Vol.2010-SE-169 No.1
2010/7/22

クラス図をクエリとして用いるクラス図構造検索手法の提案

長谷川明史[†]　　塚本享治[†]

クラス図中から目的の構造を見つける際に，クエリ文を用いる代わりに検索クエリにクラス図を用いて検索を行う方法を提案する．単純な構造の一致だけではなく，クエリクラス図中でステレオタイプを用いて is-a 関係や関連の推移を利用した検索を行った．クエリのクラス図を使って検索対象のクラス図を検索するため，検索対象は RDF にクエリのクラス図は SPARQL に変換して検索を行った．

A Proposal to Search for Class Diagram's Structures by Class Diagrams

Akifumi Hasegaw[†]　Michiharu Tsukamoto[†]

We propose an approach which search for class diagrams by class diagrams as query. The query diagram written using stereotypes can find not only simple structures but also transitive is-a relations and association. At first the target class diagrams are transformed into RDF graphs. The query class diagrams are transformed into SPARQL statements. And the RDF graphs are searched using the SPARQL statements.

1. はじめに

UML はソフトウェアの設計図であり，分析から設計まで広く利用されている．この UML を使ったソフトウェア開発では，典型的な構造が現れる場面や，過去の設計を利用することができる場面もある．このような時に，既存の UML ドキュメントから実際にその場所を見つけることは，設計の規模が大きいほど難しくなる．

そこで，本稿では，UML の中でもソフトウェアの静的な分析や設計に用いられるクラス図を対象にとりあげ，クラス図をクエリとして用いて検索対象のクラス図中から目的の構造を検索する手法を提案する．複雑な検索クエリ文を記述させるのではなく，見つけたい構造をそのままクラス図で記述し検索用クエリとして用いることができる．また，クエリのクラス図ステレオタイプを使うことで，クエリとの完全一致ではなく，汎化と実現を is-a という広い区分で扱ったり，関連を推移させたりと，より柔軟な検索を可能にする．実際には検索対象クラス図は RDF に，クエリのクラス図は SPARQL に変換して検索を行う．

[†] 東京工科大学大学院　バイオ・情報メディア研究科
Tokyo University of Technology Graduate School

2. クラス図をクエリとして用いる検索

UML はソフトウェアの設計図であるため，設計段階でのソフトウェアの構造が記述されている．実装段階のソフトウェア構造を表しているものとして，ソースコードや実行ファイルなどが検索されることもある．

このようにソフトウェアの構造が読みとれるものは，UML や DFD など様々なドキュメントとソースコードや実行ファイルなどソフトウェアそのものに分類できる．[2]では，ソースコードを分析対象とし，細粒度でソースコードを解析し RDF を利用したソースコードのリポジトリを構築する．一方[1]の研究では，異種のダイアグラム間で検索と相互変換を行い，それによってダイアグラムの再利用を実現している．また，筆者ら[3]も静的なソフトウェア構造を表すクラス図を対象に RDF 化を行っている．

検索の方法としては，検索したいキーワードやパターンをクエリ文として記述することがよく行われる．[2]では，RDF を対象にしているため，RDF 検索用言語 SeRQL を利用してソフトウェアの情報を取得できる． [3]では，RDF 検索言語 SPARQL を用いて検索を行っている．そのため，複雑な構造検索をするためには，それだけ複雑な SPARQL を記述しなければならなかった．

この問題を解決するため，クエリ文を図から生成する方法が考えられる．本稿では，クラス図を検索クエリとして用いてクラス図を提案する．これを実現するため，検索対象は検索対象のクラス図を RDF に，クエリクラス図を SPARQL 文に変換にしてクラスの検索を行う．

3. クラス図とそれを検索するためのクエリクラス図

3.1 検索対象とするクラス図の要素

クラス図はソフトウェアの静的な構造を記述したものであり，属性や操作といったクラスがそれぞれ持つ情報と，関連や汎化などクラスとクラスの間に表わされる情報の２つに分類できる．これらの要素が組み合わせられることで複雑なクラス図が作られる．ここでは，クラスと図１にあげた関連，依存，汎化，実現の 4 つの関係を検索対象とする．

このとき，関連は接続している 2 つのクラスに関してそれぞれ，誘導可能性，集約，

ⓒ2010 Information Processing Society of Japan

コンポジションの情報も併せて用いる．

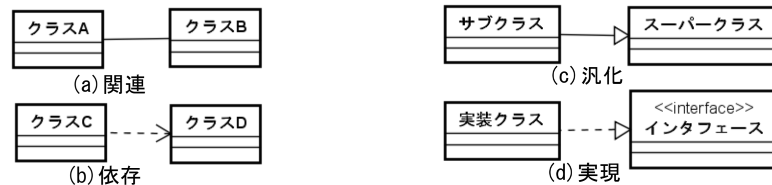


図 1　クラスとクラスの間の関係

3.2 クエリに用いるクラス図

クエリとして用いるクラス図の記述は，通常のクラス図の記述と同様にして行う．このとき，検索対象とする要素名を“?”から始めることで，この要素をクエリ中の変数として扱う．図 N を例に挙げると，この図中の“?AssociatedClass”というクラスと“?associationName”という関連が変数として扱われる．一方“？”から始まらない要素は，そのまま検索対象クラス図中の要素と一致させる．つまりクエリ中の“ClassA”というクラスは変数ではなく，検索対象クラス図の中の ClassA と一致する．

このクエリクラス図を使った検索は，変数つきのクラスの構造にマッチする部分を検索対象クラス図中から見つけることで行う．“?associationName”は ClassA との間にある関連名に，“?AssociatedClass”は ClassA と直接関連を持つクラスに対してマッチする. これを満足する例として, 簡単な例を図 2 で示す. この場合, “?associateionName”には“example”が入り，“?AssociatedClass”には“ClassB”が入る．

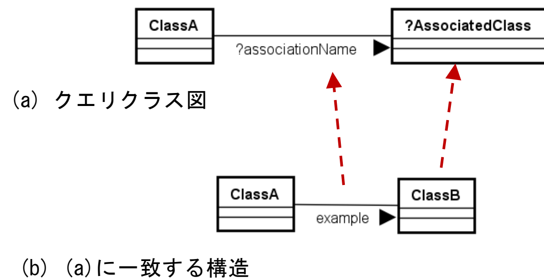


図 2　関連の検索クエリクラス図(a)と，その一致例(b)

3.3 ステレオタイプを利用したクエリクラス図の拡張

クラス図を検索するためのクエリクラス図を記述する際，単純に構造が完全に一致する箇所を探すだけでは柔軟な検索を行うことができない．そこで，ステレオタイプをクエリ中に用いることで，完全一致ではなく，そのステレオタイプの意味に応じた検索を行う．

具体的には，クラスの is-a 関係と推移的な関連の検索クエリに対して，それぞれ<<is-a>>と<<transitive>>というステレオタイプを利用することで検索対象と完全には一致しない構造の検索を行う．

3.3.1 is-a 関係を検索するためのステレオタイプ

汎化や実現関係はどちらもサブクラスとスーパークラス（もしくはインタフェース）との間に is-a 関係がある．そこで，汎化も実現もどちらも is-a として検索できなければならない．

また，サブクラスはスーパークラスとして振る舞うことができる．このとき，直接のスーパークラスだけではなく，スーパークラスのスーパークラスとも is-a 関係が成立する．例えば図 3 の場合，“GrandChild”は“Child”と is-a，“Child”も“Parent”と is-a の関係にある．したがって“GrandChild”と“Parent”との間にも is-a 関係がなければならない．

図 3 の（a）の構造を（b）のクエリで検索しようとすると，“？Super”に一致する結果は“Child”クラスだけになる．一方でクエリ（c）は<<is-a>>ステレオタイプを用いている．これによって“GrandChild”と is-a 関係にあるすべてのクラスが“Super2”に一致できるようにする．

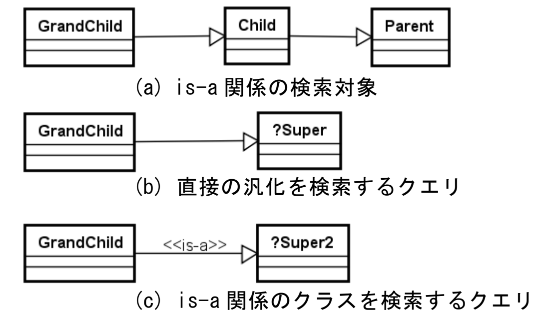


図 3　is-a 関係と２つの検索クエリ

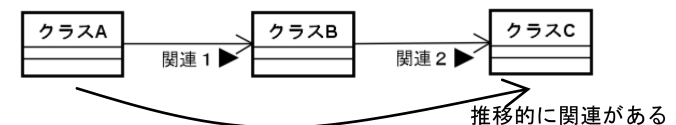


図 4 推移的な関連

ⓒ2010 Information Processing Society of Japan

3.3.2 推移的な関連を検索するためのステレオタイプ

推移的な関連とは，例えばクラス A とクラス B の間に関連があり，クラス B とクラス C との間にも関連があるとき（図 4）の，クラス A とクラス C の間の関係である．クラス A とクラス C はクラス B との関連を通してつながっている．このような推移的な関連を検索するために<<transitive>>というステレオタイプを用いる．

<<transitive>>ステレオタイプを使うことで，関連をたどってたどりつけるすべてのクラスを一致させてしまうということは，関連が続く限り端から端まですべてのクラスと推移的な関連を持つことになる．そこで，この関連を推移する条件を次のように定義し，構造的な意味の近い関連だけを推移するようにする．

1. ２つの関連の種類が同じである
2. 集約やコンポジションでは，まとめる側とまとめられる側の関係の向きが常に等しい
3. ２つの関連間で対応する２組の関連端それぞれの誘導可能性が共通している

条件の１は，関連，集約，コンポジションはそれぞれ個別に推移し，組み合わせることができない，ということを表す．

条件の２は，集約（もしくはコンポジション）しているクラスと集約されているクラスの方向の関係の向きが同じでなければ推移しないということも表している．例えば，図 5（a）では，クラス A がクラス B，クラス B とクラス C の間にコンポジションがある．このとき，クラス A はクラス B をまとめ，クラス B はクラス C をまとめている．このように，集約やコンポジションでは，まとめる側とまとめられる側の向きが同じでないと推移は行わない．

条件の３は，誘導可能性が同じでなければ推移しないということである．例えば，図 5 の関連 1 は，クラス A 側の誘導可能性は未定義であり，クラス B 側へは誘導可能である．関連 2 も同様に，クラス B 側の誘導可能性は未定義であり，クラス A 側へは誘導可能である．このように，関連の誘導可能性が常に一致しているときに，推移的な関連があることにする．これらの条件によって，端から端まですべてと推移してしまうような問題を防ぐことができ，構造の似た関連だけを推移させることができる．例えば，図 5 の（b）のクエリで（a）を検索した場合，“?Part”に一致するクラスは，“クラス B”と“クラス C”である．“クラス D”は関連の種類，誘導可能性の向きが異なっているため，クラス A と推移的な関連があるとはみなさない．

4. 提案手法の実現

4.1 RDF グラフを利用したクラス図の検索

実際にクラス図をクラス図で検索するため，セマンティック Web の検索技術を利用した．まず，検索対象となるクラス図をセマンティック Web のデータ構造である Resource Description Framework (RDF)によって表現する．RDF はトリプルと呼ばれる主語，述語，目的語からなる非常にシンプルな３つ組構造(図 6)によって作られる．このトリプルが集まることで複雑なラベル付き有向グラフとなる．クラス図をこの RDF グラフで表現することによって，RDF 検索クエリ言語をクラス図の検索に利用することができる．

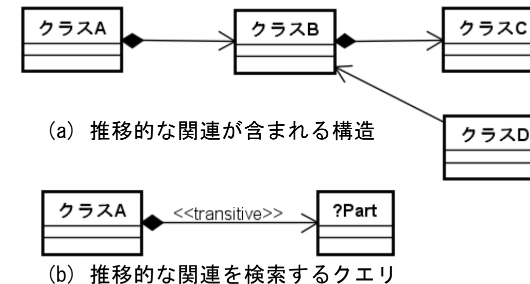


図 5　推移的な関連とクエリ

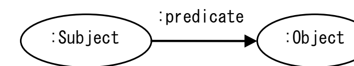


図 6　トリプルの例

次に，検索対象のクラス図から RDF 検索クエリ言語の１つである SPARQL の SELECT 文を作成する．SPARQL はリストのようなクエリ文になり，SELECT 句に検索結果として得たいものを，WHERE 句にグラフ中から実際に検索したいパターンを記述する．この RDF グラフと SPARQL を用い，RDF 検索エンジンで処理を行うことによって，クラスの検索を行う．リスト 1 の SPARQL は，uml:Class のインスタンスの名前を取得する SPARQL であり，図 7 のような RDF のパターンを持つ．

リスト 1　RDF 検索クエリ SPARQL

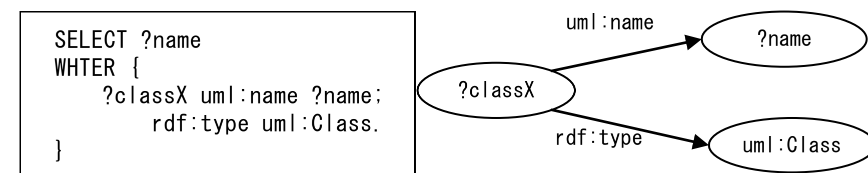


図 7　検索クエリが検索する RDF のパターン

ⓒ2010 Information Processing Society of Japan

また, セマンティック Web 技術では, Web Ontology Language (OWL) [6]や Semantic Web Rule Language (SWRL)[7]を用いて推論規則を定義することができる．OWL や SWRL による推論は RD 中のノードの関係などに基づいて，RDF グラフの中に新しくトリプルを増やす処理である．ここで追加されたトリプルは，もともとの

処理全体の流れは図 8 のようになる．検索対象のクラスを変換した RDF，クエリクラス図を変換した SPARQL，それに検索用に記述した推論規則を合わせ，推論エンジンに処理をさせることで検索結果を得ることができる．

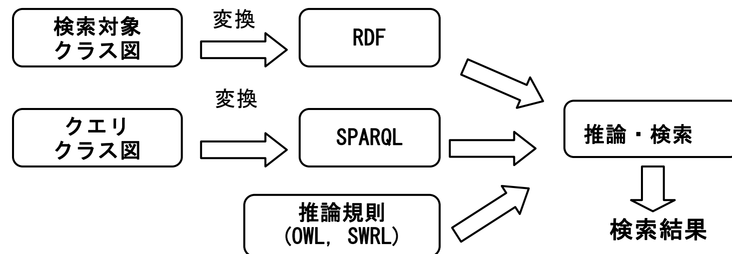


図 8　全体の処理の流れ

4.2 検索対象クラス図の RDF グラフへの変換

4.2.1 情報の抽出

クラス図の RDF グラフへの変換は，UML モデリングツールである astah* professional を用い筆者らによる[3]と同様の手法で行った．まず，検索対象のクラス図を astah* professional を用いて作図し，astah*のファイルとして保存する．保存されたファイルに対して astah* が提供している Java API を利用してファイルからクラス図と，それを作図することで内部的に生成されるモデルデータを取得し，表 1 にある１０種類のノード型と表 2 の１８種類の述語を用いて RDF を作成した．この中には，できるだけ細かく情報取得を行ったため，クエリクラス図による構造検索の直接の対象にならないものもある．

このとき，それぞれの語彙は< http://www.teu.ac.jp/g3109018/uml/>というネームスペースに，その名前を連結して表現する．例えば，表 1 の Class という RDF ノード型は，正確には<http://www.teu.ac.jp/g3109018/uml/Class>を表す．また，uml:Class のように，”uml:”をプレフィックスとして用る．一方で，クラス図から変換した RDF のノードには “software:” をプレフィックスとして用い”uml:”のネームスペースと分けることで，ソフトウェアやプロジェクト固有のネームスペースがつくことを想定している．

表 1　定義した RDF ノードの型

RDFノード型	意味
Class	クラス
Generalization	汎化
Realization	実現
Association	関連
Association-end	関連端
Attribute	属性
Operation	操作
Parameter	引数
Dependency	依存
FQN	完全修飾名

表 2　定義した RDF の述語

述語	主語の型(rds:domain)	目的語の型(rdfs:range)	意味
stereotype	ノード	文字列リテラル	ステレオタイプである
name	ノード	文字列リテラル	要素の名前
fqn	Class	FQN	完全修飾名
generalize	Class	Generalization	汎化している
realize	Class	Realization	実現している
has-dependency	Class	Dependency	依存をもつ
has-operation	Class	Operation	操作をもつ
has-attribute	Class	Class	属性をもつ
abstract	Class, Operation	論理リテラル	抽象クラスかどうか
accesser	Operation, Attribute	public, protected, package, private	可視性
return	Operation	Class	返り値
arg	Operation	Parameter	引数
has-end	Association	Association-end	関連端をもつ
type	Realization, Generalization, Dependency, Attribute, Parameter, Association-end	Class	型
composit	Association-end	論理リテラル	コンポジットかどうか
aggregate	Association-end	論理リテラル	集約かどうか
navigation	Association-end	論理リテラル	誘導可能性
multiplicity	Association-end	文字列リテラル	多重度

クラス図から変換される RDF では，個々のノードに対して astah* が内部的に各要素につけている id 値を利用し，プレフィックス”software:”の後に id 値を連結することによって URI を定める．名前などを URI にしてしまった場合に，偶然同じ名前の要素が現れることで，URI が重複してしまう．そこで，astah*の API から取得できる各要素の id を利用して URI の重複を防ぐ．このようにして割り振られた URI は，”software: 5c0z-g4dd5g47--8z0hta--1gsycq-e91333df13f6d6e175940e51aa34a5d9” のようになる．

4.2.2 クラス図と RDF グラフとの対応関係

図 8，9 はクラス図とそれを変換した RDF グラフを並べて図示したものである．RDF グラフ中の楕円はノードを表し，矩形はリテラルを表す．このとき，URI は省

ⓒ2010 Information Processing Society of Japan

Vol.2010-SE-169 No.1
2010/7/22

略し，ノードにはそのノードの型を書いた．また，リテラルには実際にその値を矩形中に記述した．さらに，それぞれのノードがクラス図中のどこに対応しているかを破線で示した．

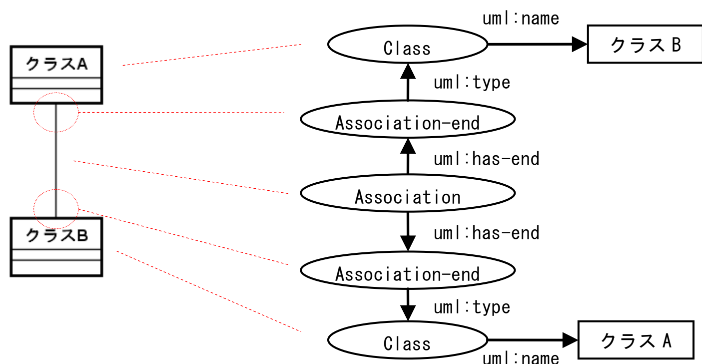


図 8　関連に関しての，クラス図と RDF との対応関係

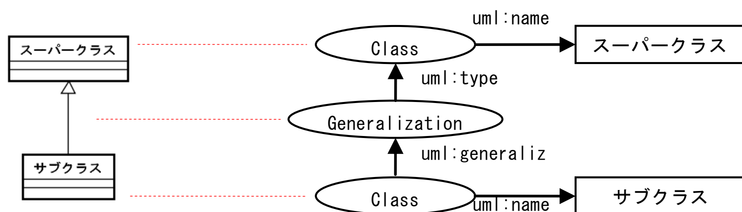


図 9 汎化に関してのクラス図と RDF との対応関係

図 8 は関連を表している．関連には，関連名やコンポジションや集約であるかどうか，誘導可能性，多重度などと多くの情報を持つが，トリプルで作られている RDF で多くの情報を持たせるためには，ある程度複雑な構造になってしまう．ここでは，クラスと関連をつなぐ Association-end にそれぞれの多重度や誘導可能性，関連の関係にあるクラスを持たせ，Association ノードは Association-end を 2 つ持っている，という構造にした．

図 9 は汎化関係のクラス図と RDF グラフである．汎化には誘導可能性や多重度などは存在しないため，Generalization ノードを 1 つ作り，汎化に関する情報はこの Generalization ノードを主語にする．実現や依存関係の RDF の構造も汎化と同様の形になり，Generalization 型の代わりにそれぞれ Realization，Dependency 型のノードが作られる．

4.3 推論規則

SPARQL による検索は，SELECT 文中の WHERE 句に記述された RDF グラフのパターンに完全にマッチする部分の検索である．したがって，前章で述べた is-a 関係や推移的な関連の検索を行うことができない．そこで，OWL や SWRL を使って推論を行い，RDF 中に新しいトリプルを追加する．これによって，is-a 関係や推移的な関連を検索できるようにする．

4.3.1 is-a の推論規則

is-a 関係の推論では 2 つのことを行う．1 つ目は，汎化と実現をどちらも is-a として扱うことである．もう 1 つは，is-a 関係が推移するようにすることである．

まず，汎化と実現を is-a にするためには，rdfs:subPropertyOf を用いる．これは，述語に親子関係をつけるものであり，子プロパティの関係があれば，親プロパティの関係があると導くことができる．そこで， uml:generalize と uml:realize の 2 つの述語を uml:is-a のサブプロパティとして定義することで，この 2 つの述語を区別することなく uml:is-a として扱えるようになる．

次に，is-a 関係が推移するように SWRL を使って記述する．リスト 2 がこの推移をするための SWRL の記述であり，図 10 の実線矢印による構造が出現したときに，破線矢印による uml:is-a 関係のトリプルを導くことができる．

リスト 2　is-a 関係を導くための SWRL

```
<swrl:Variable rdf:about="#childClass"/>
<swrl:Variable rdf:about="#parentClass"/>
<swrl:Variable rdf:about="#parentIs-a"/>
<swrl:Variable rdf:about="#grandParent"/>
<swrl:Imp rdf:about="is-aTransitiveRule">
<swrl:body rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
        <swrl:propertyPredicate rdf:resource="&uml;is-a"/>
        <swrl:argument1 rdf:resource="#childClass" />
        <swrl:argument2 rdf:resource="#parentIs-a" />
    </swrl:IndividualPropertyAtom>
    <swrl:IndividualPropertyAtom>
        <swrl:propertyPredicate rdf:resource="&uml;type"/>
        <swrl:argument1 rdf:resource="#parentIs-a" />
        <swrl:argument2 rdf:resource="#parentClass" />
    </swrl:IndividualPropertyAtom>
    <swrl:IndividualPropertyAtom>
        <swrl:propertyPredicate rdf:resource="&uml;is-a"/>
        <swrl:argument1 rdf:resource="#parentClass" />
        <swrl:argument2 rdf:resource="#grandParent" />
    </swrl:IndividualPropertyAtom>
</swrl:body>
<swrl:head rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
        <swrl:propertyPredicate rdf:resource="&uml;is-a"/>
        <swrl:argument1 rdf:resource="#childClass" />
        <swrl:argument2 rdf:resource="#grandParent" />
    </swrl:IndividualPropertyAtom>
</swrl:head>
</swrl:Imp>
```

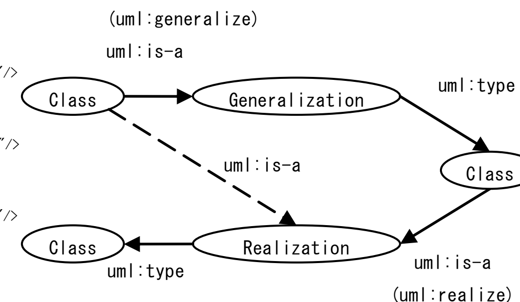


図 10　推論で導かれる is-a 関係

4.3.2 関連の推移のための推論規則

is-a と場合と同様に，<<transitive>>の推移を実現するための推論記述を行う．まず，関連の構造に着目する．RDF グラフ中の関連は，関連（uml:Association）型ノード

ⓒ2010 Information Processing Society of Japan

から 2 つの関連端（uml:Association-end）型ノードに uml:has-end という述語が伸びている．

この構造を維持しつつ推移的関連を導くため, uml:has-end の親プロパティとして，uml:transitive-end を導入する．これによって，uml:has-end であれば uml:transitive-end である，という推論がはたらく．これによって，直接関連で結ばれているクラス間は，必ず推移的な関連で結ばれていることになる．

つぎに，3 章で述べた<<transitive>>ステレオタイプによる推移の 3 つの条件を満たすように SWRL を記述する．1 つ目の条件「関連の種類が同じである」，2 つ目の条件「集約やコンポジションの向きが同じ」ということは，図 N 中の①と②の 2 組の関連端それぞれで，uml:composit，uml:aggregate の値が同じであればよい．また，3 つ目の条件である誘導可能性も①と②の 2 組の関連端がそれぞれ共通の uml:navigate の値を持っていることとして表せる．この条件を満たし，図 11 の実線矢印の構造が現れたとき，破線矢印のトリプルを追加する．これによって，推移的な関連は uml:Association 型ノードから uml:Association-end 型ノードへの uml:tramsotove-end プロパティとして表現することができる．

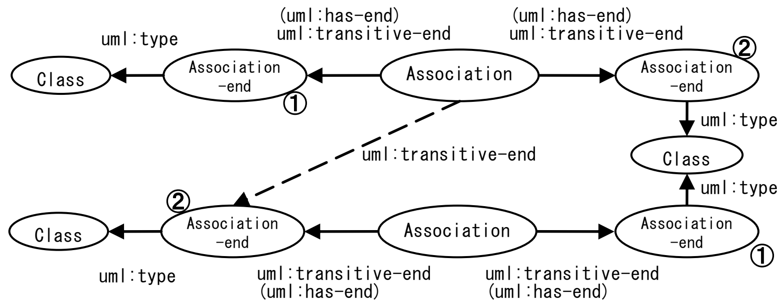


図 11　推移的な関連を導く

4.4 クエリクラス図の SPARQL への変換

RDF に変換されたクラス図に対して検索を行うため, クエリクラス図を SPARQL に変換する．クラスクエリ図も astah*によって記述する．クエリクラス図中に書かれている要素だけを抽出し，RDF 化の際に定義した語彙を使って，検索パターンを記述する. これは, クエリクラスの要素ごとにどのようなパターンを出力するか決めておき，その要素が出現したら対応するパターンを SPARQL の SELECT 句や WHERE 句に付け加える方法で行った．

SELECT 句は検索結果に必要な変数が列挙される. ここでは, クラス名, クラス URI, 関連端名，関連 URI が検索されたとき，その変数を列挙していく．

クラスが出現した場合，それが変数ならば次のパターンを付け加える．

?変数名(クラス)	uml:name	?変数名__name.

というパターンを付け加える．これは，この変数クラスにマッチするクラス名を求めるためである．変数ではないクラスが現れた時は何もしない．

関連の場合は次のようになる．変数名はそれぞれクエリ中で一意になる．

?変数名 1(関連)	uml:has-end	?変数名 2(関連端), ?変数名 3(関連端).
?変数名 2	owl:diffrentFrom	?変数名 3.
?変数名 2	uml:type	<この関連端の指すクラスの URI／クラスの変数>;
	uml:aggregate	<集約かどうかのリテラル>;
	uml:composit	<コンポジションかどうかのリテラル>;
	uml:navigation	<誘導可能性を表すのリテラル>>.
?変数名 3	uml:type	<この関連端の指すクラスの URI／クラスの変数>>
	uml:aggregate	<集約かどうかのリテラル>>;
	uml:composit	<コンポジションかどうかのリテラル>;
	uml:navigation	<誘導可能性を表すのリテラル>.
?変数名 1	uml:name	<関連名／関連名の変数(変数名 1__name)>

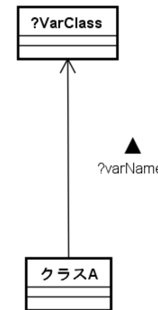
<<transitive>>ステレオタイプを指定された関連の場合は，uml:has-end の代わりに uml:transitive-end を用いる．

次に，汎化の場合を示す．

<クラスの URI／クラスの変数>	uml:generalize	?変数(汎化).
?変数(汎化)	uml:type	<クラスの URI／クラスの変数>

また, 特化, 依存は汎化の場合と同じであり, uml:generalize をそれぞれ uml:realize, uml: has-dependency に置き換える．さらに，<<is-a>>ステレオタイプがついているときは，uml:is-a をもちいる．

図 12 はクエリクラス図とこの変換を行って生成された SPARQL 文であり，図 13 の構造を検索する．




```
1 :  SELECT DISTINCT ?VarClass__name ?VarClass ?varName ?N1000A
2 :
3 :  WHERE{
4 :    ?VarClass uml:name ?VarClass__name.
5 :    ?N1000A uml:has-end ?N1000F,?N10017.
6 :    ?N1000F owl:differentFrom ?N10017.
7 :    ?N1000F uml:type
     software:3qp-gaxr2pmj-cz6c45--1gsycq-e91333df13f6d6e175940e51aa34a5d9;
8 :    uml:aggregate "false";
9 :    uml:composit "false";
10 :   uml:navigation "Unspecified".
11 :   ?N10017 uml:type ?VarClass;
12 :   uml:aggregate "false";
13 :   uml:composit "false";
14 :   uml:navigation "Navigable".
15 :   ?N1000A uml:name ?varName.
16 : }
```

図 12　クエリクラス図と変換された SPARQL 文

ⓒ2010 Information Processing Society of Japan

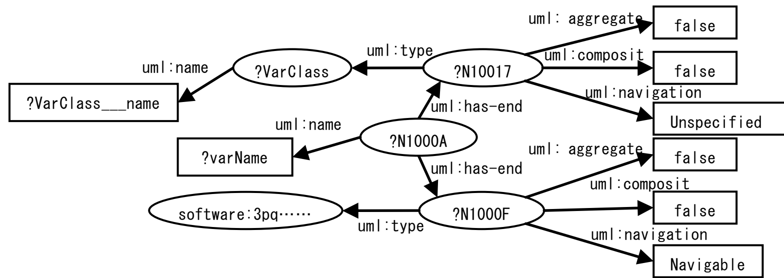


図 13 SPARQL 文が探す関連の構造

5. 実験考察

5.1 クラス図の例題に対しての検索実験

図 14 は[8]の図 7－20 というクラス図の職級体系，従業員管理，銀行管理パッケージのクラスを検索対象として抜き出したものである．このクラス図と推移しないクエリクラス図，推移するクエリクラス図を用意し，検索実験を行った．このときのクラスの数，生成トリプル，推論して生じたトリプルは表 3，表 4 のとおりである．この検索の推論には，SWRL を扱える Pellet 2.1.1[9]という Java の推論エンジンを用いた．

表 3　クラスと関連の数

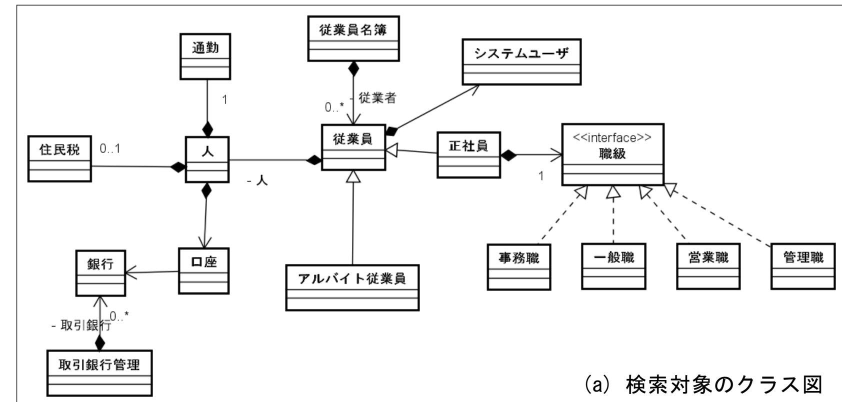
		要素数	
クラス数	検索対象クラス図中	16	18
	クエリクラス図中	2	
関連数	検索対象クラス図中	9	11
	クエリクラス図中	2	

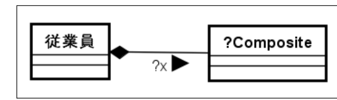
表４　変換後の RDF トリプル数

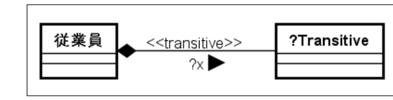
RDFグラフ		トリプル数	
推論前	検索対象クラス図	360	546
	推論のためのOWL	186	
推論後		1034	
推論増加		488	

この検索結果は表 5，6 である．クエリ２の場合推移過程で複数の関連と一致するため，重複した結果が表れている．表 7 はそれぞれ推論記述を使わずにクエリ 1 で検索した場合，推論記述をつけたままクエリ 1 で検索した場合，推論記述をつけたままクエリ２で検索した場合の検索に要した時間である．検索結果にクエリクラス図まで含まれているのは，URL をそろえるため 1 つのファイルに検索クエリと検索対象を記述したためである．

次にストラテジーパターンのクエリ（図 15）を書いて検索を行ったところ，検索に 66 分を要し，表 8 のような結果が得られた．この結果から<<is-a>>ステレオタイプによって，汎化と実現の 2 つの関係を区別せずに検索できていることがわかる．
想定していたストラテジーパターンは，職級に関するものであったが，従業員に関してもクエリと構造が一致してしまっている．また，ストラテジーの実装を表す変数“?Imple1”，“?Imple2”の 2 つに同じクラスがマッチしている行もあることがわかる．







(b) 推移のないクエリ(クエリ 1)　(c) 推移するクエリ（クエリ 2）

図 14　クラス図[8]とそれを検索するクエリ

表 5 クエリ 1 の結果

?Transitive_name	?x
?Transitive	?x
従業員管理.人	
?Composite	?x

表 6 クエリ 2 の結果

?Transitive_name	?x
?Transitive	?x
従業員管理.人	
従業員管理.住民税	
従業員管理.住民税	
?Composite	?x
従業員管理.通勤	
従業員管理.通勤	

表 7 検索時間

推論記述の有無とクエリ	検索時間(秒)
記述なし(クエリ1)	10
記述あり(クエリ1)	120
記述あり(クエリ2)	181

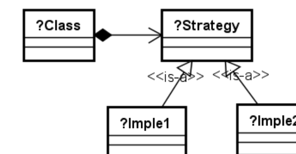


図 15　ストラテジーパターン検索クエリ

ⓒ2010 Information Processing Society of Japan

表 8　ストラテジーパターン検索結果（一部）

? Class__name	?Strategy__name	?Imple1__name	? Imple2__name
従業員管理.正社員	従業員管理.職級	従業員管理.管理職	従業員管理.営業職
従業員管理.正社員	従業員管理.職級	従業員管理.管理職	従業員管理.一般職
従業員管理.正社員	従業員管理.職級	従業員管理.管理職	従業員管理.管理職
従業員管理.従業員名簿	従業員管理.従業員	従業員管理.正社員	従業員管理.正社員
従業員管理.従業員名簿	従業員管理.従業員	従業員管理.正社員	従業員管理.アルバイト従業員
従業員管理.従業員名簿	従業員管理.従業員	従業員管理.アルバイト従業員	従業員管理.正社員
従業員管理.従業員名簿	従業員管理.従業員	従業員管理.アルバイト従業員	従業員管理.アルバイト従業員

5.2　考察

5.2.1 検索評価

OWL や SWRL による推論を用いることで，単純なグラフのパターンマッチングでは行うことのできない構造の検索を行うことができた．しかし，その一方で推論の処理に時間がかかってしまうという問題もあった．この検索の時間は推論記述や検索クエリが複雑になればなるほど，また，検索対象のクラス数が増加すればするほど増加していくと予想できる．少しでも処理時間を短くするために，推論の記述を今よりもシンプルに作り直すことや，現在推論で行っている処理を他の方法で行うことなどを検討しなければならない．

5.2.2 より複雑な検索の必要性

<<is-a>>，<<transitive>>という 2 つのステレオタイプを用いて，汎化と実現を区別せずに祖先まで is-a 関係を推移させたり，同種の関連を推移させたりできるようにした．これによって，直接構造を記述しなければならない場合よりも，シンプルにクエリクラス図を記述することができた．

柔軟なクエリの記述は今回用いた<<is-a>>と<<transitive>>だけではなく，他にも様々な関係の検索をシンプルにすることができ，またそれらを実現していかなければならない．例えば，親の持っている関連は子供にも関係がある場合がある．2 つのクラス間に関係がない，ということも明示的に示したい場合もある．他にも図 16 のように，クラス個々の関係ではなく，クラスのグループとそれに対する他のクラス（a），クラスのグループ同士がどのような関係を持っているか（b）といった検索なども必要であると考えられる．また，GoF[10]によるデザインパターンや J2EE パターン[11]など典型的な問題の解決例がどのように使われているかなどの検索も考えられる．このような高度な検索に対してどのような要求があり，どのようなことを見つける必要があるか分析を行い，実現していくことも今後の課題である．

図 16　クラスやグループとの関連

6.　おわりに

本稿ではクラス図を検索する際，検索するためのクエリ自体をクラス図で検索を行う手法を提案した．ステレオタイプを用いて，is-a 関係や推移する関連を導くことができ，単純にクエリクラス図と検索対象クラス図をマッチさせるだけではできない検索を行うことができた．一方で，このような柔軟な検索を行うために用いた OWL や SWRL によって，推論に要する時間が非常にかかってしまうことが確認できた．

より有用な検索ができるクエリクラス図の実現と，その際いかに推論量を抑えることができるかが課題として残る．

参考文献

1) 谷亀忠,川端亮,伊藤潔：同種ダイアグラムと異種ダイアグラムの検索と相互変換による再利用法,電子情報通信学会技術研究報告,KBSE2008-15
2) 吉田一,山本晋一郎,阿草清滋：宣言的なプログラム解析が可能な RDF に基づく細粒度ソフトウェアリポジトリ,情報処理学会研究報告,2005-SE-147
3) 長谷川明史,塚本享治：UML で記述されたソフトウェアの RDF グラフへの変換,情報処理学会第 72 回全国大会, 6P-5,2010
4) Resource Description Framework, http://www.w3.org/RDF/
5) SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/
6) OWL Web Ontology Language Overview, http://www.w3.org/TR/owl-features/
7) SWRL: A Semantic Web Rule Language,Combining OWL and RuleML, http://www.w3.org/Submission/SWRL/
8) 金澤典子：オブジェクト指向分析／設計教科書,p260,株式会社ソフト・リサーチ・センター,2008
9) Pellet: OWL 2 Reasoner for Java, http://clarkparsia.com/pellet/
10) Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides：オブジェクト指向における再利用のためのデザインパターン,ソフトバンククリエイティブ,1999
11) Deepak Alur, John Crupi, Dan Malks：J2EE パターン,日経 BP 社,2005

ⓒ2010 Information Processing Society of Japan