う機能を要望された．これらの意見を本研究で作成したツールに取り入れ，機能の向上に努めた．また，完成したツールを実際に試用してもらい，とても使いやすいという意見をもらった．

6.2　ツールの有用性

本研究では，実際に利用関係の大きいところに着目し，フレームワークを調査する分析を想定してツールを作成した．想定する分析手順として，まずソフトウェアのソースコードをコンパイルし，Classycle で XML ファイルを生成する．次に，クラスの外部利用先を調査する．次に，対象のフレームワークを利用しているクラスを絞り込む．次に，全バージョンについて調べ，利用関係が大きい順に並び替える．次に，調査対象のクラスを指定し，前後のバージョンの変化を調査する．最後に，原因をソースコードから調べる，という方法を考える．

URV では，想定する分析手順に対してクラスの外部利用先の調査から調査対象のクラスを指定し，前後のバージョンの変化を調査などの手間がかかる作業を自動で行う．大まかな分析結果を示すことができる一方で，実際に細かい分析を行うためにどこを注目すればよいかの提示を行うことができ，分析に役立つと考えられる．

6.3　今後の機能拡張

バージョンの前後でソフトウェア部品名（ここではクラス名）の変更があった場合でも，中身がほぼ一致している部品を同一の部品とみなし，追いかけられるようにする機能が考えられる．これらの機能があると，途中の開発方針の変更によってクラス配置の方針変更があった場合でもそれらを柔軟に取り入れることができると考えられる．ただし，クラス配置の方針変更により，クラスの役割自体も変わる場合も考えられ，機能自体が必要かどうかを検討する必要がある．

今後，多くの事例に実際に適用し，利用関係がなくなっている部分を解析し，それらに後継とすべき部品があるかどうかを調査し，どのような条件で後継部品を決定すべきかを調査する必要があると考えられる．

7　まとめ

本研究ではソフトウェア部品間の利用関係に着目し，開発を通じてどのように変化していくかを示すツールを試作した．本ツールにより，これらのようなバージョン間のソフトウェア部品間の利用関係がどのように変化したかという情報が視覚的に理解でき，それらの情報を共有することを通じて，多くの参加者がソフトウェア全体像の変化を把握しやすくなったと考える．

参考文献

[1] C.Krueger, “*Software Reuse*, ” ACM Computing Surveys, vol. 24, no. 2, pp. 131-183, 1992.

[2] Franz-Josef Elmer, “*Classycle:Analysing Tools for Java Class and Package Dependencies*, ” http://classycle.sourceforge.net/, 2012.

[3] I.Jacobson, M.Griss, and P.Jonsson,“*Software Reuse*, ” Addison-Wesley Professional, New York, 1997.

[4] K.Inoue, R.Yokomori, T.Yamamoto, M.Matsushita, and S,Kusumoto, “*Ranking Significance of Software ComponentsBased on Use Relations*, ” IEEE Transactions on Software EngineeringConference, vol. 31, no. 3, pp. 213-225, 2005.

[5] 大平雅雄，横森励士，阪井誠，岩村聡，小野英治，新海平，横川智教 “ソフトウェア開発プロジェクトのリアルタイム管理を目的とした支援システム, ” 電子情報通信学会論文誌, vol. J88-D-I, no. 2, pp. 228-239, 2005.