

複数の言語で記述されたプログラムの解析における RDF リポジトリの応用

風戸 広史^{†1} 宮田 俊介^{†1} 星野 隆^{†1}

本報告では RDF に基づくソフトウェアリポジトリを応用し、複数のメタデータや情報源を横断的に利用するプログラム解析の手法を提案する。提案手法は複数回のプログラム解析を繰り返す過程において、前段の解析結果を後段の解析手法の選択や、選択された解析手法への入力として利用する点に特徴がある。我々は RDF リポジトリに基づくプログラム解析ツールの基盤を試作し、オープンソースで公開されている 3 つの Web アプリケーションの例題へ提案手法を適用することにより、手法の実現可能性を確認した。

Applying RDF-based Software Repositories to Multi-Language Program Analysis

HIROSHI KAZATO,^{†1} SHUNSUKE MIYATA^{†1}
and TAKASHI HOSHINO^{†1}

In this report, we propose to apply RDF-based software repositories to analyze programs implemented in multiple program languages. Our technique consists of iterative steps of program analysis where outputs from the previous steps are utilized for choosing an analysis technique to apply in the following step. To show feasibility of the proposed technique, we have prototyped a supporting tool based on RDF repositories and applied it to three open source web applications.

1. はじめに

情報システム分野では単一のプログラミング言語のみでソフトウェアが構築されることが少なく、特定の目的ごとに特化した複数のプログラミング言語を組み合わせる。たとえば、Java EE で実装された Web アプリケーションでは、ユーザインタフェースの表示やインタラクションの記述に HTML や JSP、Web サーバ内の処理の記述は Java、永続化データの操作には SQL などの言語を併用して実装することが一般的である^{*1}。

このような現実的なソフトウェアの保守を支援するために、複数のプログラミング言語を組み合わせに対する解析技術の必要性が認識されている。Van Greet らはメインフレームのレガシー資産に対し、IEEE TSE などの著名な論文誌で有効性があると示されたプログラム解析手法^{1),2)}を適用して得た知見を整理し、複数言語に横断的な依存性の抽出が未開拓の研究領域であることを指摘した³⁾。また、ウィンターワークショップ 2007 のプログラム解析セッションでは同様の課題認識に基づいてグループ討論が行われ、複数の言語を対象とする解析技術や、複数の解析技術を組み合わせる必要性について議論された⁴⁾。

複数のプログラミング言語を対象とする解析の具体例として、前述の Web アプリケーションのソースコードを解析し、データフロー図 (DFD) を生成する場合を考える。この場合、HTML の入出力コントロールとして記述されたデータ項目と、Java のクラスやメソッドとして実装されたプロセス、関係データベース上のテーブルとして実現されたデータストアの間のデータ依存関係を抽象化したグラフを構築する必要がある。このように、複数の言語やメタデータを併用するソフトウェアを解析する場合、あらゆる組み合わせに対応できる解析ツールを予め開発しておくことは、現実的には不可能である。そこで、我々は複数のツールによって個別に解析されたデータをアドホックに併合し、言語やメタデータを横断する依存関係を追加した異種混在の解析データを生成したいと考える。このような解析データが利用できれば、スライシング、クラスタリング、マイニング等のより先進的なプログラム解析、保守技術を現実的なソフトウェアの解析において活用できる可能性が高まる。

我々はこれまでに、多段階の解析によりプログラム理解を支援する初期段階のアイデアを提案している⁵⁾。本稿ではこのアイデアを詳細化した解析手法を提案するとともに、細粒度のソフトウェアリポジトリの一種である RDF リポジトリを応用し、複数の言語を併用するプログラムを解析するためのツール基盤の実現性を示す。本報告の残りの部分は以下のよう

^{†1} NTT サイバースペース研究所
NTT Cyber Space Laboratories

^{*1} 組み込みシステム分野でも Java, C, C++ が併用されるなど、他分野でも同様と考える

に構成されている。まず、2節では本研究に関連するプログラム解析ツールの相互運用性に関する既存研究、および近年のRDFリポジトリを用いたプログラム解析技術の動向を概観する。次に、3節で複数の言語やメタデータを併用するプログラムを解析するための手法を提案し、続く4節で試作したプログラム解析ツール基盤の概要を述べる。5節ではこのツール基盤を3つのオープンソースのWebアプリケーションに適用した結果を報告し、現実的なプログラムに対する適用可能性を議論する。6節は本報告をまとめと今後の研究に向けた展望を述べる。

2. 関連研究

2.1 プログラム解析ツールの相互運用性

プログラム解析ツールの相互運用を支援するためのスキーマ、メタデータは数多く提案されているが、我々の把握する限りでは、プログラミング言語やそれ以外のソフトウェア開発の成果物を容易に統合可能な性質を備えるものは存在しない。

解析ツールの相互運用に関する既存のアプローチではプログラム言語に依存しない抽象的なメタデータを定義し、そのメタデータに合わせてプログラムを解析する。Ducasseらは言語非依存のプログラム解析環境であるMooseツールキットと、オブジェクト指向言語に共通的なメタデータであるFAMIXメタモデルを定義した¹⁾。Mooseツールキット自身はSmalltalkのソースコードからFAMIXのインスタンスを生成するが、他の解析ツールがC、C++、JavaなどのソースコードからFAMIXのインスタンスを生成することにより、Moose上で分析可能となる。また、Kazmanらはアーキテクチャ回復ツールの相互運用性を実現するためのモデルであるCORUM IIを提案した。CORUM IIはソーステキスト、コード構造、機能、アーキテクチャの4段階の抽象化レイヤを定義しており、解析ツールがデータの表現形式だけではなく、その背後にある意味論を共有する⁶⁾。

OMGはCORUM IIの抽象化レイヤの考え方を取り入れ、モデル駆動アーキテクチャの基盤技術をリバースエンジニアリングに活用するアプローチであるADM (Architecture-Driven Modernization) を提唱した⁷⁾。ADMはプログラム言語ごとに特化した抽象構文のメタモデル (Abstract Syntax Tree Metamodel; ASTM) と、それらの言語に非依存なメタモデルであるKDM (Knowledge Discovery Metamodel) を定義しており、まずソースコードを解析してASTMのインスタンスを生成する。次に、各言語のASTMごとに抽象化マッピングを定義し、KDMのインスタンスを得る。ADMの実体はこれらのメタモデルのみの標準化であり、プログラム言語ごとにASTMを生成するパーサや、ASTMからKDMへ

の抽象化マッピングの開発は容易ではない。

MoDisco⁸⁾はADMの一つの実装であり、Eclipse統合開発環境に実装されたJava、JSP、XMLなどのASTMやKDM、およびそれらのインスタンスを生成するためのプログラム解析ツールを提供する。我々の開発した解析ツール基盤の考え方はMoDiscoとよく似ており、試作したツール基盤の一部でもMoDiscoの成果を利用している。しかし、ADMやMoDiscoのメタデータ基盤であるMOF (Meta Object Facility)⁹⁾では、異なるメタデータを持つモデル間を横断するリンクの作成が難しい点が課題である。MOFではメタデータが規定する規約にデータが必ず従う必要があり、リンクを作成するためにはあらかじめ複数のメタデータを併合した上で、メタデータ上の関連を追加する必要がある。フォワードエンジニアリングの場合は厳格なモデリングの規約をトップダウンに伝播させることで、人的要因によるエラーの混入を未然に防ぐ利点があるが、解析データをボトムアップかつ臨機応変に結合するようなリバースエンジニアリングの状況ではMOFは適していない。

2.2 プログラム解析におけるRDFリポジトリの応用

RDF (Resource Description Framework) はセマンティック・ウェブで用いられるデータの表現形式であり、全てのデータを主語 (Subject)、述語 (Predicate)、目的語 (Object) の三つ組みからなる文として記述する。主語は何らかのリソースを参照するURI (Uniform Resource Identifier) である。述語は主語と述語の間の関係を表すURI参照であり、主語から述語への有向辺によって表現される。目的語は日付、数字、文字列などのリテラルか、もしくは何らかのリソースを参照するURIである。同一のURI参照は複数の文の主語や目的語にもなりうるため、RDFデータの集合は有向グラフを構成する。RDFの枠組みではメタデータも文として記述するため、データと同一の有向グラフ上に表現できる。

このRDFをソフトウェアの開発に活用するアイデアが提案されている¹⁰⁾⁻¹²⁾。プログラム解析のデータをRDFのデータベースであるRDFストアに格納することにより、RDFが備える以下の特徴が活用できる。

- **グラフによるデータ構造の柔軟性。** メタデータが存在しないデータや、メタデータの規約に違反するデータであってもRDFで表現し、リポジトリに可能である。また、データの重複が許容されており、述語を用いてそれらを統合することが可能である。
- **異なるメタデータを持つデータの統合。** セマンティック・ウェブはWWW (World Wide Web) に由来する技術であり、別々の組織や個人が作成した異種のメタデータに基づくデータが混在する環境を前提としている。RDFのデータはWebにおけるハイパーリンクと同じように、任意のURIの間に参照関係を定義することができる。

- **宣言的な問い合わせを用いたツール開発の容易さ.** SPARQL は RDF で記述されたグラフ構造に対する宣言的な問い合わせ言語であり、解析ツールに必要なソフトウェア関連データを効率よく検索したり、加工したりすることを可能にする。吉田らは Java プログラムの表層構文や深層構文を RDF リポジトリに格納する手法を提案し、SPARQL を用いて CASE ツールの開発容易性が向上できると主張した¹³⁾。また、Tapplet らは SPARQL をソフトウェアの解析のために拡張した iSPARQL を提案し、過去に出題された MSR (Mining Software Repositories) ワークショップの課題を効率よく実装できることを示した¹⁴⁾。

我々は各種のプログラミング言語で記述されたソースコード、フレームワークやライブラリに特有の設定ファイル、要求や設計のドキュメントなどのように、互いに異なるメタデータを持つソフトウェアの開発成果物を相互に連携させるために、ソフトウェア開発の成果物に関連する情報を RDF リポジトリに格納する。

3. 複数の言語で記述されたプログラムの多段的な解析

前節で述べた RDF リポジトリの特徴を応用し、多言語で構成されたプログラムを解析するための手法を提案する。提案手法はプログラム解析と RDF リポジトリの更新を繰り返して漸進的に解析を進める点を特徴とし、1) 個別の言語、メタデータごとの解析、2) 言語、メタデータ間の依存性の抽出、3) 多言語で構成されたモデルの解析、の 3 種類のプログラム解析手法を併用する。

以下では説明のため、Java EE の説明用アプリケーションである JPetStore 5.0¹⁵⁾ の解析を例にとり、各ステップの解析手法を述べる。

3.1 個別の言語、メタデータごとの解析

ソースコード内に含まれるファイルやフォルダを解析し、それぞれの言語、メタデータに特化した解析ツールを用いて RDF に変換する。そのためにはファイルやフォルダの構造を認識して言語やメタデータの形式を判別する手段と、判別された言語やメタデータに従ってソースコードから RDF データを生成する解析手段を組み合わせ、言語ごとの解析規則をできるだけ多く用意しておく必要がある。

- **言語やメタデータの形式の判断.** 言語やメタデータはファイル名に付与された拡張子や、UNIX の `file(1)` コマンドの結果などで簡易的な判断ができるほか、ファイルの内容を注意深く解析して正確な形式を判定してもよい。
- **RDF データの生成.** 既存のプログラム解析ツールの多くは RDF データを出力しない。

言語に特化したツールを再実装することは非効率であるため、解析ツールが出力するデータを RDF に変換したり、解析ツールの機能をプラグイン等で拡張して RDF データの出力機能を追加するといった工夫が必要である。

JPetStore のソースコードの場合には Java, JSP, 関係データベースのスキーマを記述した SQL などの複数の言語のファイルが含まれている。それらの形式のファイルがそれぞれ `.java`, `.jsp`, `.sql` の拡張子を持つという単純な判定規則をもとに、適切な解析ツールを選択して実行する。このとき、フォルダやファイル全体を走査して、解析可能と判別されたすべてのファイルを事前に解析しておくことも可能だが、本節ではこれら 3 つのファイル形式をまず解析し、必要に応じて後から追加のファイル形式を解析する。

解析ツールはそれぞれ解析結果を RDF データのグラフ構造で出力するが、この段階ではそれらのグラフは非連結である。解析の様子を表す模式図を 図 1 に示す。JSP ファイルの解析結果は Web ページ内の要素とハイパーリンクによる画面遷移関係などを含むグラフである。Java ファイルの解析結果はパッケージ、クラス、メンバ間の所有関係、クラスの継承やインタフェースの実装による型の階層関係、変数やフィールド、メソッドのパラメータ型の型の参照関係などを含むグラフである。SQL ファイルの結果はテーブルとカラムの定義、およびテーブル間の参照制約を含むグラフである。

3.2 言語、メタデータ間の依存関係の抽出

リポジトリ内の RDF グラフへ問い合わせを行い、言語やメタデータを横断する依存性を抽出する。具体的には、プログラミング言語ごとに特有のライブラリやフレームワークの API の利用箇所を検出するための問い合わせと、検出された API に関するドメイン知識に基づいて言語やメタデータ間の依存性を生成する処理を組み合わせ、依存性の抽出規則をできるだけ多く用意し、ライブラリ化しておく。

- **プログラム外部への依存性の検出.** プログラミング言語にはそれぞれ、プログラム外部への入出力やプロセス間通信のために利用されるライブラリ、フレームワークがあり、それらに特有の API が定義されている。問い合わせによりこれらの API へ依存する箇所を検出した場合、プログラムが外部の RDF グラフに依存していると判断する。
- **依存性の抽出.** フレームワークやライブラリの知識を有する分析者がプログラムのどの箇所に着目し、どのように言語間の依存性に関するメンタルモデルを構築しているかを観察やインタビューを通じて収集する。または、フレームワークやライブラリのドキュメントやソースコードを分析して依存関係を発見する。

リスト 1 は「Java ソースコードから生成した RDF グラフ内に `org.apache.struts.Action`

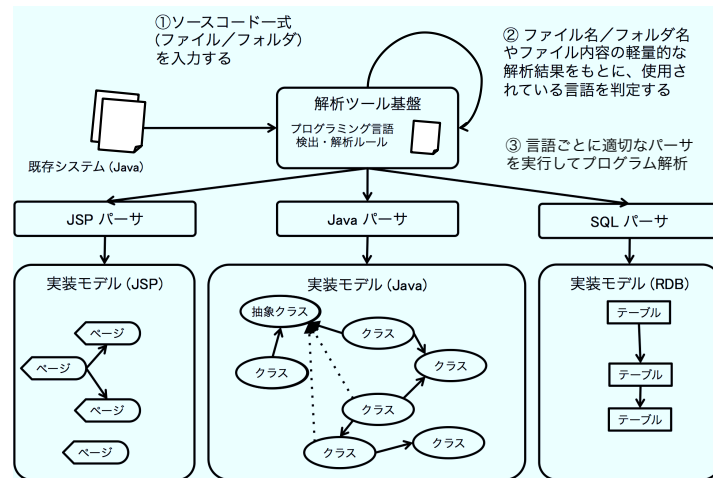


図 1 プログラミング言語ごとの検出と解析.

Fig. 1 Detection and analysis of individual programming languages.

クラスのサブクラスが存在するか」を確認するための SPARQL の問い合わせである。6, 7 行は変数 `?class` が Java のクラス宣言であり親クラス `?super` を参照すること、9, 10 行は `?super` の完全修飾名が `org.apache.struts.Action` クラスであることを問い合わせ条件として指定している。問い合わせ条件内の述語 `rdfs:subClassOf` はクラス間の継承関係を表す述語であるが、これは推移的であるため、子孫を含めた間接的なサブクラスが該当することに注意されたい。問い合わせ結果が空でなく、クラスのリストが返却された場合、そのプログラムは Struts フレームワークを利用して Web ページの遷移フローを制御していると判断する。この検出の様子を 図 2 の模式図に示す。JPetStore の RDF グラフに対してこの問い合わせを行うと該当するクラスの一覧が返却されるため、Struts フレームワークを利用していると判断できる。

ここで、Struts フレームワークに関する知識として「Action のサブクラスは画面のレイアウトを定義する JSP ファイルや、画面との入出力データを格納する ActionForm のサブクラスに対して依存性を持つ」、「それらの依存性は WEB-INF/struts-config.xml というパスに格納される Struts の設定ファイルに記述される」の 2 つを用いて、Java クラスと JSP ファイル間の依存関係を抽出する規則を適用する。まず、この Struts の設定ファイル

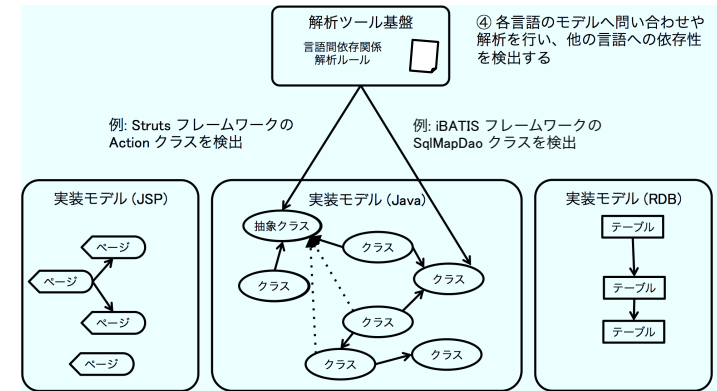


図 2 プログラミング言語間の依存性の検出.

Fig. 2 Detection of inter-language dependencies.

リスト 1 SPARQL を用いた Struts フレームワーク検出規則の実装例

```

1  PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
2  PREFIX java:<http://bonfesta.lab.ntt.co.jp/java#>
3
4  select distinct ?class
5  where {
6      ?class a java:ClassDeclaration ;
7      rdfs:subClassOf ?super .
8
9      ?super a java:ClassDeclaration ;
10     java:Type.fullyQualifiedName "org.apache.struts.Action" .
11 }

```

は RDF リポジトリ上に読み込まれていないため、XML ファイルの解析ツール用いて RDF データに変換する必要がある。次に、設定ファイルの内容を参照しながら JSP ファイル、Java クラスの間に依存関係のリンクを作成する。

リスト 2 は Action のサブクラスと、そのクラスへの入力値を提供する JSP ファイルの間の依存関係を生成するための SPARQL の問い合わせである。9, 10 行は Struts 設定ファイル内の `action` タグを指定し、12~15 行でその `action` タグの `type` 属性の値を変

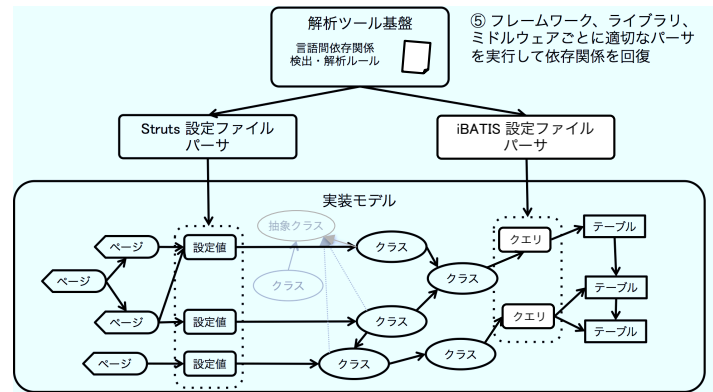


図 3 プログラミング言語間の依存性の追加.
Fig. 3 Addition of inter-language dependencies.

数 `?className` とする条件, 17, 18 行は設定ファイルに記述された `?className` の値と完全修飾名が一致する Java のクラス宣言の参照を変数 `?class` とする条件, 20~23 行は先ほどと同じ `action` タグについて, `input` 属性の値を変数 `?pageName` とする条件, 25, 26 行は `?pageName` が表すパスに存在する JSP ファイルの参照を変数 `?jsp` とする条件をそれぞれ表している. 最終的に, 7 行の `construct` 句によって問い合わせ条件により取得された `Action` のサブクラスと, そのサブクラスへ入力値を提供する JSP ファイルの関係を新しい RDF グラフの枝として生成する. なお, SPARQL の `construct` 句は実際には RDF リポジトリを更新しないため, 生成された RDF データを追加する処理が別途必要である. リスト 3 はこの抽出規則によって参照される Struts 設定ファイルの箇所を, JPetStore の例題から抜粋したものである. 抽出規則によって言語間に新たな依存性が追加される様子を 図 3 の模式図に示す.

3.3 多言語で構成されたモデルの解析

前項で追加した依存性によって連結された RDF グラフを用いてプログラム解析を行い, ソフトウェア理解や保守を支援する. 解析ツールとしては, 複数の言語に横断的な可視化のビューや, 横断的な依存関係の再ドキュメント化, 1 節で例として挙げたデータフロー図のような抽象的な設計情報の回復などが考えられる. 我々は現在, いくつかの解析ツールの開発を進めているが, 具体的なツールの事例については報告の機会を改めたい.

リスト 2 SPARQL を用いたプログラミング言語間の依存関係の追加例

```

1 PREFIX xsd:<http://bonfesta.lab.ntt.co.jp/xsd#>
2 PREFIX java:<http://bonfesta.lab.ntt.co.jp/java#>
3 PREFIX jsp:<http://bonfesta.lab.ntt.co.jp/jsp#>
4 PREFIX struts:<http://bonfesta.lab.ntt.co.jp/struts#>
5
6 construct {
7     ?class struts:Action.input ?jsp .
8 } where {
9     ?action a xsd:Element ;
10        xsd:Node.name "action" .
11
12     ?type a xsd:Attribute ;
13        xsd:Node.parent ?action ;
14        xsd:Node.name "type" ;
15        xsd:Attribute.value ?className .
16
17     ?class a java:ClassDeclaration ;
18        java:Type.qualifiedName ?className .
19
20     ?input a xsd:Attribute ;
21        xsd:Node.parent ?action ;
22        xsd:Node.name "input" ;
23        xsd:Attribute.value ?pageName .
24
25     ?jsp a jsp:Page ;
26        jsp:Page.path ?pageName .
27 }
```

リスト 3 JPetStore の Struts 設定ファイル (抜粋)

```

1 <action path="/shop/viewItem"
2     type="org.apache.struts.beanaction.BeanAction"
3     name="catalogBean" scope="session"
4     validate="false" input="/catalog/Product.jsp">
5     <forward name="success" path="/catalog/Item.jsp"/>
6 </action>
```

4. 支援ツール

我々は多言語のプログラム解析に対応したプログラム解析ツール基盤 Bonfesta を試作した。このツール基盤は図4に示すように、既存のプログラム解析ツールの出力を RDF データに変換するアダプタ群と、オープンソースで利用可能な RDF データストア Sesame¹⁶⁾ の組み合わせで構成される。

アダプタは通常、プログラム解析ツールやツールが出力するデータ形式ごとに開発する必要があるが、解析ツールが Eclipse Modeling Framework (EMF) で定義されたメタモデルのインスタンスとなるモデルを出力する場合には、そのメタモデルを OWL/RDF スキーマ形式、インスタンスとなるモデルを RDF 形式に変換する汎用の EMF-to-RDF 変換アダプタを実装した。前述の MoDisco は Java や JSP のソースコード、XML 形式の Java EE の設定ファイルを EMF で定義されたメタモデルのインスタンスへ変換する機能を持っているため、Java Web アプリケーションの RDF データは MoDisco を経由して容易に生成することが可能である。さらには、Eclipse プロジェクトの UML2 メタモデルを利用する UML エディタで編集したモデルや、EMF 上に実現された DSL ツール等のモデルを変換し、RDF リポジトリに格納することも原理的には可能である。

Java ソースコードを RDF に変換する解析ツールのスクリーンショットを図5に示す。このツールは Eclipse の Java 開発環境へのプラグインとして実装されている。Eclipse の最上位フォルダのコンテキストメニューからツールを起動すると、そのフォルダに含まれる Java ソースコード一式から MoDisco の Java モデルを生成し、前述の EMF-to-RDF 変換アダプタを通じて RDF データを得る。

SPARQL の問い合わせで取得した RDF データを用いて RDF リポジトリを更新することも可能である。3.2 項で述べた言語やメタデータを横断する依存性の追加にはこの仕組みを利用する。Sesame は Web ブラウザからアクセス可能な対話型の SPARQL インタフェースを提供しているため、クエリの編集と実行結果の確認を繰り返しながら、解析ツールのプロトタイピングを行う。完成したクエリの結果はそのまま用いるか、何らかのプログラムで RDF データを加工し、RDF リポジトリに追加する。

5. 例題への適用

試作したツール基盤が実用的に利用可能かどうかを確認するため、オープンソースで公開されている複数の Java EE Web アプリケーションを用いて処理性能を評価した。評価

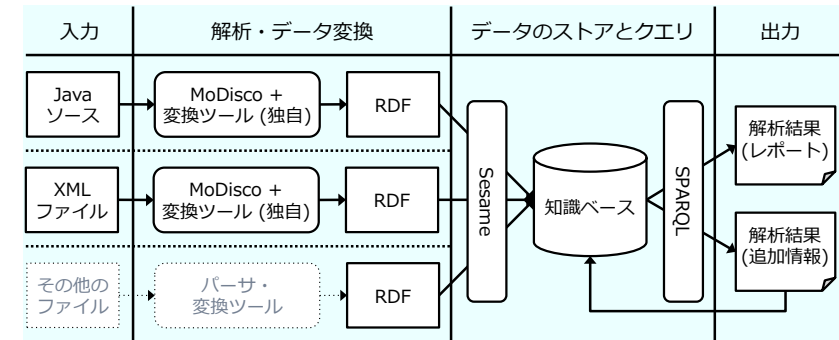


図4 試作した解析ツール基盤の全体像。
Fig. 4 An overview of the prototyped tool platform.

表1 プログラム解析の所要時間。
Table 1 Elapsed time of program analysis.

プログラム名	Step 数	RDF 文	総処理時間 (ms)	依存性抽出 (ms)	ディスク容量
JPetStore 5.0	1,705	46,594	4,570	2,170	3,795,796
MosP 給与計算 3.3.3	151,720	5,153,557	388,490	26,047	445,885,114
ADempiere 3.4.3	569,695	17,578,231	6,290,044	77,706	1,534,009,481

対象のアプリケーションは3節で例題とした JPetStore のほか、ERP の業務パッケージである MosP 給与計算サブシステム¹⁷⁾、ERP、CRM、SCM の統合業務パッケージである ADempiere¹⁸⁾ の3つを用いた。JPetStore の開発規模は約 1,700 ステップと小規模であるが、MosP と ADempiere はそれぞれ約 15 万ステップ、約 57 万ステップであり、我々が想定する商用の情報システムにおける開発規模に近い。ツールの実行環境には CPU に Intel Core i7-2620M、メモリ 8GB、80GB の SSD を備える市販のノート PC を使用し、64 ビットの Windows 7 Professional SP1 環境で 64 ビットの Oracle Java SE 6 update 25 を用いて実行した。また、Sesame の RDF ストアの種類にはネイティブストアを選択した。ネイティブストアは Sesame をデータベース管理システムとして動作させディスク入出力により RDF データを処理するため、少ないメモリ使用量で大規模なデータを保持し、問い合わせを処理できる。

各アプリケーションを RDF データに変換して RDF リポジトリへのインポートと言語間の依存性の追加処理を実行し、処理に要した時間やディスクの消費量を計測した実行結果

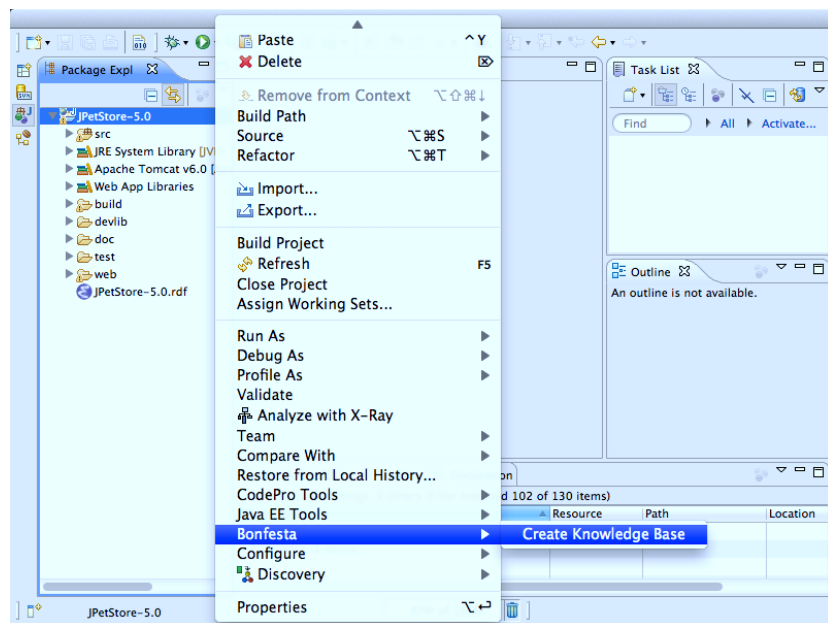


図 5 Java ソースコードの RDF への変換ツール.

Fig. 5 Our prototyped converter from Java source code to RDF.

を表 1 に示す. JPetStore のように小規模なプログラムの場合には約 5 秒で解析が終わるが, MosP, ADempiere では解析にそれぞれ約 6 分半, 17 分を要した. この処理時間のうち, メタデータを横断する依存性の抽出に要した時間はそれぞれ約 26 秒, 77 秒であり, それ以外の処理時間は MoDisco によるモデルの生成と, Sesame による RDF データのインポート処理にあたる.

これらの処理時間が現実的なソフトウェア開発において妥当かどうかはプログラム解析ツールが支援する作業によって異なるため, さらに詳細な評価が必要である. 我々はこの解析ツール基盤の適用対象として, リバースエンジニアリングの初期段階における理解の支援を考えている. この場合にはソースコードが更新されないため, 3.1 項のプログラミング言語ごとの解析は 1 回のみ行われる. RDF リポジトリへの問い合わせと更新は, 最大の開発規模を持つ ADempiere でも 1 分強で処理できることから, 事前に RDF リポジトリを構築し, 対話的に SPARQL の問い合わせを実行しながらプログラムを理解することは可能で

あると考える. 一方で, フォワードエンジニアリングや反復開発の支援に用いる場合には, ソースコードの更新ごとに RDF の再生成が必要となる. 20 分弱の解析時間は作業の中断を招くため実用的とは言えず, ソースコードの差分のみ再生成を行ったり, RDF データの生成処理を並列化するなどの対応策が必要である.

6. おわりに

本報告では RDF に基づくソフトウェアリポジトリを応用し, 複数のメタデータや情報源を横断的に利用するプログラム解析の手法を提案した. 提案手法は複数回のプログラム解析を繰り返す過程において, 前段の解析結果を後段の解析手法の選択や, 選択された解析手法への入力として利用する点に特徴がある. 我々は RDF リポジトリに基づくプログラム解析ツールの基盤を試作し, オープンソースソフトウェアの 3 つの例題へ提案手法を適用することにより, 手法の実現可能性を確認した.

提案手法はまだ, 研究の初期段階にある. 今後は提案するプログラム解析基盤を用いて, オープンソース, 商用を含めたさまざまなプログラム解析, 保守の手法を現実的なソフトウェアの解析に応用していく予定である. 同時に, 提案手法や現状のツール基盤が持つ以下の課題に取り組み, 手法の有用性を確認したい.

- **より高度なプログラム解析技術の適用と評価.** 本報告で提案した多言語からなる RDF グラフに対して, データフローや制御フローの解析を試みるなど, 既存のプログラム解析や保守の手法を応用し, ソフトウェア理解に有益な情報を得られるか検証したい.
- **動的解析結果や開発履歴の統合.** RDF は柔軟性の高いデータ構造であるため, 動的解析や履歴の解析による結果を静的なプログラム構造と関連づける形で格納できると考えている. ソースコードの一時点でのスナップショットに対する静的解析結果よりもデータ量のオーダが増えるため, 事前に解析したデータを追加すべきか, すべてのデータを追加して SPARQL で解析するかを選択のトレードオフを分析したい.
- **ドキュメントとプログラムの解析データの統合.** 要求文書や設計書などのドキュメントを解析して RDF リポジトリに追加し, 追跡性の回復手法や Feature Location 手法を適用して成果物間のリンクを追加することにより, ドキュメントとプログラムについても横断的な解析技術を開発することが可能になる. RDF への変換方法として, ドキュメントを非構造データとして自然言語処理で解析するほか, 国内の情報システム開発において頻繁に用いられる Excel ファイルについてはワークシートのテンプレート構造を利用した解析も検討したい.

参 考 文 献

- 1) Lanza, M. and Ducasse, S.: Polymetric views - A lightweight visual approach to reverse engineering, *IEEE Transactions on Software Engineering*, Vol.29, No.9, pp. 782–795 (2003).
- 2) Eisenbarth, T., Koschke, R. and Simon, D.: Locating features in source code, *IEEE Transactions on Software Engineering*, No. 3, pp. 210–224 (online), DOI:10.1109/TSE.2003.1183929.
- 3) Van Geet, J. and Demeyer, S.: Reverse Engineering on the Mainframe: Lessons Learned from "In Vivo" Research, *IEEE Software*, Vol.27, No.4, pp.30–36 (2010).
- 4) 松塚貴英, 沢田篤史, 青木利晃, 福安直樹, 妻木俊彦, 中村友昭, 浦本直彦, 羽生田栄一, 鷺崎弘宣: 「ウィンターワークショップ 2007・イン・那覇」開催報告, 情報処理学会研究報告組込みシステム (EMB), Vol.52, pp.19–25 (2007).
- 5) 風戸広史, 岡田 敏, 宮田俊介, 星野 隆: モデル駆動によるリバースエンジニアリングの支援に向けて, ウィンターワークショップ・イン・修善寺論文集, pp.9–10 (2011).
- 6) Kazman, R., Woods, S. and Carriere, S.: Requirements for integrating software architecture and reengineering models: CORUM II, Honolulu, HI, USA, pp.154–163.
- 7) Ulrich, W. M. and Newcomb, P.: *Information Systems Transformation: Architecture-Driven Modernization Case Studies*, Morgan Kaufmann Publishers Inc. (2010).
- 8) Bruneliere, H., Cabot, J., Jouault, F. and Madiot, F.: MoDisco: a generic and extensible framework for model driven reverse engineering, ASE '10, ACM, pp. 173–174 (2010).
- 9) OMG: Meta Object Facility (MOF) Core Specification, Version 2.0, ???tt <http://www.omg.org/spec/MOF/2.0/> (2006).
- 10) W3C Semantic Web Best Practices & Deployment Working Group: "Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering", <http://www.w3.org/2001/sw/BestPractices/SE/ODA/>.
- 11) Jin, D. and Cordy, J.: Ontology-based software analysis and reengineering tool integration: the OASIS service-sharing methodology, pp.613–616 (2005).
- 12) Witte, R., Zhang, Y. and Rilling, J.: Empowering Software Maintainers with Semantic Web Technologies, *The Semantic Web: Research and Applications* (Francconi, E., Kifer, M. and May, W., eds.), Lecture Notes in Computer Science, Vol.4519, Springer Berlin / Heidelberg, book part (with own title)5, pp.37–52 (2007).
- 13) 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ソフトウェア工学 (SE), Vol.147, pp.33–40 (2005).
- 14) Tappolet, J., Kiefer, C. and Bernstein, A.: Semantic web enabled software analysis, *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol.8, No.2-3, pp.225–240 (2010).
- 15) MyBatis: iBATIS JPStore 5 Sample, <http://mybatis.googlecode.com/files/JPStore-5.0.zip>.
- 16) Broekstra, J., Kampman, A. and van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema, *First International Semantic Web Conference (ISWC 2002)*, Lecture Notes in Computer Science, Vol.2342, Springer Berlin / Heidelberg, pp.54–68 (2002).
- 17) 株式会社マインド: MosP 給与計算 V3, <http://www.mosp.jp/product/payroll.html>.
- 18) Adempiere ERP Bazaar: ADempiere ERP, <http://www.adempiere.com/ADempiere.ERP>.