論文

UNICOEN: 複数プログラミング言語対応のソースコード処理フレームワーク

UNICOEN: A Unified Framework for Code Engineering Supporting Multiple Programming Languages

概要

近年，プログラミング言語の多様化とソフトウェア開発を支援するソースコードの解析および変形ツールの開発が進んでいる．しかし，これらの既存ツールの多くは 1 つのプログラミング言語を対象として開発されているため，プログラミング言語とツール間に多対多の関係があり，全ての言語とツールの組み合わせに対して実装した場合，非常に莫大なコストが必要である上，ツール毎に実装や仕様に差異が存在していて，複数のプログラミング言語で開発されたソフトウェアに適用しづらい問題がある．

本論文では，上述の問題を解決するために，複数のプログラミング言語に対応するソースコード処理フレームワーク UNICOEN を提案する．UNICOEN は言語非依存な言語モデルを提供することで，ツールの開発コストおよび言語の拡張コストを削減して，ツール間の差異を低減する．我々は，UNICOEN 上で開発した 7 種類のプログラミング言語に対応する 3 種類のツールを評価して，その有用性を確認した．

Abstract

Programming languages become more multifaceted and many analysis and transform tools for source code are being developed. However, high development costs are required to implement all combinations between programming languages and tools and there are differences of implementations and specifications in the tools because these tools support only one programming language.

In this paper, we propose a framework for processing source code supporting multiple programming languages named UNICOEN. UNICOEN reduces development costs and prevents differences of implementations and specifications between tools. Conclusively, we evaluated UNICOEN developing 3 tools which supports 7 programming languages.

1. はじめに

近年，様々なプログラミング言語（以降，言語）が開発され，その多様化が進んでいる．例えば，1972 年に開発された歴史のある C 言語は現在でも広く普及している．その一方で，Kotlin や Xtend 言語など新しい言語が次々と生まれている．

様々な言語が存在する中で，ソースコードを処理するツール（以降，ツール）が存在する．ツールの処理内容は，大きく分けてソースコードの解析と変形の 2 つに分けられる．ソースコードを解析するツールとして，メトリクス測定ツールや静的解析ツール，また，ソースコードの解析に基づいて変形するツールとして，ソースコード整形ツールやアスペクト指向プログラミング（Aspect Oriented Programming; AOP）処理系が挙げられる．これらのツールは，ソフトウェアの品質や開発効率を向上させるツールとして注目を浴びている[1)]．

現在，多くのツールが言語毎に開発されている．例えば，静的解析ツールの FindBugs[2)] は Java 言語のみに，JSLint[3)] は JavaScript 言語のみに対応している．また，AOP 処理系の AspectJ[4)] は Java 言語のみに，AOJS[5)] は JavaScript 言語のみに対応している．このように，言語とツールの間には多対多の関係がある．例えば，各ツールが 1 つの言語のみに対応していて，全てのツールと言語の組み合わせを考えた場合,「言語の種類数」×「ツールの種類数」通りの実装が必要になる．しかし，言語とツールの間の多対多の関係は，以下で示す 2 つの問題点を引き起こす．問題 1：全てのツールが全ての言語に対応するために莫大な開発コストが必要である点，問題 2：対応言語が異なる同じ種類のツール間に差異が生じる点である．そのため，優れた言語やツールが存在するにもかかわらず，使用する言語によってはその恩恵が十分に得られない．

本論文では，以上の問題点を解決するために，UNICOEN（UNIfied source COde ENgineering framework）[⋆1]を提案する．UNICOEN は，解決 1：言語非依存な統合コードモデルを提供して，解決 2：ソースコードを統合コードモデル上のオブジェクトにマッピングする．さらに，解決 3：統合コードモデル上の汎用的な共通処理を 2 種類の API で提供することで，ツールの開発コストを低

⋆1 UNICOEN は IPA の 2010 年度未踏 IT 人材発掘・育成事業の支援を受けて開発した．その成果が認められ，著者らを含む開発メンバーは経済産業省および IPA から未踏スーパークリエータの認定を受けた．

減して，ツールの利用者が自由に言語を選べるように促す．

我々は，UNICOEN 上で 7 言語の対応を実装して，その上で，3 種類のツールを開発した．言語対応の拡張とツール開発コストが既存ツールと比較して少なく，開発したツールが言語間の差異を低減するため，問題 1 と問題 2 を緩和できることを確認した．

本論文の主要な貢献は次の 3 点である．1) 7 種類の言語仕様から共通／相違点を整理した和集合を考えて統合コードモデルを設計した点，2) UNICOEN に言語対応を追加する利用者と UNICOEN 上でツールを開発する利用者向けに二種類の API を提供した点，3) 実際に UNICOEN に 7 種類の言語対応を追加して，3 種類のツールを実装して評価した点．

2. 既存ツールの問題点

2.1 問題 1：莫大な開発コスト

言語とツール間に多対多の関係があるため，全てのツールが全ての言語に対応するために必要な開発コストが莫大である．そのため，ある言語のみに対応したツールが開発されてから，長い時間を経て他の言語向けにツールが移植されたり，一部の言語に対して対応するツールが存在しないケースがある．したがって，ツール開発者は対応言語を拡張するために多大な労力を割く必要がある．また，ツール利用者は，開発プロジェクトが採用する言語によってはツールの恩恵を受けられなかったり，ツールを利用するために採用可能な言語の選択肢の幅が狭まるといった問題が生じている．

例えば，C 言語に対応した静的解析ツール Lint[6] が 1977 年にリリースされてから，JavaScript 言語に対応した同様のツール JSLint が 2002 年に，Python 言語に対応した同様のツール Pylint[7] が 2004 年にリリースされている．なお，JavaScript と Python 言語はそれぞれ 1995 年，1990 年にリリースされている．しかし，Ruby 言語に対応した同様のツールは我々が調査した限りでは存在していない．ツール利用者がツールのサポートを受けられる言語は一部のみであり，サポートが広がるまでに長い期間を要する．

2.2 問題 2：ツール間の差異

異なる言語に対応した同じ種類のツールを組み合わせる場合，ツール間の差異のために期待する結果が得られなかったり，組み合わせるために追加のコストが必要になったりする．そのため，複数の言語を利用するプロジェクトで複

数のツールを組み合わせても，これらの言語で記述されたソースコードを対象としてツールを適用できない場合がある．

例えば，テストカバレッジ測定ツールEMMA[8]はJava言語に，Code coverage measurement for Python[9]はPython言語に対応している．これらのツールを用いてステートメントカバレッジを測定する際，前者はJava VM上の命令単位で，後者はソースコードの行単位で測定する．サーバークライアントモデルにより，サーバーサイドをJava言語，クライアントサイドをPython言語で開発したソフトウェアを対象とする場合，これらのツールの測定基準が異なるため，測定結果を統合して総合的に評価することが難しい．

また，AOP処理系のAspectJはJava言語に，AOJSはJavaScript言語に対応している．同様にサーバーサイドをJava言語で，クライアントサイドをJavaScript言語で開発するソフトウェアにおいてAOPを利用する場合，JavaとJavaScript言語のソースコードに対して，それぞれアスペクトを記述しなければならない．アスペクトのモジュール性が低下する上，アスペクトの記述方法はAOP処理系毎に異なるため，ツール利用者は各記述方法を学ぶ必要がある．全てのメソッド実行時にロギングを行うAspectJとAOJSのアスペクトの例をそれぞれ図1の1から4行目と6から9行目で示す．両者のアスペクトが同じ内容を示しているのにも関わらず，記述方法が大きく異なることが分かる．

```
1 public aspect Logger {
2   pointcut all() : execution(* *.*());
3   before() : all() { System.out.println(thisJoinPoint.getSignature() + " is executed."); }
4 }
5
6 <?xml version="1.0" ?>
7 <aspectsetting><function functionname="/*" pointcut="execution">
8   <before><![CDATA[console .log ( __name__ + " is executed ."); ]]></before>
9 </function></aspectsetting>
```

図1 メソッドの実行に対するロギングするAspectJのアスペクトとAOJSのアスペクト

Fig. 1 A logging code for executing methods in AspectJ and AOJS

3. UNICOENの全体像

UNICOENの全体像を図2で示す．UNICOENは統合コードモデルを中核に

据え，汎用的な共通処理として，対応言語拡張者向け API とツール開発者向け API を提供する．UNICOEN はソースコードを構文解析して抽象構文木上で解析や変形をするツールの開発を支援する．また，UNICOEN はシンタックスに基づいてソースコードを構造化するため，特に，シンタックスに着目したり意味解析を完全に行わないようなツールの開発を助ける．

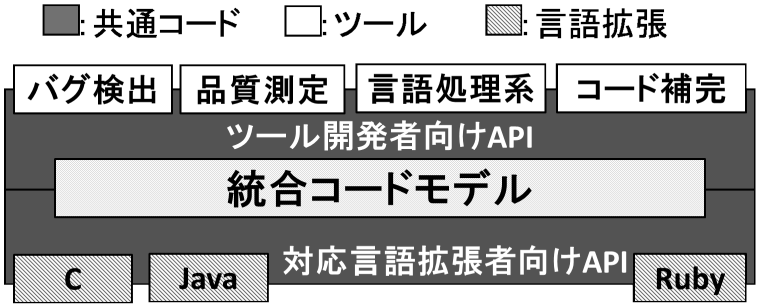

図 2　UNICOEN の全体像

Fig. 2　An overview of UNICOEN

UNICOEN は C# 4.0 で開発されており，.NET Framework および Mono 上で動作するフレームワークである．UNICOEN は，Apache2.0 ライセンスを適用したオープンソースソフトウェアであり，Github からソースコードをダウンロードできる[10]．

UNICOEN は共通な抽象構文木（AST）の仕様として，解決 1：言語非依存な統合コードモデルを提供する．ソースコードから得られる言語非依存な抽象構文木のインスタンスを統合コードオブジェクトと呼ぶ．統合コードモデル上でツールを開発することで，問題 2：対応言語が異なる同じ種類のツール間に差異が生じる点を解決する．UNICOEN は，ツール開発者向け API として，W3C が標準化した DOM と類似した機能を提供する．具体的には，同一言語内でソースコードと統合コードオブジェクトの相互変換，および，統合コードモデル上で解析や変形を行うため要素の抽出・追加・変更・削除機能を提供する．

UNICOEN がツール開発者向け API を提供するために，ソースコードと統合コードオブジェクトを相互に変換する Object-code mapper（以降，OC マッパー）を言語毎に実装する必要がある．そのため，UNICOEN は，対応言語拡張者向け API として，OC マッパーの実装に必要な汎用的な共通処理を提供することで，対応する言語の拡張を支援する．UNICOEN は，解決 2：ソースコー

ドを統合コードモデル上のオブジェクトにマッピングして，解決 3：統合コードモデル上の汎用的な共通処理を 2 種類の API で提供することで，ツールの開発コストと対応言語の拡張コストの両方を低減する．以上から，問題 1：全てのツールが全ての言語に対応するために莫大な開発コストが必要である点を解決する．

UNICOEN の利用者は，ツール開発者向け API を利用してツールを開発するツール開発者と，対応言語拡張者向け API を利用して UNICOEN に対応言語を追加する対応言語拡張者の 2 種類に分けられる．ツール開発者は，統合コードモデル上でソースコード処理を記述することで，構文解析や一部の意味解析の処理の実装を省くことができる．さらに，言語非依存な抽象構文木と共通のツール開発者向け API を利用して，UNICOEN が対応するどんな言語に対しても，同じようなコードを記述してツールを開発できる．一方，対応言語拡張者は，OC マッパーを開発することで，UNICOEN が扱える言語の幅を広げられる．

4. UNICOEN の詳細

本節では，統合コードモデル，対応言語拡張者向け API，ツール開発者向け API の 3 つに分けて UNICOEN の詳細を説明する．

4.1 統合コードモデル

UNICOEN は，様々な言語で記述されたソースコードを構造化する統合コードモデルをクラスとして提供する．統合コードオブジェクトは統合コードモデル上のインスタンスであり，木構造を有している．そのため，クラスを表す統合コードオブジェクトは，子要素としてメソッドを表す統合コードオブジェクトを持ち，さらにその子要素として引数やブロックなどを表す統合コードオブジェクトを持つといったように，再帰的な構造を持つ．

統合コードモデルは主に構文解析に基づいてソースコードをオブジェクトとして構造化するため，完全な意味解析を必要としない．例えば，UNICOEN は二項式の構文を認識するが，演算子の意味（例えば，+ 演算子が加算か文字列結合か）まで解釈しない．したがって，UNICOEN は GCC や LLVM のようなコンパイラフレームワーク，Java VM や.NET Framework のような中間言語の実行環境とは異なる．これらの既存ソフトウェアとの詳細な相違点は 6 節にて示す．UNICOEN は，ソースコードを構造化することでツールの開発コストを削減して，意味解析を省くことで対応言語の拡張コストを低減する．

我々は，C，Java，C#，Visual Basic，JavaScript，Python，Ruby の 7 種類の言語について，それぞれの文法の和集合を取ることで統合コードモデルを設計した．各言語の意味論に基づいて類似する構文を共通要素として，それ以外を異なる要素として和集合を取った．

例えば，while 文の統合コードモデルを考える．多くの言語において while 文は，ループの継続を判定する条件式，ループ内で実行する命令の 2 つの要素から構成される．しかし，Python 言語では while 文に else 節を持ち，ループの継続を判定する条件式が偽になった際に，ループを抜ける直前で実行する命令を持つ．したがって，和集合を取った結果，while 文の統合コードモデルは，ループの継続を判定する条件式，ループ内で実行する命令，ループを抜ける直前で実行する命令の 3 つの要素を持つように設計した．また，Java の package 宣言と C#の namespace 宣言は構文の見た目が異なるものの，構文が表す意味が等しいことに着目して，等しく名前空間を宣言する統合コードモデルとして扱っている．さらに，いくつかの言語では名前空間に直接メソッドやフィールドの定義を記述できるため，名前空間やクラス，インタフェースの宣言は全て特殊なブロックの宣言として扱っている．

Abstract Syntax DescriptionLanguage（ASDL）を用いて記述した統合コードモデルの定義の一部を図 3 に示す[⋆1]．一般に，多くの手続き型言語では式（エクスプレッション）と文（ステートメント）を評価値を持つか否かで区別することが多い．しかし，Ruby 言語は，他の言語で文として扱われている構文の多くを式として扱っており，文はほとんど存在しない．例えば，Ruby 以外の 6 種類の言語では，if 文や while 文，関数定義は文であるが，Ruby 言語ではすべて式である．そこで，我々は文を式の特殊なケースと考えて，両者の区別をなくした．したがって，統合コードモデルでは文と式をどちらも式として扱う．このように和集合を取って定義した統合コードモデルを利用することで，対応する 7 種類の言語で記述された任意のソースコードを統合コードモデル上で表現可能である．

4.2 対応言語拡張者向け API

UNICOEN は，対応言語の拡張コストを低減するために，対応言語拡張者向け API として，ANTLR[11] で構文解析した結果を.NETFramework 上の XML ツ

⋆1 完全な定義は https://github.com/UnicoenProject/UNICOEN/blob/master/ASDL.txt で閲覧できる．

```
Expression = If(Expression condition, Block body, Block elseBody)
    | While(Expression condition, Block body, Block elseBody)
    | DoWhile(Expression condition, Block body)
    | For(Expression initializer, Expression condition, Expression step,
        Block body, Block elseBody)
    | FunctionDefinition(ModifierCollection modifiers, Type returnType,
        Identifier name, ParameterCollection parameters, Block body)
```

図 3　ASDL を用いて記述した統合コードモデルの定義の一部
Fig. 3　The part of unified code model in ASDL

リーのオブジェクトとして表現する機能，XML ツリーのオブジェクト上を走査する機能を提供する．対応言語拡張者は，対応言語拡張者向け API を利用して，OC マッパーを構成する構文解析部，オブジェクト生成部，コード生成部の 3 つの機能部を実装する．UNICOEN はツール開発者向け API となるインタフェースを提供するため，そのインタフェースに従って OC マッパーを実装しなければならない．OC マッパーの利用例として，ソースコードと統合コードオブジェクト間の相互変換の様子とコードを図 4 と図 5 で示す．

UNICOEN はソースコードから得られた統合コードオブジェクトについて，変換元と同じ言語のソースコードにセマンティクスを変えずに逆変換できることを保証する．対応言語拡張者はこの性質を満たすように OC マッパーを開発しなければならないため，UNICOEN は上記の性質を保証していることを確認するためのテストコードを提供する．さらに，変換元の言語が持つ機能と構文の範囲で，統合コードオブジェクトを変更しても，コンパイル可能な状態でその変更を反映したようなソースコードに逆変換できることを保証する．しかし，その範囲を越えて変更した場合や，機能や構文が異なる別の言語のソースコードへ変換する場合は，UNICOEN はその変換が正常に行えることを保証しない．なお，変換先の言語の OC マッパーの開発者が個別に仕様を決めるため，例えば，クラス定義の統合コードオブジェクトに対して，構造体や関数ポインタを組み合わせた C 言語のソースコードに変換するような OC マッパーを開発することができる．

OC マッパーの実装方法は対応言語拡張者に委ねられているが，UNICOEN は ANTLR や既存のパーサーライブラリを利用して構文解析部を実装することを支援する．我々は，ANTLR を用いて OC マッパーを容易に実装できるように，UNICOEN のサブプロジェクトとして開発した Code2Xml を提供する[12]．

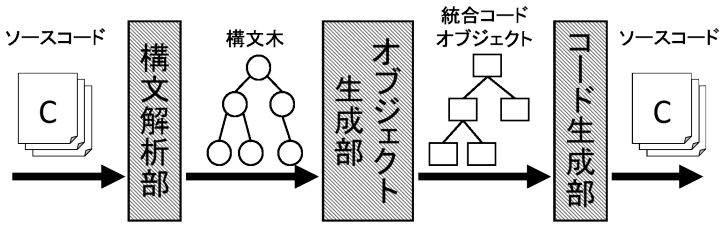

図 4 ソースコードと統合コードオブジェクトの相互変換

Fig. 4 A process of conversion and reverse conversion between source code and unified code objects

```
1 var filePath = "code.java";
2 var ext = Path.GetExtension(filePath);
3 var progGen = UnifiedGenerators.GetProgramGeneratorByExtension(ext);
4 var uco = progGen.GenerateFromFile(filePath);
5 // 必要に応じて統合コードオブジェクトに対する解析や変形処理を記述
6 var code = progGen.CodeGenerator.Generate(uco);
```

図 5 UNICOEN を利用してソースコードと統合コードオブジェクトを相互変換する C#コード

Fig. 5 A code to inter-convert between source code and unified code objects in C# with UNICOEN

Code2Xml は ANTLR が生成したパーサーを構文解析部として利用できるように自動修正する．さらに，オブジェクト生成部の実装を支援するために，XML の走査や式の解析処理などの共通処理を提供する．

以下に対応言語を拡張する手順を述べる．1) 拡張対象の言語仕様を調査して，ソースコードの構造化に必要なモデルを考える．2) 既存の統合コードモデルと考案したモデルを比較して差異を調べる．3) 得られた差異に対して，以下の手順 a,b のいずれかもしくは両方で統合コードモデルを拡張する．3.a) 対応する概念が統合コードモデルに存在しない場合は，それを表現するためのクラスを定義する．例えば，アスペクトという概念を統合コードモデルに追加する場合は，アスペクトに該当するクラスを定義する．3.b) 既に統合コードモデル上に対応する概念が存在していても既存のクラスで表現できない場合は，そのクラスにプロパティを追加する．例えば，Python の while 文における else 節のような概念が for 文にも存在する場合は，for 文を表現するクラスに else 節のプロパ

ティを追加する．4) 対象の言語に対して拡張した統合コードモデル上でマッピングする OC マッパーを実装する．

なお，統合コードモデルを拡張しても，既存の OC マッパーやツールは影響を受けない．統合コードモデルにクラスを追加した場合，そのクラスのインスタンスは既存の OC マッパーからは生成されないが，ソースコード中に該当する構文が現れなかったのか，該当する構文がその言語に存在しないのか区別できない．同様に，プロパティを追加した場合，そのプロパティは既存の OC マッパーでは null に初期化され，統合コードオブジェクトが該当する要素を持たなかったのか，該当する要素がその言語に存在しないのか区別できない．

4.3 ツール開発者向け API

UNICOEN は，様々な言語向けのツールの開発コストを低減するために，ツール開発者向け API として，ソースコードと統合コードオブジェクトを相互変換する機能と，統合コードモデル上で要素を抽出・追加・変更・削除する機能を提供する．これらの機能は，対応言語拡張者が各言語について OC マッパーを実装することで実現される．図 6 で統合コードモデルとツール開発者向け API のクラス図を示す．

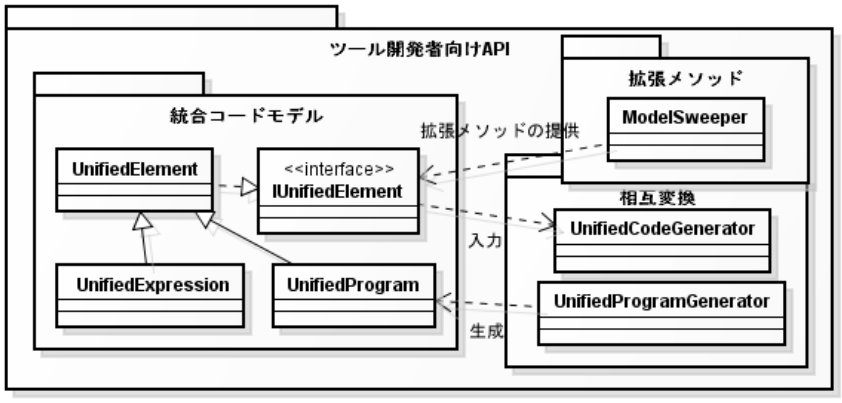

図 6　統合コードモデルとツール開発者向け API のクラス図

Fig. 6　A class diagram of the unified code model and the API for tool developers

統合コードオブジェクトの走査や変形の機能は，LINQ to XML に類似した API で提供しており，LINQ と非常に高い親和性を持つ操作系を提供する．例えば，図 7 の 1 から 3 行目と 5 から 7 行目までのように，LINQ to XML を

用いて if という名前の要素ノード数を表示する C#言語で記述されたコードと，ツール開発者向け API を用いて if 文の数を表示するコードは非常に似ている．図 7 の変数 ifElements と ifs の型は，それぞれ IEnumerable⟨XElement⟩ と IEnumerable⟨IUnifiedElement⟩ である．XElement は XML の要素ノード，IUnifiedElement は統合コードオブジェクトの要素を表す．どちらも IEnumerable を利用しており，LINQ が提供する拡張メソッド Count() を利用できる．

```
1 var xml = XDocument.Load("code.xml");
2 var ifElements = xml.Descendants("if");
3 Console.WriteLine("The number of if elements: " + ifElements.Count());
4
5 var uco = UnifiedGenerators.GenerateProgramFromFile("code.java");
6 var ifs = uco.Descendants<UnifiedIf>();
7 Console.WriteLine("The number of if: " + ifs.Count());
```

図 7 if の要素ノード数と UNICOEN を利用して if 文に該当する統合コードオブジェクト数を表示する C#コード

Fig. 7 A code to enumerate 'if' elements and enumerate unified code objects of 'if' statement in C#

ツール開発者が，ソースコード解析ツールを開発する場合は，ソースコードから統合コードオブジェクトを生成した上で，統合コードオブジェクトに対する解析処理を記述する．ソースコード変形ツールを開発する場合は，ソースコードから統合コードオブジェクトを生成して，さらに，統合コードオブジェクトを変形して，その上で，統合コードオブジェクトからソースコードを再生成する処理を記述する．例えば，統合コードオブジェクト上でステートメント数を測定する場合，ブロックの直下の子要素がステートメントに該当する統合コードオブジェクトなので，ブロックに該当する要素を全て抽出して，その直下の子要素の数を数えれば良い．図 8 でステートメント数を測定するコードを示す．

このように，ツール開発者向け API を利用して，統合コードオブジェクトを構成する要素を走査することで，任意の構文を容易に数え上がることができる．また，メソッド定義を追加したり，任意の演算子を変更したり，任意のステートメントを削除するために，統合コードオブジェクトが持つプロパティをの値を書き換えたり，Add や Remove メソッドを利用して子要素を追加，変更，削除する処理が容易に記述できる．このように，ツール開発者向け API を利用して

```
1 var uco = UnifiedGenerators.GenerateProgramFromFile("code.java");
2 var count = uco.Descendants<UnifiedBlock>().Sum(e => e.Count);
3 Console.WriteLine("The number of statements: " + count);
```

図 8　UNICOEN を利用してステートメント数を測定する C#言語コード
Fig. 8　A code to count statements in C# with UNICOEN

構造化したオブジェクト上で解析処理や変形処理を実装することで，様々な言語を対象としてソースコードを処理するツールの開発コストを大幅に削減する．

5. 評　　価

5.1　対応言語の実装

我々は C，Java，C#，Visual Basic，JavaScript，Python，Ruby の 7 種類の言語の OC マッパーを実装した．実装した OC マッパーと既存の言語処理系のステートメント数の比較を表 1 に示す．表中の OC マッパー項目は，人手で実装したオブジェクト生成部のステートメント数を表す．一方，既存の言語処理系として GCC の C コンパイラ，GCC の Java コンパイラ GCJ，Mono の C#コンパイラ MCS，Java VM 上で動作する JavaScript 処理系の Rhino，.NET Framework 上で動作する Python 処理系の IronPython と Ruby 処理系の IronRuby のコンパイラ部分を比較対象とした．なお，Rhino 以外の処理系はフレームワークの共通処理を除いて，ソースコードをマシン語もしくは中間言語に変換する処理に該当するソースコードのみを対象とするが，Rhino はフレームワークを利用していない上，コンパイラ部分を分離できないため，処理系全体のステートメント数を提示する．また，Visual Basic 処理系のソースコードがなかったため，表 1 には記載していない．

C，Java，JavaScript 言語では ANTLR を利用したが，それ以外の言語では既存のパーサーライブラリを利用した．UNICOEN は既存のソフトウェアを活用して，さらに，言語拡張者向け API を提供することで，既存の言語処理系と比較して実装すべきコード量を 20 倍から 50 倍程度も削減した．以上から，UNICOEN が問題 1 を緩和できることを確認した．

5.2　ツールの実装

5.2.1　メトリクス測定ツール UniMetrics

我々は UNICOEN を用いて McCabe の複雑度や CodeCity[13] が測定結果を街

表 1 実装した OC マッパーと既存の言語処理系のステートメント数の比較
その他処理系：GCC, GCJ, Mono, Rhino, IronPython, IronRuby

Table 1 A comparison of the number of statements between OC mappers and existing programming language processors: GCC, GCJ, Mono, Rhino, IronPython, IronRuby

言語	C	Java	C#	JavaScript	Python	Ruby
OC マッパー	727	1,003	399	626	636	501
既存の言語処理系	14,949	12,782	36,988	38,277	15,411	14,353

として可視化可能なメトリクスを測定する UniMetrics を開発した．

McCabe の複雑度を測定可能な既存ツールに，Java 言語向けの Sonar[14] と Ruby 言語向けの Saikuro[15] が挙げられる．しかし，これらの測定ツールの測定基準には拡張 for 文の扱いに相違点が存在する上，両者の測定結果を 1 つにまとめて表示できない．したがって，Java と Ruby 言語を用いて開発したソフトウェア全体の複雑度を測定して評価することは困難である．一方，UniMetrics は，同じ測定基準を用いて複数の言語における McCabe の複雑度を測定でき，測定結果を 1 つのグラフにまとめて表示する．また，CodeCity には Java，C++，C#言語向けのメトリクス測定ツールがそれぞれ存在するが，複数の言語で開発されたプロジェクトから 1 つのメトリクス測定結果を生成して，可視化することができない．UniMetrics は，UNICOEN が対応する言語で記述されたソースコードを対象とでき，例えば，Java と JavaScript 言語で開発された JsUnit のメトリクス測定結果を生成できる．

McCabe の複雑度を測定する処理のステートメント数について，Saikuro と UniMetrics の比較結果を表 2 に示す．また，CodeCity 用の測定結果を生成する処理のステートメント数について，C#言語のみに対応する PMCS[16] と UniMetrics の比較結果を表 3 に示す．

以上から，UNICOEN が言語間のツールの差異を低減することで問題 2 を緩和でき，さらに，ツールの開発コストを低減することで問題 1 を緩和できるを確認した．

5.2.2 アスペクト指向プログラミング処理系 UniAspect

我々は複数言語に対応する AOP 処理系 UniAspect を UNICOEN を用いて開発した．UniAspect では，ポイントカットを言語非依存に記述でき，アドバイ

表 2　既存ツール Saikuro と UniMetrics における対応言語と測定処理のステートメント数による比較

Table 2　A comparisons of the supported languages and the numebr of statements between Saikuro and UniMetrics

	対応言語	ステートメント数
Saikuro	Ruby	321
UniMetrics	C，Java，C#，Visual Basic，JavaScript，Python，Ruby	3

表 3　既存ツール PMCS と UniMetrics における対応言語と測定処理のステートメント数の比較

Table 3　A comparisons of the supported languages and the numebr of statements between PMCS and UniMetrics

	対応言語	ステートメント数
PMCS	C#	1478
UniCodeWorld	C，Java，C#，Visual Basic，JavaScript，Python，Ruby	203

スを織り込み先の言語毎に記述する．UniAspect は 1 つのアスペクトを複数の言語のソースコードに適用できるため，利用者の学習コストを低減して，より良いモジュール化を実現する．図 1 のアスペクトについて，UniAspect を用いて 1 つのアスペクトに記述した結果が図 9 である．

```
1 aspect Logger {
2   pointcut allMethod() : execution(* *.*());
3
4   before : allMethod() {
5     @Java       { System.out.println(JOINPOINT_NAME + " is executed."); }end
6     @JavaScript { console.log(JOINPOINT_NAME + " is executed.");        }end
7   }
8 }
```

図 9　メソッドの実行に対するロギングする UniAspect のアスペクト

Fig. 9　A logging code for executing methods in UniAspect

UniAspect によるアスペクトが織り込まれる処理の流れは次のとおりである．

1. プログラム全体に該当する統合コードオブジェクトから関数に該当する要素

を抽出する．2. 得られた関数の中から，関数名や戻り値の型を参照して，ユーザが指定した条件に当てはまる関数に絞り込む．3. 絞り込んだ関数が持つブロックの先頭に，指定したコードに該当する統合コードオブジェクトを挿入する．上記のアスペクトの織り込み処理は，UNICOEN が提供するツール開発者向け API を用いて言語非依存に実装できる．以上から，UNICOEN が複数言語に対応する統一的なツールの開発を支援して，問題 1 を緩和できることを確認した．

6. 関 連 研 究

Lattner ら[17] はコンパイラフレームワーク LLVM を提案した．類似したソフトウェアとして，Java VM や.NET Framework など中間言語の実行環境と GCC が挙げられる．これらのソフトウェアは，完全に意味解析をして中間言語に変換するため，対応言語を追加するコストが非常に大きい．さらに，中間言語はマシン語に近い非常に抽象度の低い仕様であり，多くの場合，シンタックス情報を得られない．そのため，コード整形などシンタックス情報を必要とするツール開発が困難である．また，評価の項目で実装したようなソースコードの処理ツールは完全な意味解析を必要としない．一方，UNICOEN は意味解析を完全に行わないことで，対応する言語の追加およびツールの開発コストを削減する．UNICOEN は統合コードモデルと開発者向け API を提供することで，どの言語に対しても同じようにツールを開発できる．また，UNICOEN では，必要に応じて統合コードモデル上で意味解析を行うことで，ツールに必要な処理を実装できる．したがって，意味解析を完全に必要とはしないツール開発を対象としたとき，UNICOEN は既存ソフトウェアより優位性がある．

Higo ら[18] は複数のプログラミング言語に対応したメトリクス測定フレームワーク MASU を提案した．MASU はメトリクス測定の観点から必要なセマンティクスを解釈して，言語非依存な抽象構文木を構築する．言語非依存な抽象構文木を走査することでメトリクスを測定でき，MASU のプラグインとして測定処理を実装することで，MASU を通して Java, C#, Visual Basic 言語のソースコードのメトリクスを測定できる．一方，UNICOEN は MASU が対応する言語全てに対応している上，オブジェクト指向プログラミング言語以外や動的型付け言語にも対応する．また，MASU は言語非依存な抽象構文木の提供に留まっているが，UNICOEN は対応言語拡張者向け API を提供することで対応言

語の追加を支援する．その上，ソースコードの解析処理のみならず，変形処理も言語非依存に記述可能である．

7. ま と め

本論文では，複数言語対応のソースコード処理フレームワーク UNICOEN を提案した．UNICOEN は，ソースコードを統合コードモデル上のオブジェクトにマッピングして，汎用的な共通処理をツール開発者向け API および言語拡張者向け API として提供することで，構文解析を行い抽象構文木上でソースコードの解析や変形をするツールの開発を支援する．我々は，UNICOEN に C，Java，C#，Visual Basic，JavaScript，Python，Ruby 言語の対応を追加して，その上で，これらの言語に対応したメトリクス測定ツール，CodeCity の言語対応の拡張，AOP 処理系を開発した．そして，対応言語の拡張において既存の言語処理系と比較して大幅に少ない記述量で実装可能であること，さらに，意味解析を完全に行わないようなツールの開発において，統合コードモデルとツール開発者向け API を利用することで実装に必要な記述量を大幅に低減できること，また，開発したツールが複数言語に対応した統一的な機能を提供して，複数存在するツール間の差異を低減することを確認した．以上から，UNICOEN が問題 1 および問題 2 を緩和できることを確認した．

今後の展望として，次のような点が挙げられる．現状で UNICOEN が対応する言語は手続き型言語のみとなっており，Haskell や OCaml のような関数型言語の対応を実装していない．そこで，UNICOEN にこれらの言語を対応させることで，統合コードモデルが手続き型言語のみならず関数型言語においても有用であることを示す必要がある．

また，現状では，UNICOEN の対応言語を拡張する際，人手で OC マッパーを開発する必要がある．さらに，新たな言語を拡張する際に，場合によっては統合コードモデルを人手で拡張する必要がある．このような作業は既存の処理系において対応言語を拡張するよりも容易であると考えられるが，ソースコードから統合コードオブジェクトへのマッピング記述から OC マッパーを自動生成する処理系や，ASDL のような統合コードモデルの仕様記述から統合コードモデルに該当するクラスを自動生成する処理系の開発が考えられる．

References

1) Zheng, J., Williams, L., Nagappan, N., Snipes, W., Hudepohl, J.P. and Vouk, M.A.: On the Value of Static Analysis for Fault Detection in Software, *IEEE Trans. Softw. Eng.*, Vol.32, pp.240–253 (2006).

2) Hovemeyer, D. and Pugh, W.: Finding bugs is easy, *SIGPLAN Not.*, Vol. 39, pp.92–106 (2004).

3) Crockford, D.: JSLint,The JavaScript Code Quality Tool, ,
available from ⟨http://www.jslint.com/⟩ (accessed 2012-02-04).

4) Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W.G.: An Overview of AspectJ, *Proceedings of the 15th European Conference on Object-Oriented Programming*, ECOOP '01, London, UK, UK, Springer-Verlag, pp.327–353 (online),
available from ⟨http://dl.acm.org/citation.cfm?id=646158.680006⟩ (2001).

5) Washizaki, H., Kubo, A., Mizumachi, T., Eguchi, K., Fukazawa, Y., Yoshioka, N., Kanuka, H., Kodaka, T., Sugimoto, N., Nagai, Y. and Yamamoto, R.: AOJS: aspect-oriented javascript programming framework for web development, *Proceedings of the 8th workshop on Aspects, components, and patterns for infrastructure software*, ACP4IS '09, New York, NY, USA, ACM, pp.31–36 (2009).

6) Johnson, S.C.: Lint, a C Program Checker, *COMP. SCI. TECH. REP*, pp.78–1273 (1978).

7) Logilab: pylint 0.25.1 : Python Package Index, ,
available from ⟨http://pypi.python.org/pypi/pylint⟩
(accessed 2012-02-04).

8) Roubtsov, V.: EMMA: a free Java code coverage tool, ,
available from ⟨http://emma.sourceforge.net/⟩ (accessed 2012-02-04).

9) Batchelder, N.: coverage 3.5.1 : Python Package Index, ,
available from ⟨http://pypi.python.org/pypi/coverage⟩
(accessed 2012-02-04).

10) Sakamoto, K., Ohashi, A., Ota, D., Iwasawa, H. and Kamiya, T.: UnicoenProject/UNICOEN - GitHub, The UNICOEN Project (online),

available from ⟨https://github.com/UnicoenProject/UNICOEN⟩
(accessed 2012-02-04).

11) Parr, T.J. and Quong, R.W.: ANTLR: a predicated-LL(k) parser generator, *Softw. Pract. Exper.*, Vol.25, pp.789–810 (1995).

12) Sakamoto, K.: code2xml, ,
available from ⟨http://code.google.com/p/code2xml/⟩
(accessed 2012-02-04).

13) Wettel, R., Lanza, M. and Robbes, R.: Software systems as cities: a controlled experiment, *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, New York, NY, USA, ACM, pp.551–560 (2011).

14) SonarSource: Sonar, , available from ⟨http://www.sonarsource.org/⟩
(accessed 2012-02-04).

15) Blut, Z.: Saikuro : A Cyclomatic Complexity Analyzer, ,
available from ⟨http://saikuro.rubyforge.org/⟩ (accessed 2012-02-04).

16) Doernenburg, E.: erikdoe / PMCS / overview - Bitbucket, ,
available from ⟨https://bitbucket.org/erikdoe/pmcs/⟩
(accessed 2012-02-04).

17) Lattner, C. and Adve, V.: LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation, *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, CGO '04, Washington, DC, USA, IEEE Computer Society, pp. 75– (online),
available from ⟨http://dl.acm.org/citation.cfm?id=977395.977673⟩ (2004).

18) Higo, Y., Saitoh, A., Yamada, G., Miyake, T., Kusumoto, S. and Inoue, K.: A Pluggable Tool for Measuring Software Metrics from Source Code, *Proceedings of the 2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, IWSM-MENSURA '11, Washington, DC, USA, IEEE Computer Society, pp.3–12 (2011).