SPARQL による Java パッケージの依存性解析

姫路　明宏　　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

オープンソースを利用する機会が増えている。Java を用いたオープンソースでは、非常に多くの他のオープンソースを利用しているために様々な、トラブルに遭遇する。このような時に、Jar ファイル中の複数のクラスの依存性や関係を知りたい場面がある。

本稿では、複数のパッケージやクラス・メソッド同士の依存性の解析を行い、機械的に検索を行うことで、クラスの依存関係を問い合わせするシステムを作成したので報告する。

2. 依存性解析の方法

2.1. 扱う依存性の定義

本稿では、特に Jar ファイルを対象とした依存性解析を行う。

依存性とは自オブジェクトが目的のオブジェクトを一時的に使うということである。本稿では、オブジェクト指向のクラスの関連の種類である。集約、汎化、委譲、実現を依存性として扱う。

2.2. Jar ファイルの依存情報の抽出

依存性の解析に必要な、クラス情報の抽出は Javap による逆アセンブルすることで行う。逆アセンブルされたコードでは、クラスが使用しているメソッドがわかる。また、継承するクラスについても、見つけることができる。この形は、決まった形式で出力されるため、機械的に依存性の情報を抽出するために適している。その形式から Java の処理によって依存性を示すコードを抜き出す。抜き出した情報を RDF[1]という記述形式を使うことで、機械的処理可能な関係性をつけて記述した。

2.3. 依存性情報の検索方法

RDF[1]で記述した依存方情報を効率的に検索するために SPARQL[3]を用いた。SPARQL は W3C 草案の RDF データを操作するための構文的に SQL とよく似たクエリ言語である。また、SPARQL を利用するにあたって、セマンティック WEB アプリケーション開発を目的として作られた Java 用のフレームワークである Jena を使用した

Dependent analysis of JavaPackage by SPARQL
Akihiro Himeji , Michiharu Tsukamoto
School of Media Science, Tokyo University of Technology

3. 依存性解析システムの実現

3.1. 依存性解析システムの全体像

2 で述べた手法を図 1 のような方法で実現した

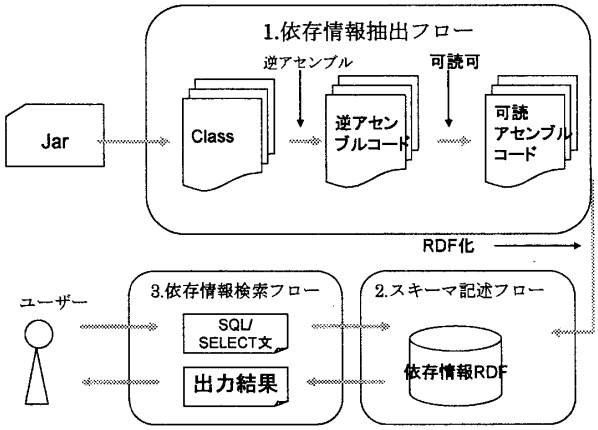

図 1 ：全体像

2.2 で述べた方法によって抽出したクラス情報の内容を解析する。スーパークラスによる依存、インターフェースによる依存、メソッドによる依存などの、情報が抽出し、その情報を RDF 化する。利用者は、自分のほしい情報を満たす SPARQL クエリを発行する。依存情報が記述された RDF から、情報を検索することができる。以下にその詳細を述べる。

3.2. 依存性の抽出

クラスは逆アセンブルされると図 2 のようなコードになる。

```
class ExClass extends ClassA implements InterfaceA{        ①
ExClass()
                    <calc 内の処理>
public int calc(int, int);
10:   iload_2
11:   invokevirtual   #4; //Method sum:(II)I        ②
14:   istore          4
16:   aload_3
17:   iload           4
19:   invokevirtual   #5; //Method ClassB.judgment:(I)V
22:   iload           4
24:   iconst_2
}
```

図 2 ：逆アセンブルされたクラス

①では、ソースコードと同じように extends 節によって、ExClass と ClassA が継承関係にあることがわかる。また、実現の関係に ExClass と InterfaceA が実現の関係にあることもわかる。

また、②#Method ClassB.judgment()V の記述部分によって、メソッド calc(int ,int)内で ClassB の judgment()のメソッドが起動され、ClassB と委譲関

係にあることがわかる。これらの、依存性情報をJavaの処理によって抽出を行う。

3.3. 依存性情報のRDF化

依存性情報の抽出を行うと同時にRDF化する。クラス同士の関係を「メソッドA」は「クラスB」と「○○の関係にある」というRDFデータモデルに沿った形によってクラス間の関係を記述することとした。

この関係を用いて、Javaの処理によって、抽出したクラスとクラス同士の関係の表現のために図4のようなOWL[2]を記述した。

```
<owl:ObjectProperty rdf:about="#依存する">
  <rdfs:domain rdf:resource="#メソッド"/>
  <rdfs:range rdf:resource="#メソッド"/>
  <owl:inverseOf rdf:resource="#依存される"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="継承する">
  <owl:inverseOf>
  <owl:ObjectProperty rdf:ID="継承される"/>
  </owl:inverseOf>
  <rdfs:range rdf:resource="#クラス"/>
  <rdfs:domain rdf:resource="#クラス"/>
</owl:ObjectProperty>
```

図4：RDF記述による関係性の定義

RDFによって、クラス同士の関係性をつなぐと図5のように出力される。

```
<クラス rdf:ID="ExClass">
 <クラス名 rdf:datatype="http://www.w3.org/#string"> ExClass</クラス名>
  <継承する>
   <クラス rdf:ID="ClassA">
    <クラス名 rdf:datatype="http://www.w3.org/#string"> ClassA</クラス名>
    <継承される rdf:resource="#ExClass"/>
   </クラス>
  </継承する>
 <含む>
  <メソッド rdf:ID="int calc(int, int)">
  <メソッド名 rdf:datatype="http://www.w3.org/#string"> int calc(int, int)</メ
ソッド名>
  <含まれる rdf:resource="#ExClass"/>
  <依存する>
   <メソッド rdf:ID="void_judgment-int-">
    <メソッド名 rdf:datatype="http://www.w3.org/#string"> void
udgment(int)</メソッド名>
  <依存される rdf:resource="#int_calc-int_int-"/>
```

図5：生成されるRDFの例

3.4. SPARQL文

生成したRDFからは、SPARQLによって情報の検索を行った。SPARQLによって、特定のメソッドの依存範囲や、複数のクラスについての情報の、多目的な検索を一度のクエリで行えるようになった。図5は、SPARQL文とその出力結果である。

```
PREFIX java: <http://www.owl-
ontologies.com/Ontology1198736702.owl#>
SELECT ?x ?含まれるクラス ?依存するメソッド
WHERE {
?x java:依存する ?y.
FILTER REGEX(str(?y),"void_io-java.io.InputStream-")
?x java:含まれる ?含まれるクラス .
?x java:依存する ?依存するメソッド
}
```

メソッド	含まれるクラス	依存するメソッド
java.lang.String_doBsh-java.lang.String	bsh.Remote	java.lang.String_substring-int-
java.lang.String_doBsh-java.lang.String	bsh.Remote	int_indexOf-java.lang.String-
java.lang.String_doBsh-java.lang.String	bsh.Remote	java.lang.String_substring-int_int-
java.lang.String_doBsh-java.lang.String	bsh.Remote	int_length--

図5：SPARQL文と出力結果

4. 試用実験と考察

本システムを利用してTomcat6.0.14のJarファイルの解析を行った。

本システムによって解析を行いRDF化されたTomcatの依存情報から、javax.servlet.http.HttpServletを継承する複数のクラスの中から、ある任意の一つのメソッドを用いているクラスを検索した。その検索のために図6のようなSPARQLクエリ文を発行した。

```
PREFIX java:<http://www.owl-
ontologies.com/Ontology1198736702.owl#>
SELECT ?結果
WHERE {
?結果 java:継承する ?y .
FILTER REGEX(str(?y),"javax.servlet.http.HttpServlet")
?依存されるメソッド java:依存する ?z .
FILTER REGEX(str(?z), "void_InitialContext--")
?結果 java:含む ?依存されるメソッド
}
```

図6：実験SPARQL

その結果として、図7のような結果を得ることができた。

結果
org.apache.catalina.ssi.SSIServlet
org.apache.catalina.servlets.DefaultServlet
org.apache.catalina.servlets.InvokerServlet

図7：SPARQLによる検索結果

これによって、複数の関係性を条件にした、検索が可能であるということがわかった。

5. おわりに

本稿では、Javaを対象に依存性を解析し、セマンティックWebの技術を用いて、ある程度の量のJarファイルの関係を検索することのできるシステムを生成することができた。

しかし、Jbossなど大量のJarファイルを解析することができなかった。これは、システムがメモリ上だけで情報操作を行うためであった。これを解決するためには大量のRDFを検索可能にするシステムの開発が必要である。

参考文献

[1]. W3C "Resource Description Framework", http://www.w3.org/RDF/, 2006

[2]. W3C "OWL Web Ontology Language" http://www.w3.org/TR/owl-features/, 2004

[3]. W3C "SPARQL Query Language for RDF" http://www.w3.org/TR/rdf-sparql-query/, 2006