プログラム解析技術のサービス化の試み

Towards Program Analysis Service

桑原 寛明*　渥美 紀寿†　山本 晋一郎‡

Summary. This paper describes a prototype of program analysis service. Program analysis service provides several program analysis features, e.g. parsing, semantic analysis and flow analysis, commonly used in CASE tools as a service. We expect this service decreases the cost to construct and coordinate CASE tools. This paper also discusses problems that service providers have to tackle: the granularity of function, input/output data, data format and so on.

1　はじめに

これまでにプログラム解析技術を用いたコーディング支援や保守支援など数多くのソフトウェア開発の支援手法が提案され，CASE ツールとして実現されている．それぞれのツールが実現する支援の内容は様々であり，効果的な開発支援を実現するためには複数のツールを連携させて利用することが必要である．しかし，CASE ツールにおけるプログラム解析結果の表現方法が個々のツールに固有であったり，解析結果をツールの外部に取り出す手段がないなど，ツールの連携は容易ではない．プログラムの大規模化や解析内容の複雑化により解析に長い時間と大量のメモリを要することも多くなり，CASE ツールの開発者あるいは利用者のローカルな環境で解析を実行することが難しくなっている．

プログラム解析の基礎となる字句解析や構文解析，意味解析を行う解析器とその解析結果を利用するための API を提供するフレームワークとして，WALA [1] や Soot [2]，Sapid [3] [4] などが開発されている．ソースコードの情報を利用する CASE ツールの開発においてこれらのフレームワークを利用することで，ツール開発者は字句解析，構文解析，意味解析といった基本機能の実装を省略し，CASE ツール本来の機能の実現に専念できる．しかし，これらのフレームワークを利用するためには環境を構築する必要がある．実際にフレームワークを試用した上で開発したいツールの性質に合致するフレームワークを選択することは難しくはないがある程度の作業を要する．

本研究では，プログラム解析の基盤環境であるフレームワークのより簡便な利用を目的として，プログラム解析技術のサービス化を提案する．フレームワークが提供する機能を Web サービスとして公開し，それぞれの機能を HTTP を利用して呼び出す仕組みを用意する．これにより，フレームワークが提供する機能の手軽かつ疎結合な活用が期待できる．活用の一例として，Web サービスとして提供される機能を HTTP で結合して統合開発環境を構成する WebIDE [5] との連携が考えられる．本稿では研究の第一歩として実際に作成したサービスの具体例を紹介し，サービスとして提供する機能やその粒度について議論する．

2　プログラム解析サービス

本章ではソースコードの情報を元にコーディング支援や保守支援を行う CASE ツールを効率良く開発するためのプログラム解析サービスについて述べる．これら

*Hiroaki Kuwabara, 立命館大学情報理工学部

†Noritoshi Atsumi, 名古屋大学大学院情報科学研究科

‡Shinichiro Yamamoto, 愛知県立大学情報科学部

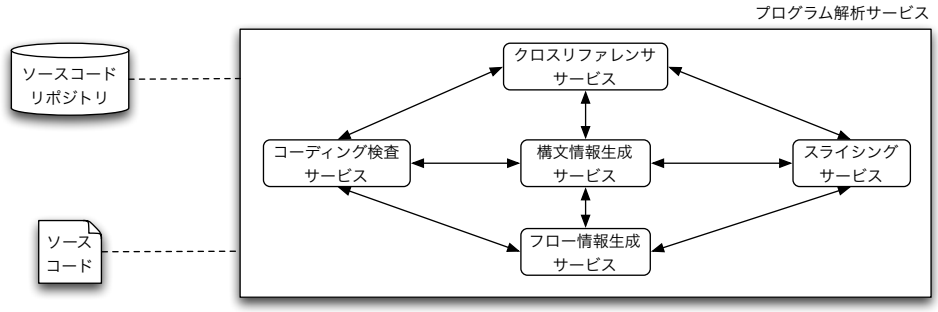

図 1　プログラム解析サービスの概要

の CASE ツールでは，字句解析，構文解析，意味解析を行うことが共通して必要となる．フロー解析や依存解析を必要とする CASE ツールも少なくない．CASE ツールの基盤となるこれらの機能を提供するフレームワークは複数存在するが，インストールなどを行って利用するための環境を構築する必要がある．そこで，基盤機能をサービスとして提供することにより，CASE ツール開発者や CASE ツール利用者は環境構築を行うことなく基盤機能を利用することが可能となる．

他のツールとの連携を容易にするため，ソースコードの解析結果を XML で表現する方法が数多く提案されている．Java 言語を対象とする JavaML [6] や JX-model [4]，C 言語を対象とする ACML [7] や CX-model [8]，複数の言語を対象とする srcML [9] などがある．解析結果を XML で表現することによって，解析結果にアクセスするための専用の API を使うことなく，DOM，SAX，XPath，XSLT，XQuery などの XML のための汎用 API を用いて CASE ツールを開発することが可能となる．

提案するプログラム解析サービスの概要を図 1 に示す．これらのプログラム解析サービスでは，ツール間あるいはサービス間の連携を容易にするため，解析結果の表現方法として XML を用いる．字句解析，構文解析，意味解析を行う構文情報生成サービスが最も基本的なサービスであり，ソースコードを与えられると，構文構造を XML の木構造で表現した XML 文書を返す．フロー情報生成サービスはソースコードを与えられると，構文情報生成サービスにソースコードを渡し，返された XML 文書を解析してフロー情報を付加した XML 文書を返すサービスである．このようにして個々のサービスが解析した結果を XML 文書として授受することによりサービス間の連携を実現する．

3　サービスの具体例

プログラム解析サービスの具体例として，いくつかのプログラム解析を Java Servlet を用いて Web サービスとして試作した．各サービスに対して URI が割り当てられており，HTTP を利用してサービスにアクセスできる．プログラム解析には CASE ツール・プラットフォームである Sapid および Sapid 上に作成されたツールを利用する．特に今回は，JavaScript 言語を対象とする JSX-model とその周辺技術 [10] を利用した．Sapid が提供する機能をできるだけそのままの形でサービス化することとし，Sapid が提供する機能の呼び出しと HTTP による入出力の仲介を行うだけの単純な Java Servlet を Google App Engine[1] 上に実装した．以下で述べるサービスとそれぞれの簡易なユーザインタフェースを `http://sapidjsx.appspot.com/` において暫定的に公開している．

[1] `http://code.google.com/appengine/`

3.1 構文情報生成サービス

構文情報生成サービスは最も基本的なサービスであり，ソースコードに対して字句解析，構文解析，意味解析を行って抽象構文木と記号表に相当する構文情報を生成する．構文情報生成サービスが生成した構文情報に基づいて他のプログラム解析サービスがより深いプログラム解析を行う．試作した JavaScript プログラムを対象とする構文情報生成サービスは，図 2 のように POST された JavaScript プログラムのソースコードに対し，レスポンスとして JSX-model に基づく XML 形式の構文情報（以下，構文情報 XML と呼ぶ）を生成して返すサービスである．POST されたデータを JavaScript プログラムとみなして Sapid の解析器で解析し，生成された構文情報 XML を返すだけの単純な実装となっている．

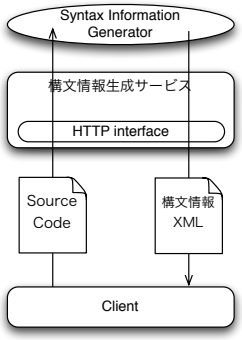

図 2　構文情報生成サービス

3.2 制御フロー解析サービス

制御フロー解析サービスは，図 3 のように POST された JavaScript プログラムに対して制御フローグラフを生成し，生成された制御フローグラフの XML 表現をレスポンスとして返すサービスである．入力として JavaScript プログラム，あるいは構文情報 XML のいずれかを受け付ける．それぞれの入力は POST される際のリクエストパラメータの名前によって区別される．JavaScript プログラムが POST された場合はサービスの内部で構文情報生成サービスを利用して構文情報 XML を取得する．構文情報 XML をバックエンドの制御フロー解析器に入力して制御フローグラフを得る．

3.3 コーディング検査サービス

コーディング検査サービスは，図 4 のように POST された構文情報 XML に対してコーディング検査を実施し，検査結果を表す XML 文書をレスポンスとして返すサービスである．このサービスの入力は構文情報 XML であるため，クライアントが構文情報生成サービスを利用して検査対象のソースコードの構文情報 XML を事前に用意する必要がある．サービスのバックエンドには Sapid を利用したコーディング検査器 [10] を利用している．

このサービスはコーディング検査を行うのみで，検査結果を表示するための整形などは行わない．そのため，サービスを利用するクライアントとしてソースコードの投入と検査結果の表示を行うユーザインタフェースが必要である．HTML と JavaScript を用いて Web ブラウザ上に構築したクライアントの例を図 5 に示す．検査結果を格納した XML 文書から取り出した情報を画面下部に表示している．

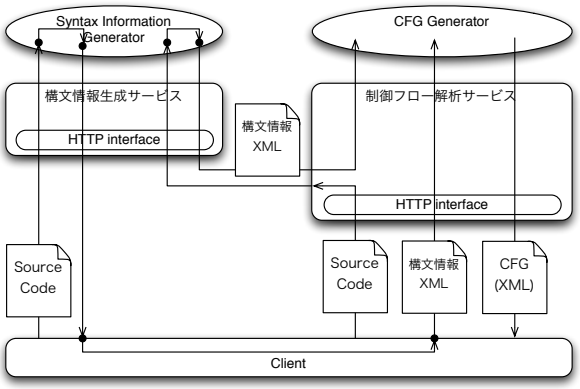

図 3　制御フロー解析サービス

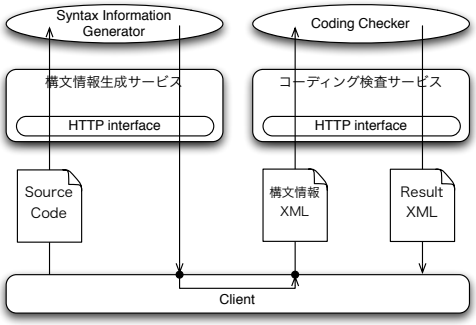

図 4　コーディング検査サービス

4　議論

プログラム解析技術のサービス化により，CASE ツール開発者は CASE ツール本来の機能の開発に注力できる．CASE ツール利用者は手元の計算機資源を用いることなく CASE ツールの機能を利用できる．サービス提供者がサービスとバックエンドの解析器の動作環境を整備することで，スケーラブルなプログラム解析の実現が期待できる．サービス化によってこれらの利点があるが，サービス化する機能の粒度やデータの受け渡しに関して課題が残る．本章ではこれらの課題について議論する．

4.1　サービスとして提供する機能

本稿で試作したサービスが実現する機能は Sapid が提供する機能の一部である．Sapid を初めとする CASE ツール・プラットフォームがコマンドあるいは API として提供する機能はいずれもサービス化が可能であるが，それぞれの機能がサービス化に適しているか否かは検討の余地がある．各機能が単体の原子サービスとして提供できる基本機能なのか，他のサービスを組み合わせた複合サービスとして提供される抽象度の高い機能なのか明らかにする必要がある．

サービス化された機能の利用時にはネットワークを介した呼び出しが発生する．また，サービスは基本的にステートレスであり処理に用いるすべてのデータを呼び出しごとに渡す必要がある．そのため，構文情報の生成やフロー情報の生成のように処理は重いが一度だけ実行すればよい機能はサービス化に適している．一方，処

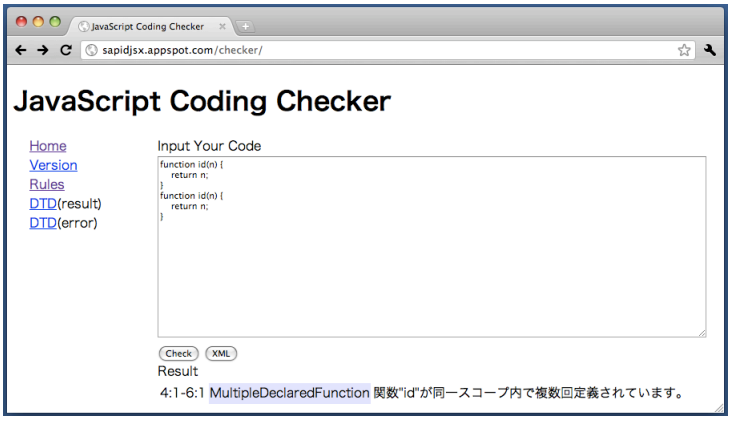

図 5　コーディング検査サービスのクライアント

理が軽く高い頻度で実行される機能や，大きなデータを必要とする上に何度も実行される機能はサービス化にはなじまない可能性が高い．

例えば，到達不能コードを発見する機能では，制御フロー解析によって生成された制御フローグラフを操作して到達不能コードを見つける．この時，サービスの提供者には以下のような選択肢がありえるが，いずれを選択すべきかは自明ではない．

1. グラフの生成のみをサービスとして提供する．
2. グラフの生成に加えて，グラフのノードを取得する機能とエッジをたどる機能をサービスとして提供する．
3. グラフの生成に加えて，到達不能なノードの発見をサービスとして提供する．

1. の場合，サービスの利用者が必要なグラフ操作を独自に実現する必要がある．柔軟な操作を実現できるが，サービスが返すグラフの表現形式を理解した上で必要な操作をすべて実装する必要があり，サービス利用者の負荷は高くなる．2. のようにグラフ操作がサービス化されていれば，サービスの利用者はグラフの表現形式を理解する必要はなく，必要な操作を行うサービスを必要なだけ呼び出せばよい．この場合，粒度の細かい多数の機能がサービスとして提供され，サービスの呼び出しごとにグラフを表現するデータの送信が発生する．サービスの利用者にとっては 3. が最も簡便であるが，必要とされる機能を事前にすべて用意することは不可能である．もし，必要な機能がサービス化されていなければ 1. の場合と同様である．

4.2　サービスの連携と入出力

サービスの入力は解析対象のソースコードや他のサービスの出力であり，サービスの出力はプログラム解析の結果である．本稿で試作したサービスでは，バックエンドの解析器による XML 形式の出力をそのままサービスの出力としている．XML には，構造の表現が容易，要素名や属性を用いた意味の表現が可能，DOM を初めとする XML 文書を操作する API が整備されており様々な環境で利用可能，といった利点がある．一方で，XML 文書のスキーマやサイズに関して課題が存在する．

サービスを接続する際，前段のサービスが出力する XML 文書から後段のサービスの入力として適切な XML 文書を得るためにスキーマ変換が必要となる可能性がある．スキーマ変換の技術を含め，サービスを効率的に接続するための技術が必要である．特に，既存のプログラム解析フレームワークは独自の内部表現やスキーマを中心に構成されており，他のフレームワークとの連携は考慮されていないことが多い．提供者の異なるサービスの接続は容易ではない可能性が高い．

XML 文書はサイズが大きくなる傾向があり，ネットワークを介して呼び出されるサービスの入出力データの表現形式に適しているか評価する必要がある．例えば，

Java 言語を対象とする JX-model では構文情報 XML のサイズがソースコードのおよそ 10 倍になる [4]．JavaScript 言語を対象とする JSX-model ではおよそ 10 倍から 20 倍であり，ライブラリである prototype.js（バージョン 1.7）[2]の場合，約 160KB のソースコードに対して約 2.9MB の構文情報 XML が生成される．サービスによってはこのような数 MB のデータが呼び出しごとに送信あるいは受信される．入出力データのサイズがパフォーマンス上の障害にならないか評価する必要がある．

プログラム解析フレームワークでは，専用の API を通して内部的に構築されて解析に活用され，最終的に永続化されずに破棄される揮発的なデータ構造が利用される．このようなデータ構造の生成機能と，データ構造を利用する解析機能を個別にサービス化する場合，データ構造の表現形式を新たに定義し，その形式に基づく入出力機構を用意する必要がある．本稿の制御フロー解析サービスで用いた Sapid の JavaScript 向け制御フローグラフも揮発的なデータ構造であり，サービスの実装時に制御フローグラフの XML による表現形式を定義して入出力機構をバックエンドに追加した．サービス化によって既存のプログラム解析フレームワークが想定していないポイントで入出力が発生することがある．

5 おわりに

本稿では，プログラム解析技術のサービス化を提案し，解析サービスのプロトタイプを実装した．サービス化により，複数のプログラム解析技術の手軽かつ疎結合な連携やスケーラブルなプログラム解析の実現が期待できる．しかし，サービスとして提供される機能やその粒度，サービスとクライアントの間あるいはサービス間で受け渡されるデータに関して検討を重ねる必要がある．サービスを利用した CASE ツールの開発と実運用に向けた評価，サービスの充実化も今後の課題である．

謝辞 本研究の一部は文部科学省科学研究補助費 (若手 B:課題番号 21700042) による助成を受けた．

参考文献

[1] IBM. The T. J. Watson Libraries for Analysis (WALA). `http://wala.sourceforge.net`, 2011.

[2] Raja Vallée-Rai, Laurie Hendren, Vijay Sundaresan, Patrick Lam, Etienne Gagnon, and Phong Co. Soot - a java optimization framework. In *Proceedings of CASCON 1999*, pp. 125–135, 1999.

[3] 福安直樹, 山本晋一郎, 阿草清滋. 細粒度リポジトリに基づいた CASE ツール・プラットフォーム Sapid. 情報処理学会論文誌, Vol. 39, No. 6, pp. 1990–1998, 1998.

[4] 吉田一, 山本晋一郎, 阿草清滋. XML を用いた汎用的な細粒度ソフトウェアリポジトリの実装. 情報処理学会論文誌, Vol. 44, No. 6, pp. 1509–1516, 2003.

[5] Ken-ichi Nakatani, Takayuki Omori, and Katsuhisa Maruyama. A Programming Environment Consisting of Web Services. In *Proceedings of 14th IASTED International Conference on Software Engineering and Applications*, pp. 460–467, 2010.

[6] Greg J. Badros. JavaML: A Markup Language for Java Source Code. In *Proceedings of the 9th International World Wide Web Conference on Computer Networks*, pp. 159–177, 2000.

[7] 川島勇人, 権藤克彦. XML を用いた ANSI C のための CASE ツールプラットフォーム. コンピュータソフトウェア, Vol. 19, No. 6, pp. 21–34, 2002.

[8] 渥美紀寿, 山本晋一郎, 小林隆志, 阿草清滋. CASE ツール・プラットフォームのための C ソースプログラムの XML 表現とその応用. ソフトウェアエンジニアリング最前線 2010, pp. 135–142, 2005.

[9] Michael L. Collard. Addressing source code using srcML. In *IEEE Int. Workshop on Program Comprehension Working Session: Textual Views of Source Code to Support Comprehension*, 2005.

[10] 桑原寛明, 末次亮, 山本晋一郎, 阿草清滋. 拡張可能な JavaScript 言語向けコーディング検査器. ソフトウェア科学会第 27 回大会 (6C-1), 2010.

[2] `http://prototypejs.org/`