# UMLで記述されたソフトウェアのRDFグラフへの変換

長谷川　明史　　塚本　享治
東京工科大学大学院　バイオ・情報メディア研究科

## 1. はじめに

UML(Unified Modeling Language)はソフトウェアの設計図として用いられている。このUMLをセマンティックWebのデータ構造であるRDFに変換することによって、ソフトウェア開発の場面にセマンティックWeb技術を持ちこみ、活用することができる。

本稿では、実際にソフトウェアの静的構造を表すクラス図をRDFグラフに変換する。これによって、RDF検索クエリ言語のSPARQLを用いてクラス図の検索を行った。

## 2. アプローチ

### 2.1 クラス図

UMLにはユースケース図、アクティビティ図などの様々な図があり、目的によって使い分けられる。これらの図の中で、本稿ではソフトウェアの静的な構造を表すクラス図をRDFへの変換対象とする。

クラス図は、名前や属性といったそれぞれのクラスが持つ情報と、関連や実現といったクラス間に表現される情報の2つがある。たとえば、図1のクラス図の中には、3つのクラスが存在している。クラスAは"名前"という属性と"名前を取得する"という操作を持っている。また、クラスAとインターフェースの間に集約関係が、クラスBとインターフェースの間には実現関係があることが読みとれる。

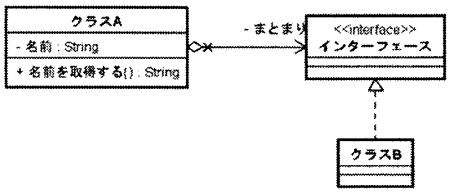


図1 クラス図の例

### 2.2 クラス図からの情報の取得方法

クラス図をRDF化するためには、まずクラス図を作成し、作成したクラス図からクラスやクラス間の関係の情報を取得しなければならない。このクラス図の作成と情報の取得は、UMLモデリングツールastah* professional[1]を用いて次の手順で行う。

1. モデリングツール上でクラス図を作成する
2. 作成したクラス図をプロジェクトとしてファイルに保存する
3. asthar* professionalのJava APIを通して、プロジェクトファイルからモデルの情報を取得する

## 3. ソフトウェアのRDF化

### 3.1 ノードを識別するためのURI

RDFは主語、述語、目的語という3つ組を単位に作られたラベル付き有向グラフの構造を持っている。RDF中の各ノードはURIによって一意に識別されるため、URIが重複を防がなければならない。名前をそのままURIに用いると、同じ名前の要素がクラス図中に存在していた場合、同じノードを表すことになってしまう。このような事態を避けるため、ノードに割り当てるURIには、モデリングツール側中のID値を利用する。

まず、ソフトウェアごとにURIを決める。このソフトウェアのURIの後にモデリングツール側で割り振られたIDをつけることで、URIの重複を防ぐことができる。

### 3.2 クラス図とRDFグラフの対比

図2は図1のクラス図から変換したRDFグラフを図示したものである。図中の楕円はそれぞれ一意なノードを表し、楕円の中にはrdf:typeで明示されるノードの型を示した。ラベル付きの矢印は述語を表し、主語側のノードから目的語側のノードやリテラルを指す。

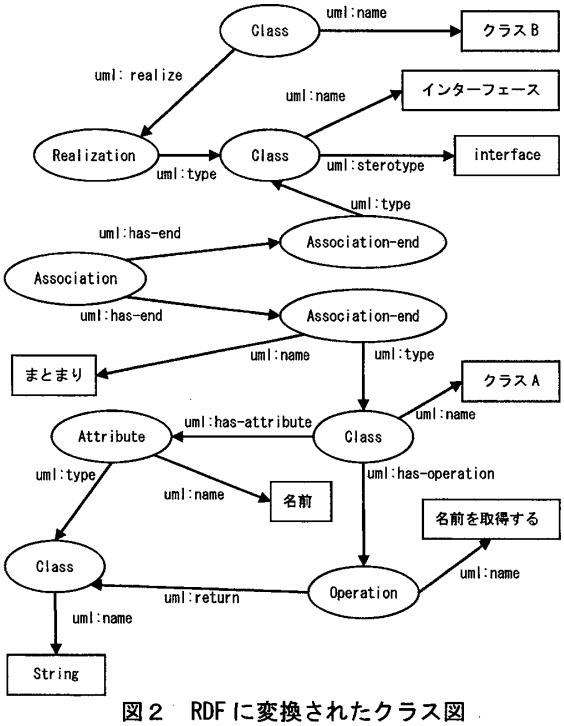


図2　RDFに変換されたクラス図

Generating of RDF Graph from UML Diagrams.
Akifumi HASEGAWA, Michiharu TSUKAMOTO
Tokyo University of Technology, School of Media Science

## 4. SPARQLによる検索

### 4.1 手動作成のクラス図の検索

SAPRQL は RDF クエリ言語であり、RDF グラフ中から任意のパターンを検索することができる。この SPARQL を利用することで、クラス図に対してのパターン検索を行うことができる。まず図１のクラス図から作った RDF に対して検索を行った。

リスト１は図１中のクラス A が関連を持っているクラスを求めるための SPARQL である。このクエリの結果として、"インターフェース"が得られ、図１と一致していることが確認できた。

リスト１　クラス A の関連先を求める SPARQL

```
SELECT ?name
WHERE{
    ?classX uml:name ?name.
    ?classA uml:name "クラスA".
    ?Association uml:has-end ?end1, ?end2.
    ?end1 uml:type ?classA.
    ?end2 uml:type ?classX.
    FILTER ( ?end1 != ?end2 ).
}
```

### 4.2 ソースコードから作成したクラス図の検索

次にオープンソースとして実際に存在しているソフトウェアに対して、クラス図の検索を行った。

astah* professional は Java のソースコードを取り込んで、モデルデータを作ることができる。このモデルデータは、ソースコードからわかるクラスの属性、操作、関連などの情報を持っているため、そのままクラス図として表すことができる。この機能を利用し、Java のオープンソースデータベースである HSQLDB に対して検索を行った。

リスト２は HSQLDB の中のクラスを検索して"java.sql.ResultSet"を実現しているクラスを見つける SPARQL クエリである。このクエリの結果として"org.hdqldb.jdbc.jdbcResultSet"が得られた。この検索結果を確認するため、モデリングツールで jdbcResultSet のクラス図を自動生成したところ、図３の関係を確認することができた。

リスト２　ResultSet の実現クラスを求める SPARQL

```
SELECT ?name
WHERE{
    ?ResultSet uml:name "java.sql.ResultSet".
    ?Realization uml:type ?ResultSet.
    ?ResultSetImpl uml:realize ?Realization;
      uml:name ?name.
}
```


図３　jdbcResultSet の実現関係

## 5. 考察

### 5.1 RDF グラフへの変換による効果

(1)複雑な UML の構造検索が可能

SPARQL を用いることで、指定されたグラフパターンを対象の RDF グラフの中から探すことができる。大規模なソフトウェアではクラスの数が多くなり、関係も複雑になる。SPARQL のクエリに調べたいグラフパターンを記述することで、複雑な関係であっても検索をすることができる。

(2)セマンティック Web 技術の応用が可能

セマンティック Web 技術には、本稿で検索に用いた SPARQL 以外にも、RFD の語彙やオントロジーを定義して新しいトリプルを導くための RDFS や OWL など、様々な技術が存在している。RDF に変換することで、これらの技術を利用することができる。さらに、[2]のクラスファイルなど、クラス図以外のものとも組み合わせることができる。

### 5.2 RDF グラフの問題点

(1)RDF の複雑さ

クラス図の関連には多重度や誘導可能性、関連端名などの様々な情報を持たせることができる。一方 RDF グラフは、１つ１つの述語に複雑な情報を持たせることができない。関連などを表すために、間に多くのノードが必要となる。

このため、UML の単純な関係でも、それを見つけるためのクエリ文は複雑になってしまう。クラスとクラスを直接関連としてつなぐトリプルを新たに RDF グラフに追加することで、クエリをシンプルにできると考えられる。

(2)ソースコードを利用した場合の情報

ソースコードから作ったクラス図からは、ソースコードからわかる情報しか得られない。設計者の意図の記述が失われてしまい、通常の関連なのか、集約やコンポジションなのかの区別するのは難しいなどの問題が生じる。

しかし、このクラス図を編集したり、新しいクラス図を付け加えたりすることで、ソースコードでは表せないクラスの関係も扱うことができる。

## 6. おわりに

クラス図を RDF に変換することによって、セマンティック Web の検索技術である SPARQL をクラス図の検索に利用することができた。さらに高度な検索をクラス図に対して行うため、セマンティック Web の推論を利用することなどが考えられる。また、RDF は情報を組み合わせて扱うことが容易であるため、クラス図だけでなく、他の UML の図や別のドキュメントなどと組み合わせて、検索できるようにしていく必要がある。

### 参考文献

[1] astah* professional, 株式会社チェンジビジョン, http://astah.change-vision.com/ja/product/astah-professional.html

[2] 長谷川明史,塚本享治, OWL2.0 の推論を用いたオープンソース Java ソフトウェアの構造検索と評価, 情報処理学会研究報告, 2009-SE-163, pp.137-144 (2009)