セマンティック Web 技術を用いた Java ソフトウェアの構造検索

長谷川 明史　　塚本 享治
東京工科大学　メディア学部　メディア学科

1.はじめに

現在オープンソースソフトウェアが利用される機会が増え、ソースコードを読むことが増えている。しかし、利用者は大量のプログラムから目的の場所を見つけなければならない。

このとき既存の文字列検索、ソースコードブラウザなどではプログラムの構造検索ができない。

そこで、セマンティック Web の技術を用いたクラスの構造やクラスの関係に基づく Java ソフトウェアの構造検索方法を提案する。

2.アプローチ

2.1. ソフトウェアの構造

Java ソフトウェアの構造について述べる。Java のソフトウェアは Java クラスからできており、フィールドやメソッドを持つ。このことは表 1 のような階層構造と考えることができる。

表 1　Java クラス構造のツリー

クラス																									
メソッド															フィールド				Jarファイル	パッケージ	修飾子	アクセス修飾子	インタフェース	スーパークラス	クラス名
処理									修飾子	アクセス修飾子	Throwされるクラス	返り値の型	引数	メソッド名	修飾子	アクセス修飾子	変数の型	変数名							
例外のキャッチ	例外のスロー	クラスのインスタンス化	クラスメソッド起動	インスタンスメソッド起動	クラス変数の取得	クラス変数の代入	インスタンス変数の取得	インスタンス変数の代入																	

この Java クラスはそれぞれ別のクラスと関係を持つ。この関係は大きく、is-a 関係、has-a 関係、依存関係の 3 つに分類できる。is-a 関係は継承や実装の関係、has-a 関係は他のクラスの参照変数を持つ。依存関係はそれ以外の、実行時に生じる一時的な関係である。

2.2. クラスの解析方法

Java ソフトウェアの構造を検索できるようにするため、そのソフトウェアのクラスの構造とクラス関係に対して解析を行う。このクラスの解析対象として Java クラスファイルを用いる。[2]が構文解析対象にしているソースファイルとは異なり、Java クラスとクラスファイルが 1 対 1 対応している。また、確かに動くという保証もある。

[1]の研究を参考に、ソフトウェアに含まれるクラスファイルを取り出し、javap を用いて逆アセンブルを行う。逆アセンブルされたコードには、スーパークラスやインタフェース、フィールドなど表 1 に挙げたクラスの構造がすべて含まれている。

3.RDF による構造表現

3.1. クラスの RDF 表現

RDF はトリプルと呼ばれる主語、述語、目的語の 3 つ組を単位に作られ、ラベル付き有向グラフ構造としてあらわすことができる。"ClassA is-a ClassB"と記述することで、この 2 つのクラスの間に is-a 関係があるということを表す。

RDF グラフのパターンマッチによる構造検索ができるように、解析結果を RDF で表現する。たとえば図 1 において、「Interface と is-a 関係にあり①、ClassB に依存し②、ClassD と関連のある③クラス」というパターンに対して、ClassA がこの検索結果として現れる。

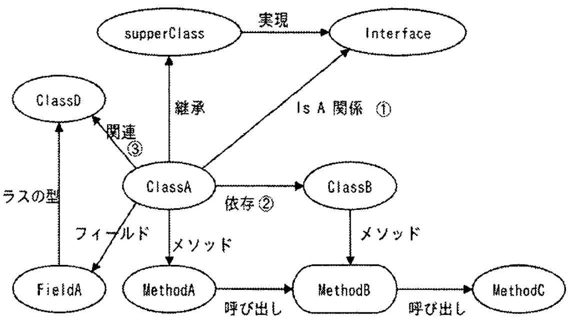

図 1　RDF のグラフ構造

3.2. OWL を用いた推論処理

クラス構造の RDF 化を行った後、OWL を使った推論を行う。推論を行うことで、すでにある RDF に対して新しいトリプルを増やす。

is-a 関係などは推移的であり、" ClassA is-a ClassB. ClassB is-a ClassC" の時に" ClassA is-a ClassC" でなければならない。しかし、クラスの解析を行った場合、直接の継承や実装しか見えない。OWL 推論を行うことで ClassA と ClassC に is-a 関係があるというトリプルを補うことができる。

4.SPARQL の検索実験

4.1. 検索システムの構築

本研究で提案する手法を実現するため、次の手順で構造検索を行う。

1. Java のソフトウェアから Java のクラスファイルを収集する
2. クラスファイルを逆アセンブルする
3. 逆アセンブルした Java のクラスから、クラスの構造を RDF 化する
4. OWL による推論を行い、Java クラスの RDF グラフ情報を付加する
5. 解析したソフトウェアの調べたい構造を RDF グラフから SPARQL を用いて検索する

Search for Java Software Structure by Semantic Web Technology.
Akifumi Hasegawa, Michiharu Tsukamoto
Tokyo University of Technology School of Media Science

4.2. クラス間の関係検索

リスト1の SPARQL はメインメソッドを持つクラスと関係の無いクラスを求めるクエリである。すべてのクラスから、メインクラスと何らかの関係を持つクラスの差がこのクエリの結果になる。

リスト1　使われないクラスを検索する SPQRQL

```
1 : SELECT ?o
2 : WHERE{
3 :     ?o rdf:type cl:ClassType.
4 :     OPTIONAL{
5 :         :org.h2.tools.Console cl:relateToTransitively ?o.
6 :         LET ( ?z := true).
7 :     }
8 :     FILTER (!BOUND(?z)).
9 : }
```

このクエリによって得られる検索結果は表2のようになった。ソフトウェアのクラスファイルの中からこれらのクラスを削除し、このソフトウェアを起動したところ、問題なく起動できることを確認した。

表2　使われないクラス

o
org.h2.fulltext.FullText
org.h2.fulltext.FullTextLucene
org.h2.fulltext.FullTextSettings
org.h2.fulltext.IndexInfo
org.h2.jdbcx.JdbcConnectionPool
org.h2.jdbcx.JdbcConnectionPool%24PoolConnectionEventListener
org.h2.jdbcx.JdbcDataSource
org.h2.jdbcx.JdbcDataSourceFactory
org.h2.jdbcx.JdbcXAConnection
org.h2.jdbcx.JdbcXid
org.h2.message.TraceWriterAdapter
⋮

4.3. ソフトウェアの構造検索

リスト2の SPARQL は GoF の AbstractFactory パターンのクラス構造を求めるためのクエリである。抽象ファクトリクラスで生成するインタフェースを、具象ファクトリクラスで生成するインスタンスを決めている。図2のような構造をソフトウェアの中から探すことになる。この検索結果は表3である。

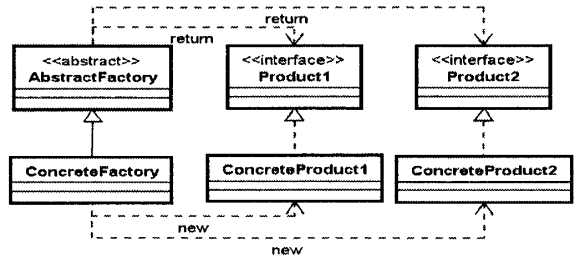

図2　AbstractFactory の構造

リスト2　AbstractFactory の構造検索クエリ

```
SELECT DISTINCT ?AbstractFactory
WHERE{
    ?AbstractFactory rdf:type cl:AbstractClass.
    ?ConcreteFactory rdf:type cl:Class;
        cl:isA ?AbstractFactory.

    ?AbstractFactory cl:return ?Product1;
        cl:return ?Product2.
    ?Product1 rdf:type cl:Interface.
    ?Product2 rdf:type cl:Interface.

    FILTER (?Product1 != ?Product2).
    FILTER (?AbstractFactory != ?Product1).
    FILTER (?AbstractFactory != ?Product2).

    ?ConcreteFactory cl:return ?Product1;
        cl:return ?Product2.
    ?ConcreteProduct1 cl:isA ?Product1.
    ?ConcreteProduct2 cl:isA ?Product2.
    ?ConcreteFactory cl:new ?ConcreteProduct1, ?ConcreteProduct2.

    FILTER (?ConcreteFactory != ?ConcreteProduct1).
    FILTER (?ConcreteFactory != ?ConcreteProduct2).

    OPTIONAL{
        {  ?Product1 cl:isA ?Product2. }
        UNION{ ?Product2 cl:isA ?Product1. }
        LET (?x := true).
    }
    FILTER (!BOUND(?x)).
    FILTER (!regex(str(?Product1), "http://soft/jdk")).
    FILTER (!regex(str(?Product2), "http://soft/jdk")).
}
```

表3　AbstractFactory

AbstractFactory
:org.jboss.classloading.spi.metadata.ClassLoadingMetaDataFactory
:org.jboss.metadata.plugins.loader.AbstractMetaDataLoader
:org.jboss.deployers.spi.structure.StructureMetaDataFactory
:org.jboss.beans.metadata.spi.builder.BeanMetaDataBuilder
:org.jboss.kernel.plugins.annotations.InjectableMemberAnnotationPlugin
:org.jboss.metadata.plugins.loader.AbstractMetaDataLoader
:org.jboss.metadata.plugins.loader.BasicMetaDataLoader
:org.jboss.managed.api.factory.ManagedObjectFactory
:org.jboss.metadata.plugins.loader.AbstractMetaDataLoader
:org.jboss.config.plugins.AbstractConfiguration

4.4. RDF のデータ量と処理時間

表4は HSQLEB、H2、Tomcat、JBoss という4つのソフトウェアに対して、RDF 化と推論を行った際の、トリプルの数と処理時間である。推論を速く終わらせるため、推論対象をしぼって推論を行った。

表4　トリプル数と推論時間

ソフトウェア	HSQLDB 1.8.0	h2 1.0.73	Tomcat 6.0.16	JBoss 5.0.0.CR2
クラス	300個	422個	2,009個	8,927個
メソッド	4,413個	6,335個	23,761個	72,444個
is-a関係	544個	946個	4,906個	23,375個
has-a関係	632個	864個	3,706個	11,638個
依存関係	3,044個	5,197個	18,057個	65,320個
解析トリプル	111,703組	148,627組	554,694組	1,497,106組
推論対象トリプル	6,314組	9,727組	36,773組	132,512組
推論増加トリプル	58,881組	162,133組	461,829組	2,761,122組
全トリプル	170,584組	310,760組	1,016,523組	4,258,228組
解析時間	1分10秒	1分45秒	8分23秒	30分30秒
推論時間	1分5秒	3分32秒	37分	4684分

5.考察

いくつかの検索を行い、セマンティック Web 技術を用いた構造検索が確かに行えることが確認できたが、検索が困難な場合もいくつか存在することが分かった。

たとえば、リフレクションや DI によって実行時にクラスが決まるような関係性はこの静的な構造検索では検索できない。ほかにも、クラスファイルを解析対象にしたことで、コンパイル時に定数参照が値に置き代わり関係が失われ、実装方法によって検索に漏れてしまう場合もある。これら問題は、クラスファイル以外のソースや UML などを解析対象に加えていくことで解決できると考えられる。

表4の推論対象トリプルの数と推論にかかる時間をグラフにプロットしたところ、推論トリプル数に対して推論にかかる時間はおおむね n^2 オーダで増えているとみなすことができる。実用的な検索にするためには、推論を高速に行わなければならない。推論エンジンや推論ルールを変える、推論対象のデータの範囲を狭める、推論をかけるタイミングを変えるなどの解決策が考えられる。それぞれの方法でどれだけ効果があるか検証し、推論時間の短縮を行わなければならない。

6.おわりに

構造解析対象を増やす必要とともに、セマンティック Web の推論ルールや検索 SPARQL を工夫することで、より実用的な検索を行う必要がある。

参考文献

[1] 姫路, 塚本, SPARQL による Java パッケージの依存性解析, 第70回全国大会論文, 2008

[2] 丸山, 山本, 亀井, 吉原, 再利用性の高い Java-XML リポジトリを用いた次世代ソフトウェア変更環境の開発, 次世代ソフトウェア開発事業, 2002