Vol.2011-SE-171 No.3
2011/3/14

クラス図とシーケンス図からのセマンティック Web 技術を用いたソフトウェアの検索とその評価

長谷川明史[†]　　塚本享治[†]

ソフトウェアの設計において, UML のクラス図とシーケンス図はそれぞれ静的な構造と動的な振舞いを表す目的で使われている．この 2 つのダイアグラムから特定の箇所を検索する方法を提案する．この方法は，検索したい箇所を指定するために検索対象と同種のダイアグラムをクエリとして用いる．まず，検索対象のダイアグラムを RDF グラフに変換し，次にクエリのダイアグラムを SPARQL に変換して検索する．

Searching Class And Sequence Diagrams by Diagrams Translated into SPARQL Statements

Akifumi Hasegawa[†]　and Michiharu Tsukamoto[†]

Class Diagrams and Sequence Diagrams are used for Software design. Class Diagrams describe software's static structures. On the other hands, Sequence Diagrams describe software's dynamic behavior. We propose an approach which search for both diagrams. At first the target diagrams are transformed into RDF graphs. Next, the query diagrams are transformed into SPARQL statements. And RDF graphs are searched using SPARQL statements.

1. はじめに

ソフトウェア開発の場において UML はその設計図として分析から設計段階まで広く利用されている．同じ用途や類似した用途のソフトウェアでは，その中の本質的な部分の構造は類似しているため, UML やそのほかのダイアグラムを再利用することによって，典型的な構造が出る場面や過去の設計を再利用したり，参考にしたりすることができる．

少ない数のダイアグラムから特定の設計を見つける場合，人が見て判断することも容易であるが，近年のソフトウェア開発は大規模で複雑になり，ダイアグラムの数も多くなる．それによって，それらを見つけることは困難になるってしまう．

† 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

本稿では，UML の中でも実際のソフトウェアの構造を表すクラス図と，動作を表すシーケンス図を検索対象とし，同じくクラス図，シーケンス図を利用して記述された検索クエリによって目的の箇所を発見する手法を開発した．クエリ用ダイアグラムで検索対象を指定することにより，複雑な検索クエリ文の記述が不要になり，見つけたい構造をそのままクエリとして記述することができる．このような検索の実現にはセマンティック Web の検索技術を利用した．検索対象のシーケンス図は RDF に，クエリのシーケンス図は SPARQL に変換することにより，一致箇所を検索する．

2. ダイアグラムを用いた設計の検索方法

2.1 ソフトウェアの検索

オブジェクト指向開発においては，再利用できることや想定されるソフトウェアの仕様変更には強く設計することが行われ，そのための方法論としてパターンやイディオム[1][2]などによる設計の再利用が提唱されている. 同じ用途や類似した用途のソフトウェアでは，その中の本質的な部分の構造は類似している．データベースであればデータアクセスオブジェクトと呼ばれる設計やトランザクション処理などがある．異なる用途・目的のソフトウェアであっても，データを加工していくための処理手順が共通であったり，広く知られている設計がされていたりすることも多い.

ソフトウェア中から任意の設計を使われている部分を調べることができれば，その設計を再利用したり，参考にしたりすることができる．ソースコードやオブジェクトコードの再利用はライブラリやフレームワークという形で広く行われている．これと同じように，UML やそのほかのダイアグラムを再利用することによって，典型的な構造が出る場面や過去の設計を利用できると考えられる.

既存のソフトウェアを再利用したり参考にしたりするためには，まずそのダイアグラムを発見しなければならない. 再利用を目的としたダイアグラムの検索する試みは，ユースケース図やアクティビティ図で行われている[3][4]. より下流工程のクラス図やシーケンス図の検索も，不整合箇所やリファクタリング適用箇所の検出目的で行われている[5][6][7]. これらは, Prolog でクエリを記述したり, 見つけたいパターンを XPath や整合性記述文法で指定したりする必要がある．これらの方法は，検索文に精通している場合や，あらかじめ定義された構造だけを検索する場合は問題がないが，ダイアグラム中の任意の構造を検索するために記述するには難しいと考えられる.

検索対象のダイアグラムの中から目的のものを検索するためには，何を検索するかを指定する必要がある．キーワード検索や正規表現による一致などの検索方法もあるが，ダイアグラムの場合は複雑な構造を持っているため，このような検索方法では不十分である.

クエリとして関係データベースの SQL が有名であるが，SQL でいくつものテーブ

ルを結合するような複雑な構造を指定するためには，それだけそのクエリも複雑になってしまう．このようなクエリ文は直感的な検索とはならない．そこで，検索対象と同じダイアグラムをクエリとし，そのダイアグラムの記法をそのまま用いることによって，容易に記述できるクエリを作る．

図中の検索したい要素とクエリに書く要素が同じものになるため，ダイアグラム中のどのような要素を検索するのかがわかりやすい．また，クエリ文にしてしまうと数十行にわたるような検索でも，図であればすぐに書くことができる．

2.2 アプローチ

ダイアグラムの検索にはセマンティック Web の技術を用いる．セマンティック Web では Resource Description Framework（RDF）によって情報を表現する．この RDF はトリプルと呼ばれる主語(Subject)，述語(Predicate)，目的語(Object)の 3 つ組を最小単位としたラベル付き有向グラフであり，グラフ構造の辺が述語になり，ノードは主語や目的語となる(図 1).

また, RDF ではノードや辺を Unified Resource Identifier(URI) を用いて一意に識別している．実際に RDF を記述する際は，XML の名前空間やプレフィックスを利用して略記することができるため, 本稿でもそれにならい, 次のように名前空間を利用する．

rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
owl	http://www.w3.org/2002/07/owl#
uml	http://www.teu.ac.jp/g3109018/uml/
soft	http://www.teu.ac.jp/g3109018/soft/

この RDF を検索するためのクエリ言語として SPARQL を用いる．SPARQL は SELECT 句と WHERE 句の 2 つからなる. SPARQL の WHERE 句には URI と疑問符“？”から名前が始まる変数を用いてグラフパターンを記述する．WHERE 句のグラフパターンの中の変数と一致する箇所を，対象 RDF 中から見つけることで検索を行う．この検索の流れは図 2 のようになる．

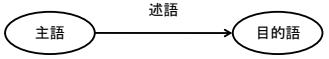


図 1 トリプルの例

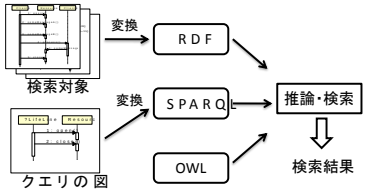


図 2 処理の流れ

クラス図とシーケンス図を RDF にするためには，クラス図の何を RDF の要素にするか決めなければならない．本稿では，RDF のノード型として表 1 を，述語として使うプロパティとして表 2 のものをあらかじめ定義した．また，作図には UML モデリングツール astah* professional[8]を用い，JavaAPI を介してモデルのデータを取得した．

表 1 ノードの型

現れる場所	ノード型	意味
クラス図	Class	クラス型
	Operation	操作型
シーケンス図	LifeLine	ライフライン
	Message	メッセージ
	Activation	実行仕様

表 2 述語として用いるプロパティ

現れる場所	プロパティ	rdfs:domain	rdfs:range	意味
クラス図	associate	Class	Class	関連の最上位プロパティ
	is-a	Class	Class	is-a関係
	depend	Class	Class	依存の最上位プロパティ
	hasOperation	Class	Operation	クラスが操作を持つ
シーケンス図からクラス図へ	lifelineType	LifeLine	Class	ライフラインの型
	operation	Message	Operation	メッセージに対応する操作
シーケンス図	invoke	Activation	Message	メッセージの送信
	implicit	Activation	Message	間接的なものも含むメッセージの送信
	activate	Message	Activation	実行仕様を始める
	lifeline	Activation	LifeLine	実行仕様の属するライフライン
	next	Message	Message	同一実行仕様上での次のメッセージ
	follow	Message	Message	同一実行仕様上での以降のメッセージ
	after	Message	Message	以降のメッセージ
共通	name	owl:Thing	std:string	名前

3. クラス図の RDF 化と SPARQL 化

3.1 検索対象として用いるクラス図の要素と表記方法

Java などのクラスベースのオブジェクト指向プログラミング言語では，クラスを単位としてプログラムを記述する．このクラスはオブジェクトのひな型としての役割を持ち，クラスからその実態であるオブジェクトを生成することができる．

このクラス図の要素は大きく 2 つに分類できる．1 つは属性や操作といったクラスがそれぞれ持つ要素であり，もう 1 つは関連や汎化などのクラスとクラスの間にある要素である．

過去，筆者らによる[9][10]では，クラス図の RDF 化を細かく行っているため，RDF のデータ量によっては推論処理に時間がかかってしまう問題があった．また,「関連がある」,「依存関係がある」ということを RDF として表現しているだけであり，実際のその関連がどのようなものかわからない点が問題であり, 例えば part-of の関係なのか，処理の委譲先なのか，それとも管理者と管理対象であるのかといった区別がつかなかった．これは，クラスや操作，クラス間の関係の RDF 化をクラス図の語彙で行っていたためである．

そこで,「クラス図の関連がある」と表現するのではなく，具体的にどのような関係が 2 つのクラス間にあるのか，どのような役割のクラスなのかを，作図する人が，その対象領域に応じて定義できるようにする．これには, UML のステレオタイプを利用して，図 3 のような記述方法を用いる．この図では，クラス A からクラス B へと誘導可能な関連があり，<<委譲する>>というステレオタイプが存在する，ということが

クラス図の記法からわかる．このような場合に，クラス A はクラス B に委譲する，という関係を考えることができる．そこで，この「委譲する」をそのまま RDF の述語として用いトリプルとして表現することで，クラス図の RDF を表記に意味を付け加えることができる．ただし，この記述方法はクラス図の関連に対して次の記述方法の制約がある．

- 1 方向の誘導可能性を定義しなければならない
- ステレオタイプを使ってどのような関係か記述しなければならない

ステレオタイプにすることで，複数の関係を表せる．<<委譲>>と<<管理>>などを持つことができる．また，RDF では有向グラフであるため，どちらからどちらへの関係か，その向きを確実に表す必要がある．そこで，双方向の誘導可能性と誘導可能性を定義しない記述は避ける．

クラスの関係を直接述語としてトリプルを作るため，次のものを扱うことができなくなってしまうが，これらの要素が持っていた意味はステレオタイプの中に含まれているものとみなす．

- 多重度
- 関連名
- 関連端名
- 集約，コンポジションかどうか

図 3 関連の意味を付け加えたクラス図の記述とその RDF

このように，語彙によってクラスとクラスの関係を表す新しい RDF モデルでは，クラスとクラスの関連を表すためのプロパティを図 4 のような階層構造にする．まず，最上位の関連として uml:associate というプロパティを定義する．この uml:associate のサブプロパティとしてどのような関連なのかを表すプロパティを定義する．図中では，model:delegate，model:aggregate，model:manage というような関連を定義している．この関連のプロパティは何をモデル化するか，その対象領域ごとに語彙が異なる可能性がある．ドメインごとに uml:associate のサブプロパティを定義し利用する．

次にクラスに対してもステレオタイプで語彙の定義を行えるようにし，図 5 のような意味に基づいたクラスを作成する．UML のクラスは uml:Class のインスタンスとして RDF に変換するが，より詳細にどのようなクラスなのかを表すため，UML のクラスに記述されているステレオタイプを uml:Class のサブクラスを階層構造にする．

uml:Class は UML 中に記述されるクラスを表す OWL のクラスである．図 5 中央ではこのクラスの特殊な場合として，rdfs:subClassOf を用いて uml:Class のサブクラスとして model:Finalizable を定義する．そして，soft:Stream という UML のクラスは model:Finalizable 型のノードであると表現する．

操作に対しても同様の拡張を行う．図 5 右では，uml:Operation のサブクラスとして model:Finalizer と model:Initializer を定義し，soft:close，soft:open をそれぞれ属させる．

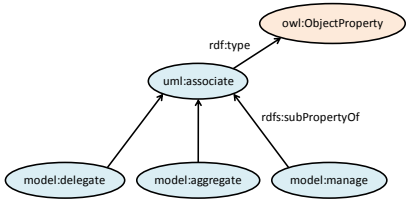


図 4 関連を表すプロパティの階層構造

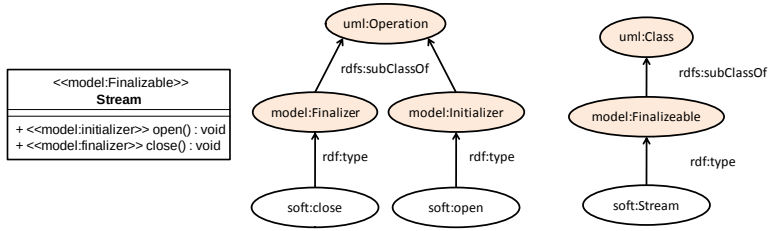


図 5　クラスとクラスの型，操作の型の階層構造

このようにすることで，操作名と切り離した操作の役割を表現することができる．例えば，“close”という言葉から，オブジェクトが不要になった時に呼び出すものと想像するのではなく，model:Finalizer は終了するときに呼び出すもの，と定義することができる．これによって，操作の名ではなく役割による検索ができる．実際の操作名では，close 以外にも dispose や unlock，shutdown なども最後に呼び出されることが想定される名前である．UML のクラスは uml:hasOperation で操作を持つため，図 5 のクラス図は図 6 のように RDF 化することができる．さらに，依存関係はクラスとクラスを直接 uml:depend で結び，汎化と実現は uml:is-a を述語として用いる．

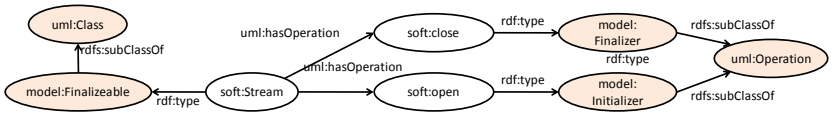


図 6　クラスと操作に語彙の定義を利用したクラス図の RDF 化

操作と is-a 関係を利用して，親の操作は子が継承するということを OWL で表現できる．owl: Property Chine を用い，uml:is-a と uml:hasOperation という述語でたどれる要素間に uml:hasOperation という述語を導く．

このように，全体としてクラスの関係をシンプルに定義することによって，図 7 のように，関連を表すためのトリプル数が大きく減り，実際[10]で推論ができなかったクラス図のトリプル数が, 614 トリプルから 130 トリプルに減少したことを確認した．同時に，複雑な推論規則の記述が不要になる．この記述と変換方法では，UML モデルの対象領域によって使われる語彙やその意味が変わってしまうため，それに応じた定義と推論規則の用意が必要であるが, より意味をもったクラス図の記述が可能である．また，定義された語彙による関連は図 8，汎化や実現関係は図 9 のように表わす．

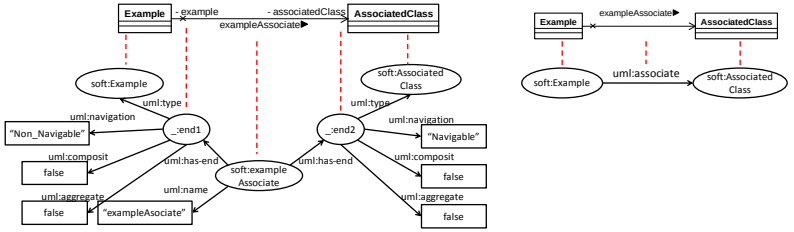


図 7　[9]での関連（左）と本稿での関連（右）の比較

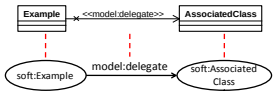


図 8　委譲という意味を持つ関連

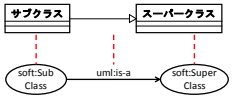


図 9　is-a 関係

3.2　クエリ用

クエリとして用いるクラス図は通常のクラス図と同様に記述する．このとき，検索対象の要素名を疑問符“？”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．この変数に一致する要素を検索対象のクラス図から見つけることで検索を行う．

クエリの変数とできる要素はクラスと操作であり，それぞれクラス名，操作名を疑問符から始めることにする．疑問符から始まっていないものは，検索対象と一致する要素として扱う．クラスが決まらなければ操作が決まらないため，変数のクラスは変数ではない操作を持てない．

ステレオタイプがクエリで指定されていた場合，それと同じか owl のサブプロパティ，OWL のサブクラスのステレオタイプを持つ要素の検索を行う．ステレオタイプが指定されていない場合，uml:Class や uml:Operation，uml:associate など，最上位の要素として検索するため，ステレオタイプの有無にかかわらず検索ができる．

図 10 を例に挙げると，この図中の?ClassA と?ClassB，?operation の 3 つの要素がこのクエリの変数になる．つまり，?ClassA が関連を持つ?ClassB と，?ClassB の持っている操作?operation を検索するクエリになり，破線矢印のような一致の仕方をする．

このとき，検索対象側では関連が<<model:delegate>>であり，検索クエリ側では指定がないので uml:associate を検索する．model:delegate は uml:associate のサブプロパティであることから，推論によって検索することができる．また，?ClassB が Example2 に一致し，?ClassB に一致するクラスが持っている操作?operation は ExampleFunction と一致する．

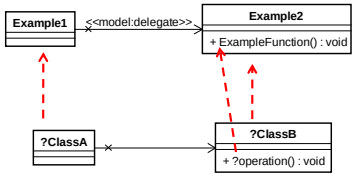


図 10　クエリ中の変数と検索対象との一致

4.　シーケンス図の検索

4.1　検索対象として用いるシーケンス図の要素と表記方法

シーケンス図はソフトウェアの動作を表す相互作用図の一つであり，オブジェクトのライフラインとその間でやり取りされるメッセージが時系列に沿って記述される．あるライフライン上で行われる一連の動作は実行仕様（活性区間）としてまとめることができる．筆者らは[11]で，シーケンス図だけの検索を行った．このシーケンス図の検索は時系列に沿った表現であるため，次の要素を検索対象にした．

- ライフライン
- 実行仕様（活性区間）
- メッセージ
- メッセージの順序

シーケンス図の要素がそれぞれどのような RDF になるかは図 11 に示す．

（a）はライフラインと実行仕様の対応を表している．Soft:LA がライフラインを表すノードであり，_:b1 が実行仕様を表すノードである．_:b1 は soft:LA の実行仕様であるため，uml:lifeLine という述語でつながる．

（b）はライフラインの間にあるメッセージのやり取りを RDF にしたものである．_:b1 はメッセージ送信側の実行仕様，_:b2 はメッセージ受信側の実行仕様である．また，soft:M1 がメッセージを表すノードである．送信側からは uml:invoke を述語にした

辺がメッセージに，メッセージから受信側に uml:activate を述語にした辺が受信側実行仕様に伸びる．

（c)は 1 つの実行仕様上におけるメッセージの順序の表現方法の例である．メッセージ 1 の後にメッセージ 2 をやり取りしている．そこで，それぞれのメッセージは同一実行仕様中で次に呼び出されるメッセージがあれば，uml:next を使ってそれを表す．

（c）は同一実行仕様上におけるメッセージの順序として定義し，実行仕様を問わないメッセージの順序表現としては uml:after を用いた．すべてのメッセージは，そのメッセージよりも下に記述されたメッセージを uml:after の目的語として持つ．

（d）は同一実行仕様上でそのメッセージの後に送受信されるものを表す．ここでは，soft:M1，soft:M3，soft:M2 の順番でメッセージが呼び出されているため，soft:M1 と soft:M3 の間，sofut:M3 と soft:M2 の間に uml:next がある．この uml:next の親プロパティとして uml:follow を定義し，uml:follow を推移的プロパティとする．これに対して推論を行うことによって，soft:M1 と soft:M2 の間に uml:follow の述語を導くことができる．

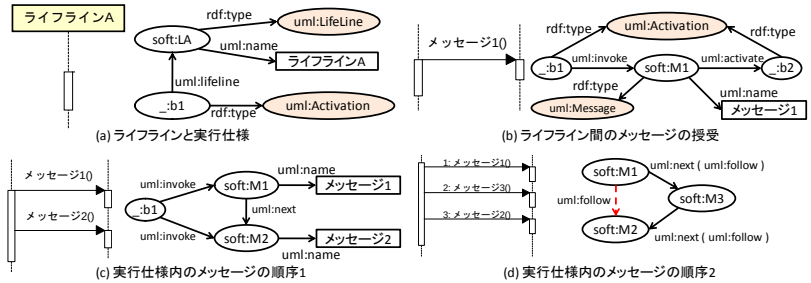


図 11 シーケンス図と RDF との対応関係

4.2 シーケンス図とクラス図の対応

クラスとはオブジェクトの雛型であり，これをインスタンス化したものがオブジェクトである．シーケンス図中のライフラインはまさにこのオブジェクトに該当する．そこで，シーケンス図中のライフラインとクラス図中のクラスの対応関係を RDF で表現する．ここでは，uml:lifelineType というプロパティを述語として用いることでこの関係を表現する．

メッセージと操作の対応関係は uml:operation というプロパティを述語として用いることで表す．シーケンス図中でライフラインがメッセージを受信しているが，すべてのメッセージを受信できるわけではない．本来，受信できるメッセージは，対応する操作がクラスに定義されているものだけである．これらの対応関係は図 12 である．

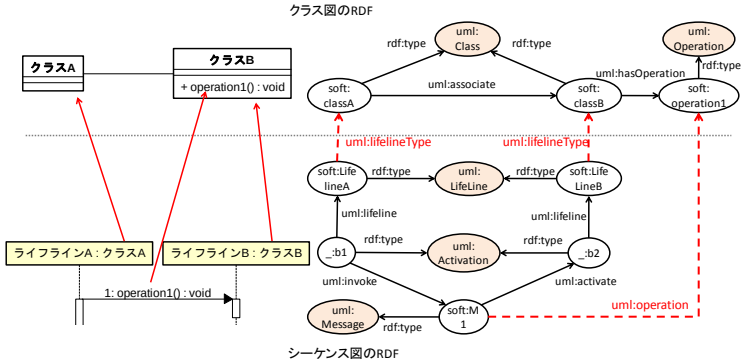


図 12 クラス図とシーケンス図の対応関係

操作のオーバーライドはシーケンス図と組み合わせた際にメッセージの対応を複雑にする．図 13 の場合，ライフライン:SubClass は:ClientA と:ClientB から operation2 というメッセージを受信している．このとき，クラス図を見ると ClientA は SubClass ではなく SuperClass との関連を持っているため，SuperClass の operation2 を送信していると判断できる．一方で ClientB は SubClass との関連を持つため，operation2 は SubClass の持っている operation2 を送信していると判断できる．このように，オーバーライドされている操作とメッセージを対応付けるときは，メッセージを送信するライフラインが，メッセージを受信するオブジェクトに対して，どのクラスのインスタンスとして見えているか，どのクラスのインスタンスとして扱っているかが重要になる．そのため，図 13 の operation2 ようにシーケンス図を作図する際にそのことを意識して，メッセージと操作を対応付けなければならない．

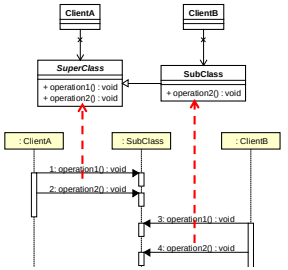
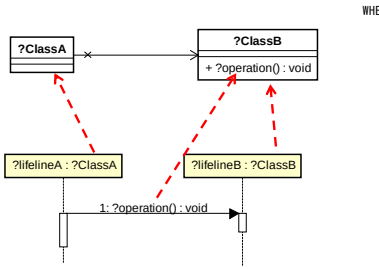


図 13 オーバーライドにおいて，対応付ける操作を切り替える

4.3 クエリ用

シーケンス図のクエリもクラス図のクエリと同じように，検索対象の要素名を疑問符“？”から始めることにした．クエリの変数として記述できるのは，ライフライン名，ライフラインの属するクラス名，メッセージ名であるが，ライフラインのクラス名とメッセージ名はそれぞれ，クラス図のクラス名と操作名と対応づいている．そのため，シーケンス図側で指定する変数はライフライン名だけである．他方[11]では，クラス図と組み合わせていなかったため，ライフライン名とメッセージを指定した．図 14 はクラス図とシーケンス図からなるクエリと，それによって生成される SPARQL のパターンである．さらに，他のライフラインを介した間接的なメッセージのやり取りも検索できるように，<<implicit>>というステレオタイプを用いる．




```
WHERE {
   ?ClassA uml:name ?ClassA__Name;
      a uml:Class.
   ?ClassB uml:name ?ClassB__Name;
      a uml:Class.
   ?ClassB uml:hasOperation ?operation.
   ?operation uml:name ?operation__Name.
   ?ClassA uml:associate ?ClassB.
   ?N65557 a uml:LifeLine;
      uml:lifelineType ?ClassB;
      uml:name ?lifelineB.
   ?N65562 a uml:LifeLine;
      uml:lifelineType ?ClassA;
      uml:name ?lifelineA.
   ?N65545 a uml:Message;
      uml:operation ?operation;
      uml:activate ?N65547.
   ?N65547 uml:lifeline ?N65557.
   ?N65549 uml:invoke ?N65545;
      uml:lifeline ?N65562.
}
```

図 14　クエリのダイアグラムと SPARQL のパターン

5. 検索実験

5.1 実験データ

クラス図とシーケンス図を組み合わせた検索を行うため，ソフトウェアのクラス図と対応するシーケンス図が載っている書籍から検索用データを作成し，実際にクエリを記述することでそれらを検索できることを確かめた．

実験として用いたデータは，GoF のデザインパターン[1]から Abstract Factory, Builder, Chain Of Responsibility，Composite，Decorator，State，Strategy，J2EE パターン[12]から Data Access Object，Business Delegate，Composite Entity，Transfer Object，Intercepting Filter である．また，Java における典型的な終了化処理の例として java.io.InputStream と java.awt.Graphics, java.util.concurrent.ExecutorService の利用を想定したクラス図とシーケンス図も用意した．

各要素とトリプルの数はそれぞれ表 3　表 4 に示す．3 つのグラフの和とは，クラス図，シーケンス図，語彙定義を合わせたものであり，この状態で推論を行った際の結果である．3 つのグラフを合わせて推論を行うことで，はじめて導かれるトリプルもあるため，推論の増加量は個別に推論したものより多い．また，個別に推論をした場合，3 つの推論後の RDF で重複したトリプルが生じるが，同じトリプルはいくつあっても 1 つとして数える．これらのことから，推論後のトリプル数は個別に行った場合の和と，あらかじめ和をとってから推論を行った場合で数が異なる．

実験用ダイアグラムを作図する際に，クラスとクラスの間にどのような関係があるかを考え，model というプレフィックスで定義していった．この要素は表 5 である．

表 3 検索対象の要素数　表 4 変換して得られたトリプル数　表 5 新たに定義した語彙

要素	記述数
クラス図	29
シーケンス図	32
クラス	166
操作	147
ライフライン	143
実行仕様	258
メッセージ	206
関連	85

RDFグラフ	推論前トリプル数	推論後トリプル数
クラス図	1345	2263
シーケンス図	3198	5157
語彙定義	120	347
3つのグラフの和	4463	7762
next(follow)	81	137
invoke(implicit)	206	382

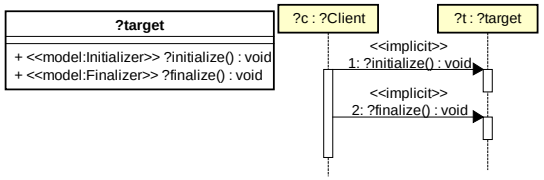
親	定義した用語
uml:Class	Bug
uml:Operation	Initializer
	Finalizer
	Setter
	Getter
uml:associate	has
	delegate
	create
	use
	lookup
	manage
	factory
	aggregate
	composite
uml:depend	create

5.2 初期化と終了化

クラス図の要素に語彙を定義した意味の付与を行ったため，それを利用した検索の実験を行う．必要な時に初期化を行い，不要になったら終了化を行うことはプログラミングの基本である．そこで，操作に model:Initializer（初期化子の意）と model:Finalizer（終了化子の意）を持っている場合，初期化子のメッセージが送信された後に終了化子のメッセージを送信している部分の検索を行う．

これらを検索するために記述したクエリが図 15 であり，model:Initalizer の操作を呼び出してから model:Finalizer の操作を呼び出す部分を検索する．検索結果が表 6 である．

表 6　クエリの検索結果



初期化、終了化の検索結果
"FileInputStream"
"ConnectionImpl"
"DataSource"
"Graphics"
"ThreadPoolExecutor"
"BufferedInputStream"

図 15 初期化と終了化の対応を検索するクエリ

FileInputStream と BufferedInputStream は図 16 のような委譲関係で使われていたが，

そのどちらも検索することができた．ConnectionImpl,DataSource は Data Access Patarn のものであった．また，図 17 は終了化子 shutdown の後で submit メッセージを送信するシーケンス図である．このようなものは，リスト 1 のような推論記述で検索が可能である. この推論は 図 17 中の Client クラスを <<model:Bug>>型であると報告する．

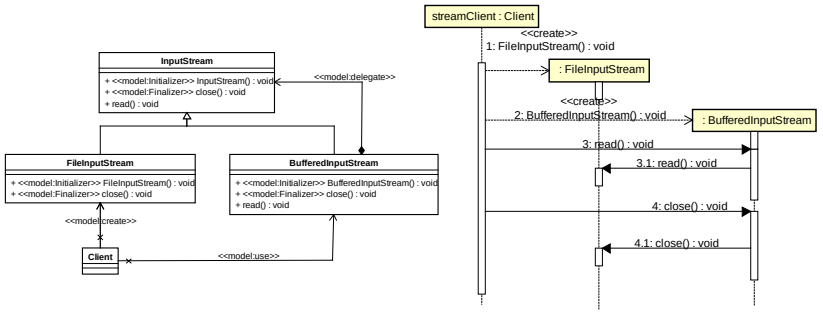
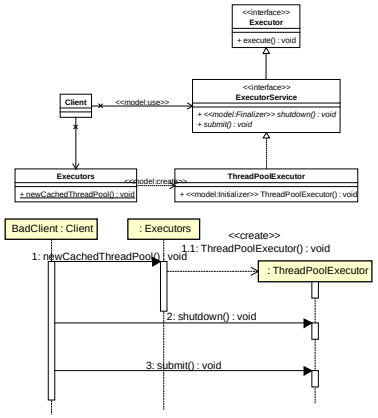


図 16　BufferedInputStream と FileInputStream の委譲構造

リスト 1　推論でバグを報告する OWL



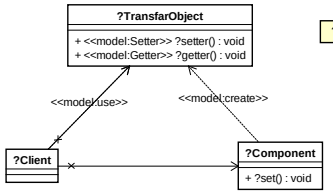

```
<owl:Restriction>
  <owl:onProperty rdf:resource="&uml;invoke"/>
  <owl:someValuesFrom>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty>
          <owl:inverseOf rdf:resource="&uml;follow"/>
        </owl:ObjectProperty>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&uml;operation"/>
          <owl:someValuesFrom rdf:resource=&model;Finalizer"/>
        </owl:Restriction>
      </owl:someValuesFrom>
    </owl:Restriction>
  </owl:someValuesFrom>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&uml;lifeline"/>
      <owl:allValuesFrom>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&uml;lifelineType"/>
          <owl:allValuesFrom>
            <owl:Class rdf:about=&model;Bug"/>
          </owl:allValuesFrom>
        </owl:Restriction>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Restriction>
```

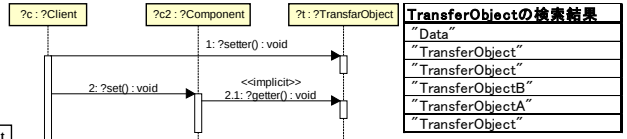
図 17　バグを含む Executor Service

5.3　Data Transfer Object

Data Transfer Object とは複数のデータを転送するためにもちいられるパターンであり, クラスの間でデータをやり取りするために, データを直接扱わずに, Transfer Object を介した受け渡しを行う. Transfer Object を介してデータ渡す場合, まず Transfer Object に実際に渡したいデータをセットする．次に，データを渡したい相手にこのオブジェクトを送る．データを受け取った相手は，Transfer Object からデータを取り出す．このような一連の流れをクエリとして記述したものが図 18 であり, 実際にこのクエリで検索を行ったところ, 表 7 のクラスが見つかった. “Data” は J2EE の Data Access Object パターンの節にあったクラスであり，確かに Data Transfer Object の役割をしていた．他は，Data Transfer Object パターンの節にあったものであり，Data Transfer Object として記述したものはすべて見つかった．

表 7　Transfer Object の検索結果





TransferObjectの検索結果
"Data"
"TransferObject"
"TransferObject"
"TransferObjectB"
"TransferObjectA"
"TransferObject"

図 18　Transfer Object のクエリ

6.　評価・考察

6.1　検索にかかる時間

表 8 は実験にもちいたクラス図とシーケンス図を複製してデータを増やした時の，全推論の時間，全トリプルを検索した時の時間，Data Transfer Object のクエリで生成された SPARQL で推論済みの RDF を検索した結果である．これをグラフ化したのが図 19 である．グラフからわかるように，トリプル数に対して推論や RDF 検索に必要な時間が大きく増加することが分かる．この中で，特に推論の増加する割合が多い．

表 8　トリプル数と推論，検索時間

トリプルは個、時間は分:秒

推論前トリプル	推論後トリプル	全推論時間	全検索時間	DTO検索時間
4663	7762	00:05.9	00:02.1	00:03.5
9156	14947	00:15.9	00:03.1	00:08.7
13649	22132	00:30.9	00:04.6	00:18.1
18142	29317	00:52.6	00:06.7	00:30.7
22635	36502	01:17.6	00:09.8	00:48.1
27128	43687	01:53.3	00:13.3	01:05.9
31621	50872	02:35.0	00:17.7	01:30.5

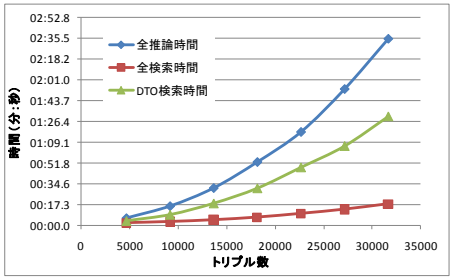


図 19　トリプル数による処理時間の推移

また，表 9 から，推論のタイミングによっても検索に必要な時間が異なることが分かる．この結果から，あらかじめ推論を行い全トリプルを導いてから，RDF の検索を行ったほうが全体の処理時間が短くなると判断できる．

そこで，表 10 は初期化終了化の検索で行った Bug を導く規則の有無に対して，推論時間がどのように変化するかを示した表である．Bug を導く規則を追加するため，全推論に必要な時間が非常に増加することが確認できた．

表 9 推論のタイミングと処理時間

トリプルは個、時間は分:秒

検索方法		推論・検索時間	
推論と検索の同時実行		686:16.7	
順番に行った場合	推論	00:05.9	00:09.4
	検索	00:03.5	

表 10 Bug を導く推論の有無と処理時間

トリプルは個、時間は分:秒

トリプル数	Bugの規則の有無	全推論の時間
991	無	00:02.9
	有	00:03.7
4663	無	00:05.9
	有	19:47.4
9156	無	00:15.9
	有	145:08.65

本稿のようなパターンを個別に作成したデータでは，それぞれのダイアグラム間でのつながりが全くないため，それぞれのダイアグラム間でまとめて RDF 化，推論をするのではなく，分割して推論を行うことが可能であると考えられる．このように，対象グラフを小さくして推論し，マージしてから検索を行うことで推論の規模を小さくできる．実際，Bug のような複雑な推論も表 10 のようにすることで，検索の時間を短くすることができた．しかし，検索対象を分割して検索を行う場合，相互に密接な関係を分割してしまうと，推論結果が変わってしまう可能性がある．そのため，分割するタイミングとその範囲は適切に選ぶ必要がある．

6.2 ダイアグラムを対象とした検索

ダイアグラムを検索の対象として扱うときの問題点として，正確に記述されていない場合があった．astah* professional は UML などをモデリングするためのツールであるため，内部的なモデルデータのデータ構造の整合性は保たれていると考えられる．しかし，クラス図やシーケンス図としては一見正しく見えても，図の要素に対して誤った使い方，誤った表現の仕方をしている可能性がある．ソースコードであれば，コンパイラが解釈できない文法エラーなどにあたるが，ダイアグラムの場合でもそのような妥当性の検証が必要である．

クエリ用のクラス図やシーケンス図から SPARQL クエリに変換するために，あらかじめ変換パターンを定義し，ダイアグラム中に特定の要素が現れたら対応するグラフパターンを SPARQL の WHRER 句に出力する方法で行った．この方法は，直接クエリを記述する場合に比べ，記述できるクエリに制限がある．そこで，ダイアグラムを検索クエリに使う際に，様々な場合を区別して記述できる方法が必要である．

7. おわりに

ソフトウェアの分析設計にもちいられるダイアグラムのうち，静的な構造を記述するクラス図と動的な振舞いを記述するシーケンス図を対象に，クラス図とシーケンス図をクエリとした検索する方法について述べた．この方法は，クラス図とシーケンス図それぞれ個別に RDF とその検索言語 SPARQL に変換し，それらを合わせることで構造と振舞いをもったソフトウェアの検索を可能にした．

ソフトウェア設計上の語彙や特定ドメインの語彙を整理し，それに基づいたモデリングが行を行うことによって，より柔軟な検索が可能になる．しかし，実用的な検索手法とするには，推論に必要な時間とメモリを減らすこと，不整合のない図の記述すること，様々な検索条件をクエリ側で指定することなどが必要である．

参考文献

1) 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
2) Frank Buschmann, Hans Rohnert , Michael Stal , Regine Meunier , Peter Sommerlad , 金沢 典子 (翻), 桜井 麻里 (翻), 千葉 寛之 (翻), 水野 貴之 (翻), 関 富登志 (翻) ,ソフトウェアアーキテクチャ―ソフトウェア開発のためのパターン体系,近代科学社 ,2000
3) 笠原 利春,川端 亮,伊藤 潔,ジェネリックタスクを包含したドメインモデルに基づくユースケースダイアグラム再利用法,信学技報,KBSE-2006-35, 2006
4) 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,信学技報,KBSE-2008-15, 2008
5) 増田 敬史,吉田 則裕 ,浜口 優,井上 克郎,UML モデルを対象としたリファクタリング候補検出の試み,信学技報,SS-2008-5 KBSE-2008-5, 2008
6) 佐々木亨, 岡野浩三, 楠本真二,制約指向に基づいた UML モデルの不整合検出・解消手法の提案,電子情報通信学会論文誌. D, 情報・システム J90-D(4), 1005-1013, 2007
7) 佐々木亨, 岡野浩三, 楠本真二,UML モデルに対する XPath と XMI-difference を用いた不整合検出と解消,信学技報,SS-2005-48, 2005
8) astah* professional,株式会社チェンジビジョン,
http://astah.change-vision.com/ja/product/astah-professional.html
9) 長谷川明史,塚本 享治: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,情報処理学会研究報告, Vol. 2010-SE-169, No.1 ,2010
10) 長谷川明史,塚本 享治:クラス図によるクラス図構造検索の評価, Fit2010 B-016, 2010
11) 長谷川明史, 塚本享治, “シーケンス図をクエリとして用いるシーケンス図の振舞い検索の提案”, 信学技報,KBSE2010-43, 2011
12) Alur, John Crupi, Dan Malks,J2EE パターン,日経 BP 社,2005