プログラミング教育のための多言語間プログラミング言語翻訳システム

松澤 芳昭[1,a)]　坂本 一憲[2]　大畑 貴史[3]　筧 捷彦[4]

概要：Computational Thinking(CT) の育成を目指したプログラミング教育において，その教育目的は特定のプログラミング言語の習得ではないが，学習過程では学習段階や目的に適合した特定の言語が必要となる．そこで本研究では，多プログラミング言語間の翻訳システムを応用開発することで，教育現場で様々な言語を選択できる，特定の言語に依存しないプログラミング教育環境を提案する．テキスト型言語だけでなく，ビジュアル型言語や日本語プログラミング言語にも対応し，学習者，指導者，または学習目的の多様性に対応するプログラミング教育環境を目指している．現在，メタ言語構文の設計と実装が完了し，Java-ブロック型ビジュアル言語間の相互変換システムが新プラットフォーム上で動作している．今後，多言語対応を進めるとともに，ソフトウェアメトリクス等の教育研究に役立つデータの出力 API を提供することによって，教育研究者を支援するインフラストラクチャを設計していく計画である．

1. はじめに

IT 技術の進歩を背景として，一般情報教育におけるプログラミング教育は影を潜めていたが，近年国内外でプログラミング教育が再評価されている．英国では，操作偏重教育への反省からプログラミング教育が再評価され，プログラミング教育の復活が議論されている [1]．米国では，Computational Thinking[2](以下 CT) と名付けられた「世の中の様々な問題について，抽象化，自動化の観点からモデルを組立て，解決するスキル」の概念が提唱され，21 世紀の知識社会に生きるすべての人に必要なスキルとして議論が展開している．我が国で平成 25 年度に閣議決定された「世界最先端 IT 国家創造宣言」では，初等中等教育でのプログラミング教育の重要性が言及されている [3].

CT を問題解決能力育成を指向したプログラミング教育と解釈すれば，その学習環境の研究は S.Papert の LOGO 研究 [4] 以来 30 年以上行われてきた．近年は計算機の速度向上により，プログラミング学習のためのグラフィカルな開発環境が実用速度で動作する時代になり，特に 2000 年以降種々の環境が提案され，実際の教育現場で使われるようになった．A.Kay の Squeak[5] をはじめとする Scratch[6] などのビジュアルプログラミング言語環境，言霊 [7] などの日本語プログラミング言語環境，ドリトル [8] などの初学者用オブジェクト指向プログラミング言語，PEN[9] などプログラムの動作を可視化する環境などが挙げられる．

その一方で，既存の教育環境（または開発環境）は言語に強く依存しており，言語を横断して運用できないという課題がある．例えば，PEN に実装されたビジュアルデバッガは有用であるが，言語が xDNCL に制約されるため，選択の機会が限定される．

教育に利用される言語は教師の嗜好にも制約を受けるため，学習者にとって最適な言語が選択されていない可能性もある．最新のプログラミング教育研究成果では，教育段階によって学習者が必要な言語が異なることが明らかになっている．松澤らは Java と Block(ビジュアル型) 言語の相互変換システムを開発し，学習者に利用しやすい言語を選択させた所，Block から Java へシームレスに移行していくことを明らかにしている [10].

言語の相互運用の難しさに起因する問題は，研究分野の発展も妨げている．プログラミング教育環境の研究では言語による記述能力や開発環境の相違が大きいという理由で比較研究が難しい．近年のプログラミング教育環境改善は提案が行われるものの簡易な評価しか行われておらず，学術上の進化が滞っている．相互運用環境の整備による精緻な比較研究がのぞまれる．

[1] 青山学院大学社会情報学部
Aoyama Gakuin University
[2] 国立情報学研究所
National Institute of Informatics
[3] 静岡大学大学院情報学研究科
Shizuoka University
[4] 早稲田大学
Waseda University
[a)] matsuzawa@si.aoyama.ac.jp

本研究では，これらの問題を解決するために，多プログラミング言語 (以下,「多言語」と略す) 間の翻訳システムを開発して，プログラミング言語に非依存なプログラミング教育環境基盤の構築を目指す．この基盤技術として，複数のプログラミング言語に対応したコード処理フレームワーク UNICOEN[11] を利用する．提案基盤は，教育及び解析環境から言語を分離するアーキテクチャを採用して，言語に非依存な環境の開発を支援する開発基盤も提供する．

本稿は本研究の基本コンセプトの説明と中間報告を目的とするものであり，全５章からなる．続く２章では，プログラミング教育用の言語についての先行研究を解説する．第３章では，本研究の目的を整理し，本研究の主張であるプログラミング教育基盤の基本設計を述べる．第４章で，現在実装が完成している言語翻訳，解釈部分にについて，その詳細設計，結果を報告する．第５章はまとめである．

2. 先行研究

過去 30 年以上のプログラミング教育研究において，言語の選択は大きなテーマであり続けてきた．ACM の SIGCSE メンバによるサーベイでも，”Language Choice”は４つあるプログラミング教育研究のテーマの一つになっている [12]．このサーベイにおいては，言語選択の要因の例として，faculty preference（教師の嗜好），industry relevance（産業界との関連, 実用性），technical aspects of the language（技術的側面），the availability of useful tools and materials（有用なツールや教材の利用可能性）をあげている．既存の言語からこれらの要素を全て満足する言語を選択するのは難しい．なぜなら，これらの要因は相反する場合が多いためである．例えば Java は 2000 年代で多くの産業界，教育現場が採用したが，入門教育での利用には多くの障害があった [13]．プログラミング教育者はこれらの要素を考慮しながら，教育目的に沿った言語の選択をする必要があった．

新しい言語の開発の過程で，どの言語が教育現場に最適なのか比較する研究も行われてきた．例えば，西田らは PEN の有用性を示すために，同様の教育を Javascript で行った結果との比較を行っている．しかしながら，教育現場では厳密な比較統制実験は難しいため，例えば実施した大学が異なることなどを理由に，これは決定的な結果とは言えない．Lewis らは，同様のカリキュラムを Logo と Scratch で実施する実験を行っている [14]．この研究での統制環境は妥当なものと考えられる，しかし，実際のパフォーマンスであまり差が出ておらず，これも明確な結論を示すことには成功していない．これらの研究の課題には ComuputationalThinking を要求する課題も含まれているが，最終的にはまだ文法事項の理解の比較の議論にとどまっていることも課題である．Alice と Scratch の比較研究 [15] が述べるように，目的によって使い分けるべきという結論までがこの分野で得られているコンセンサスである．

近年では，学習者の段階によって言語を切り替えるべきである，という主張もいくつか試みられている．例えば，Bieniusa らの研究 [16] では，scheme での入門教育が後の Java のオブジェクト指向プログラミング教育に役立ったデータを示している．カーネギーメロン大学では伝統的に，3D アニメーションを作成することに強みを持つ Alice という言語が使われており，Dann らは Alice での入門教育が Java の教育に役立ったという主張をしている [17]．松澤らは OpenBlocks[18] を改良した Block 言語と Java の相互変換機能を用いて，プログラミング入門教育の受講者たちがビジュアル型言語から Java に自然に移行するデータを示している [10]．

計算機性能の向上によって，実用速度で動く言語変換器が比較的容易に開発できるようになり，実開発・教育現場双方で言語変換器が実用的に使われている．例えば，現在 Web プログラミングで主流の言語は Javascript であるが，型がないことに不便を感じるユーザには CoffeeScript, TypeScript などの Javascript 互換の言語が用意され，Javascript に変換するアプローチが取られている．スマートフォンで主流の iPhone の開発では 2014 年に Swift と呼ばれる新言語が発表されたが，全てのライブラリも含めて従来の Objective-C のリソースが利用できるようになっており，交ぜ書きと相互変換ができるようになっている．教育現場では，先にあげた松澤らの研究以外に，google 社の PencilCode も双方向の Visual-Text 言語変換機能を主張している．Blockly[19] はビジュアルからテキストへの単方向であるものの，Javascript, Python など複数の言語への出力機能を持っている．オブジェクト指向教育開発環境で著名な BlueJ[20] は，現在 Oracle 社がスポンサーとなり Greenfoot と改名し開発が続けられている．2015 年度版の Greenfoot では，Java 言語に基づく構造ビジュアルエディタ機能が提案されている．その目的はこれらの研究と同様であり，ビジュアルー Java 相互変換機能を用いて，初期学習の足場かけを行うことである．

プログラミング教育言語選択の究極的な解決案として，自分で言語を作る所から始める，または自分で言語を作り変えられる環境であるべきという提案を，パーソナルコンピューティングの提唱者である Kay 氏が行っている．Kay 氏による STEPS と呼ばれる NSF(National Science Foundation) に採択されたプロジェクトでは，主要な成果として OMeta[21] が開発された．OMeta は言語の定義を記述するメタ言語を提供して，言語変換機能の実現を支援する．Warth らは OMeta を応用して，TileScript と呼ばれるブロック型言語と Javascript の相互変換システムも試作している [22]．しかし，相互変換の技術的検証にとどまっており，教育現場で利用可能なブロック編集システムの提案と評価は行われていないことは課題である．OMeta を利用して

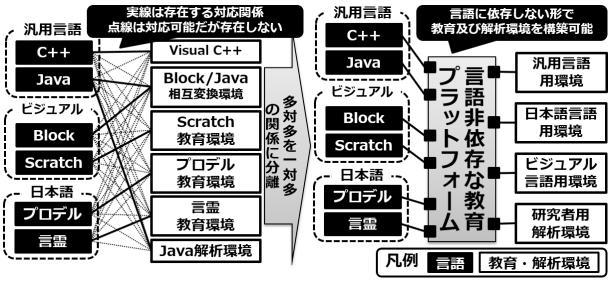

図 1 提案プラットフォームによる問題解決
Fig. 1 Problem solving by developing the proposed platform.

KScript という Functional Reactive Programming(FRP) の処理系を約 700 行で記述し，Squeak 上にリッチテキストエディタを約 1 万行で書くデモンストレーションが行われている [23]．この例は学習者が言語を開発したわけではなく，開発者自身によるデモンストレーションであり，学習者自身が言語の定義を行う環境の構築，または言語の選択システムは未完である．しかし，開発環境自身が複数言語で読める記述，かつ小さなサイズに収斂していることは，開発環境を読む機会を学習者に提供する上で教育上の価値がある．

技術的には，複数のプログラミング言語に対応したコード処理技術 UNICOEN[11] が実用になったことで，当該技術を利用した多言語間のプログラミング言語翻訳システムの可能性が得られた．UNICOEN は，プログラミング言語の抽象表現を提供して，言語非依存なツール開発を支援する基盤で，静的解析などのソースコード処理 [24] を可能にしている．ソースコード翻訳への応用についても，特定の組込みシステムに特化することで，C 言語とフローチャート間での翻訳が UNICOEN を利用して既に実現している [25]．UNICOEN を利用しても現在使われている主要な全ての言語の全ての機能の相互変換を開発するのは，直感的には容易ではないことが考えられる．しかしながら，対象の機能をプログラミング入門教育に絞ることで，実用的かつ拡張性のあるシステムが可能であると我々は試算している．

3. 提案システムの目的と基本設計

3.1 目的

本研究の研究テーマは，多プログラミング言語間の翻訳・解釈プラットフォームを開発して，プログラミング言語に非依存なプログラミング教育環境基盤の構築を行うことである．提案プラットフォームによる問題解決の様子を図示し，**図 1** に示す．提案プラットフォームは，教育及び解析環境から言語を分離するアーキテクチャを採用して，言語に非依存な環境の開発を支援する開発基盤を提供する．

本研究には，2 つの大きな目標がある．1 つめの目標は，教育者・学習者に対するもので，言語と教育・解析環境の分離により，教育で使用する言語選択の制約を取り除くとともに，学習者と教育者は目的や学習段階に応じて，使用する言語の選択を可能にすることである．同僚の学習者が異なる言語を利用しても良いし，学習者が記述する言語と，教師が評価する言語が異なっても良い．プログラミング入門の教師はフローチャートによる表現を好む．その時，説明のためのフローチャートをプログラムから自動生成することもできるし，学習者がフローチャートで記述したものをそのまま動作ターゲットの言語に変換して動作を確認することもできる．現在は C 言語を利用しているが，ブロックエディタのような新しい環境も段階的に導入したいと考えている教育者などのニーズに応えることもできる．プログラミング教育の文脈，例えば Web システム，組込みシステム，サーバーサイドなど，の選択は学習者の動機付けを得るため等に重要であるが，文脈と言語を分離して，最良の文脈と言語を選択できる可能性も広がる．本環境では，プログラムが表現されている言語は単なる一時的な View となる．従って，プログラミング教育からアルゴリズムを分離して，「書きたい言語で書き，読みたい言語で読む」環境を学習者に提供することが可能になる．

2 つめの目標は，教育開発環境の開発者に対するもので，開発者は少ない実装コストで新しい教育環境や対応言語，学習状況の解析機能の追加を可能にすることである．プログラミング教育では，言語だけではなく，プログラミング教育用の IDE(Integrated Development Environment) の開発 (例えば，Dr.Java [26]，Dr.Scheme [27]) , やビジュアルデバッガの開発（例えば，Jeliot3 [28], JGrasp [29], PEN [9]) によって学習を支援する試みがされてきた．しかし，これらのツールは特定の言語に強く依存しており，複数の言語への対応が高コストであることから，移植は試みられてこなかった．近年ではプログラミング教育版 Learning Analytics と呼ぶべき，学習データ分析ツール（例えば，Programming Process Visualizer [30]，Cocoviewer [31]）が開発されてきており，今後も高度な学習記録分析が行われていくことが期待される．しかし，これらも特定の言語に依存しているため，移植が難しく，普及が阻害されていた．本プラットフォームの開発目標は，言語変換に対応する言語の追加，および IDE, ビジュアルデバッガ, Learning Analytics ツールへの容易な拡張性を持つことである．

3.2 基本設計

本研究で提案するプログラミング教育環境プラットフォームの全体像を**図 2** に示す．

プラットフォームは大まかに，「多言語間言語翻訳システム」および，「教育解析環境の開発基盤」からなる．「言語翻訳システム部」開発の基礎技術として，派生版 UNICOEN を開発する．UNICOEN の長所は既に多くの言語に対応するプロセッサを備えており，幅広い能力を持つ中間言語へ

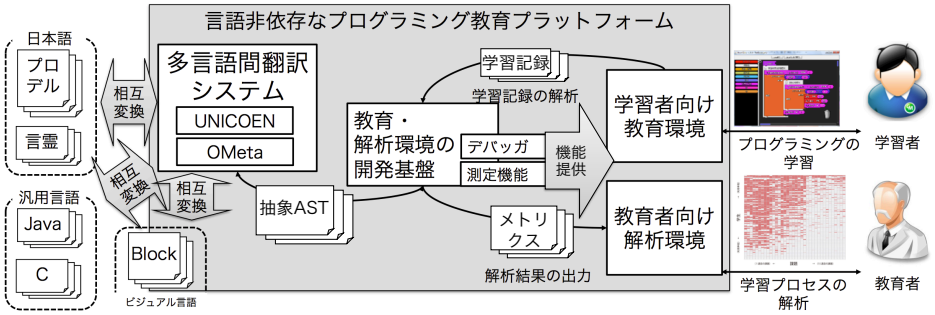

図 2 提案プラットフォーム全体のアーキテクチャ
Fig. 2 The overall architecture of the proposed platform.

マッピング可能になっている．しかし，追加の言語対応に関しては，プログラミング言語毎に実装する必要がある点と，言語翻訳システムに利用する場合，言語によって異なる記述能力の違いをどのように吸収し，中間言語を設計するかという点が課題となる．オリジナルの UNICOEN は言語の意味解析については考慮していないため，各文法の和集合 (Union) をとって解析結果を処理している．しかし，本フレームワークでは翻訳が必要なため，基本的には，基礎文法部分についての共通部分 (Intersection) をとって解析処理を行う方針で新しい抽象 AST を設計する．しかし，静的型の処理など，共通部分とはならないが必要と考えられる文法事項については抽象 AST に含み，静的型付けをしない言語ではアノテーションなどをつけるようにして変換する．

「教育解析環境の開発基盤」については，言語によって異なるコンパイラ処理系，デバッガなどを抽象化し，プラグイン化し低コストで組み込めるように設計した上で，なんらかの VM(Virtual Machine) 上で様々な言語から変換されたプログラムを動作させるように設計している．本環境は，多言語を並行的に学習者が利用できる API を提供する．初学者用ビジュアルデバッガ，研究者に必要な学習者の操作ログの取得機能 [30] について，本フレームワーク用に移植する．メトリクス測定に関しては，UNICOEN によって既に実装されているので，メトリクスの履歴取得と可視化環境を提供する．

4. プロトタイプの開発：ブロックー Java 相互変換システムの再実装

提案するプラットフォームのうち，「多言語間言語翻訳システム」部のプロトタイプ開発を行った．例題として，ブロックー Java 相互変換とタートルグラフィックスライブラリを利用したプログラミング教育環境 [10] を用い，これと同等の開発環境を新しいプラットフォームで再現することを目標とした．

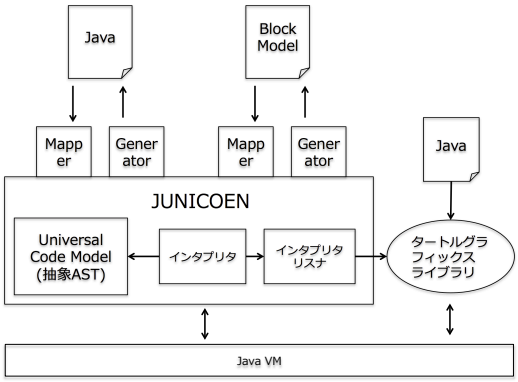

図 3 プロトタイプの設計
Fig. 3 The design of the prototype implementation.

4.1 設計

プロトタイプ実装の設計図を**図 3** に示す．

UNICOEN 部で扱うコードは抽象コードモデルであるが，実際に動作するソフトウェアの設計においては，UNICOEN 自体を動かすホスト言語の選択が必要である．本プロトタイプでは我々はホスト言語に Java を用い，ターゲットマシンとして Java Virtual Machine を想定した．再現の対象となるシステムがすべて Java で書かれていたため，資産の再利用がしやすいことと，JVM がマルチプラットフォーム上で安定して高速に動くことを考慮しての選択結果である [*1]．オリジナルの UNICOEN は C #で実装されており，今回の Java 実装版は JUNICOEN と名付けられた．

JUNICOEN は，クラスライブラリとして定義された Universal Code Model(以下，Uni-Model) と，インタプリタ部からなる．Uni-Model は現在，当該カリキュラムで利用する Java の文法に対応した，概して Java 文法のサブセットとして設計されている．インタプリタは Uni-Model を直接，解釈実行するモジュールである．今回はホスト言

[*1] ただし，Java はセキュリティ上の問題が深刻になってきており，Web ブラウザで動く Javascript エンジン上で動作することが，教育現場でも必須の条件となってきている．そのため，今後はインタプリタ部の Javascript 版を開発し，ライブラリを開発して Web 上で直接動作する実装も検討している．

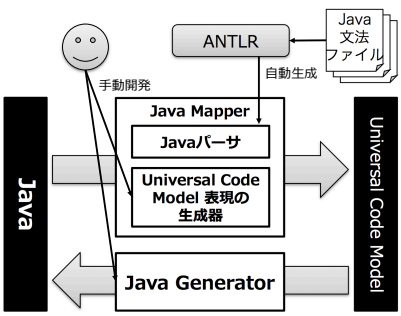

図 4 Java Mapper と Generator の設計
Fig. 4 The design of the Java Mapper and Generator.

語として Java を利用しているため，Java に変換し，Java コンパイラでコンパイルしたバイトコードを動かすことも可能である．しかし，その方法では，抽象レイヤで動作するビジュアルデバッガが開発できないため，直接インタプリタで実行するモジュールが設計されている．インタプリタリスナは，抽象実行部とネイティブ言語で直接記述されたライブラリを接続し，ライブラリを利用するプログラムを動作させるモジュールである．この設計によって，ユーザプログラムを抽象レイヤで実行しつつ，タートルグラフィックスのようなネイティブライブラリコールを伴うプログラムが実現できる．

本プロトタイプ実装では，Java およびブロック言語へのそれぞれ Mapper,Generator が部分的に実装されている．Mapper はゲスト言語から Uni-Model への変換，Generator はその逆変換モジュールを指す．この部分に関しては，対応する言語毎に個別実装をする必要があるが，できる限り安価なコストで実装できるように設計されている．

4.2 JUNICOEN ー Java 相互変換システム

本節では Java のソースコードと Uni-Model 表現の相互変換を実現するシステム（Java 変換システム）について説明する．図 4 で Java 変換システムの設計を示す．既に述べたとおり，Java 変換システムは Java Mapper と Java Generator から構成され，さらに，Java Mapper は Java パーサと Uni-Model 表現の生成器から構成されている．Java パーサは ANLTR[32] を用いて自動生成しており，Uni-Model 表現生成器と JavaGenerator は我々が手作業で開発した．ANLTR はパーサジェネレータであり，GitHub 上に公開されている Java 言語用の文法ファイル [*2] を入力して，Java パーサを生成した．

Uni-Model 表現生成器は，ANTLR が生成した Java パーサが出力する具象構文木から JUNICOEN の抽象構文木（Uni-Model 表現）を生成する．ANTLR が生成するパーサが出力する具象構文木は，Visitor パターンで操作できるように設計されている．したがって，Uni-Model 表現生成器は ANTLR が提供する Visitor クラスを拡張することで，出力される具象構文木の構造を読み取り，対応する JUNICOEN の Uni-Model 表現を構築する．以降で，if 文を例に，Java Mapper と Java Generator の実装について説明する．

図 5 で if 文に関連のある Java 文法ファイルの断片を示す．文法ファイルは ANTLR が独自に定めた Extended Backus-Naur Form（EBNF）ベースの言語で記述できる．1-7 行目で `statement` 非終端記号を定義しており，`statement` は if 文に該当する `ifThenStatement` および `ifThenElseStatement` 非終端記号によって構成されている．`ifThenStatement` および `ifThenElseStatement` 非終端記号は，9-10 行目および 12-13 行目で定義されており，else 節があるかないかで区別されている．

図 6 で具象構文木中の if 文に該当するノードから Uni-Model 表現を生成する Xtend コードの断片を示す [*3]．1-6 行目で `ifThenStatement` 非終端記号に，8-14 行目で `ifThenElseStatement` 非終端記号に対応するノードに訪れた際の処理を示す．

JUNICOEN では if 文を条件式，条件式が真の時の処理，偽の時の処理という 3 要素でモデリングしており，具象構文木上では偽の時の処理がある場合とない場合でノードの種類が異なっているのに対して，JUNICOEN では 1 種類に統合されている．したがって，具象構文木の 2 種類のノードに対して，`visitIfThenStatement` および `visitIfThenElseStatement` メソッドを定義しているが，どちらのメソッドも if 文に該当する Uni-Model 表現を生成している．2 および 9 行目は該当ノードが持つ子ノードを Visitor パターンで再帰的に訪問して Uni-Model 表現を生成している．3 および 10 行目は条件式に，4 および 11 行目は条件式が真の時の処理に，12 行目は条件式が偽の時の処理に該当する Uni-Model 表現を取得している．5 および 13 行目で取得した Uni-Model 表現を用いて，if 文に該当する `UniIf` クラスのオブジェクトを生成している．

図 7 で `UniIf` オブジェクトから Java コードを生成する Java コード断片を示す．if 文は常に条件式と条件式が真の時の処理を有しているので，2-6 行目で該当する Java コードの文字列を生成する．7 行目で偽の時の処理も有しているかどうか確認した上で，8-9 行目で該当する Java コードの文字列を生成する．

以上のように，具象構文木の各ノードに対応する `visit` メソッドを用意して，対応する Uni-Model 表現を生成するプログラムを作成すれば，Mapper を開発できる．また，各 Uni-Model 表現に対して，対応する言語の文字列を生

[*2] `https://github.com/antlr/grammars-v4/blob/master/java8/Java8.g4`

[*3] JUNICOEN は Java 言語と Xtend 言語の両方を用いて開発をしており，Java Mapper は Xtend 言語で記述されている．なお，Xtend コードをコンパイルすると Java コードが生成されるので，結果的には JUNICOEN のバイナリファイルは Java コードから生成される．

```
statement
  : statementWithoutTrailingSubstatement
  | labeledStatement
  | ifThenStatement
  | ifThenElseStatement
  | whileStatement
  | forStatement ;

ifThenStatement
  : 'if' '(' expression ')' statement ;

ifThenElseStatement
  : 'if' '(' expression ')' statementNoShortIf
                              'else' statement ;
```

図 5 if 文に関連のある Java 文法ファイルの断片

Fig. 5 The Java code fragment related to the if statement.

```
override visitIfThenStatement(Java8Parser.IfThenStatementContext ctx) {
        val maps = createMaps(ctx)
        val cond = maps.getOne("expression") as UniExpr
        val trueBlcok = maps.getOne("statement") as UniBlock
        return new UniIf(cond, trueBlcok, null)
}

override visitIfThenElseStatement(Java8Parser.IfThenElseStatementContext ctx) {
        val maps = createMaps(ctx)
        val cond = maps.getOne("expression") as UniExpr
        val trueBlcok = maps.getOne("statementNoShortIf") as UniBlock
        val falseBlcok = maps.getOne("statement") as UniBlock
        return new UniIf(cond, trueBlcok, falseBlcok)
}
```

図 6 if 文に関連する具象構文木を走査して Uni-Model 表現を生成する Xtend コードの断片

Fig. 6 The Xtend code fragment for generating the representation of Uni-Model related to the if statement.

```
public void traverseIf(UniIf node) {
        printIndent();
        print("if (");
        parseExpr(node.cond);
        println(") {");
        genBlockInner(node.trueBlock);
        if (node.falseBlock != null) {
                printlnIndent("} else {");
                genBlockInner(node.falseBlock);
        }
        printlnIndent("}");
}
```

図 7 UniIf オブジェクトから Java コードを生成する Java コード断片

Fig. 7 The Java code fragment for generating Java code from a UniIf object.

成するプログラムを作成すれば，Generator を開発できる．Mapper と Generator の両方を開発すれば，対応言語と Uni-Model との相互変換が実現する．

4.3 JUNICOEN ーブロック相互変換システム

本節では BlockModel と Uni-Modle 表現の相互変換を実現するシステム（Block 変換システム）について説明する．**図 8** に Block 変換システムの設計を示す．BlockModel は，BlockEditor で入出力可能な Block コード形式（DOM）である．この入出力部分の大半は，オリジナルの OpenBlocks の機能を利用している．BlockMapper と BlockGenerator は我々が手作業で実装している．以下，if 文を例に Block Mapper と Block Generator の実装について説明する．

図 9 に if を表す BlockModel の例を示す．BlockModel と Uni-Model 変換をするに当たり，BlockModel に若干の修正を加えている．例えば，変数宣言を表す BlockModel に型情報が含まれていなかったものに，型情報を持つように再設計している．これは Uni-Model の変数宣言が型情報を持つため，Uni-Model への変換に不足する情報が出てしまったためである．このため，フレームワークの提供する BlockModel の再設計をし，モデル間の情報の差をなくした BlockModel を Block 変換システムに利用している．

図 11 に BlockModel の if 文に該当するノードから Uni-Modle 表現を生成する Java コードの断片を示す．**図 13** で表現された UniIfModel を変換する際には，条件式，条件式が真の時の処理の処理を再帰的に解析して Uni-Model 表現を生成している．2 行目では BlockModel の if 文の条件式，真の時の処理のブロックノード，偽の時の処理のブロックのノードを取得し，3 行目でそれらを再帰的に解析して Uni-Model 表現を取得している．3 行目では取得した条件式，真の時の処理，偽の時の処理の Uni-Model を用いて Uni-If モデルを生成している．

図 10 に UniIf モデルから Uni-Modle 表現を生成する Java コードの断片を示す．BlockMapper は，1 行目で if に該当する Block のモデルを生成し，4 行目で Uni-If モデルの条件式を再帰的に解析し，条件式の BlockModel を，5-6 行目で真の時の処理と偽の時の処理を解析し，BlockModel を生成しており，8 行目で解析した結果生成された if を表す全ての BlockModel を生成して返している．生成された DOM を BlockEditor で読み込み，BlockEditor 上で Block 型言語で Uni-Model で表現されたプログラムが表示される．

Java 変換システムとは若干手法が異なるものの，BlockModel を再帰的に走査し，対応する Uni-Model を生成するプログラムを作成することで Mapper を作成することができた．各 Uni-Model 表現に対して，対応する BlockMode を作成するプログラムを作成して Generator を作成することができた．従って，今後新たに Uni-Model との相互変換に対応した言語は，Block 型言語との相互変換をすることが可能である．

4.4 インタプリタ部の実装とタートルグラフィックスライブラリの対応

本節では JUNICOEN が有するインタプリタと，タートルグラフィックスライブラリの呼び出しについて説明する．

JUNICOEN は，Uni-Model 表現を実行するインタプリタを有している．図 14 で `UniIf` オブジェクトを実行する Java コード断片を示す．JUNICOEN は if 文を式の一つと

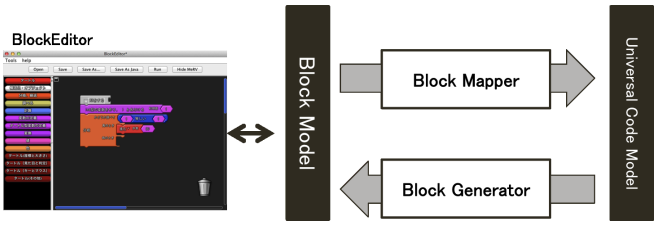

図 8 Block Mapper と Generator の設計
Fig. 8 The design of the Block Mapper and Generator.

```
<Block genus-name="ifelse" id="137" kind="command">
    <Label>分岐</Label>
    <Location>
        <X>65</X>
        <Y>127</Y>
    </Location>
    <BeforeBlockId>141</BeforeBlockId>
    <Sockets num-sockets="3">
        <BlockConnector con-block-id="138" connector-kind="socket"
            connector-type="boolean" init-type="boolean" label="かどうか調べて"
            position-type="single" />
        <BlockConnector con-block-id="144" connector-kind="socket"
            connector-type="cmd" init-type="cmd" label="真のとき" position-type="single" />
        <BlockConnector connector-kind="socket"
            connector-type="cmd" init-type="cmd" label="偽のとき" position-type="single" />
    </Sockets>
</Block>
```

図 9 説明用サンプルの BlockModel 表現
Fig. 9 The sample model represented by BlockModel.

```
1  public UniIf parseIfBlock(Node node, HashMap<String, Node> map){
2    Node argsNode = getChildNode(node, "Sockets");
3    List<List<UniExpr>> args = parseSocket(argsNode, map);
4    return new UniIf(args.get(0).get(0), (UniBlock) args.get(1), (UniBlock) args.get(2));
5  }
```

図 10 BlockModel の if ノードから Uni-Modle 表現を生成する Java コードの断片
Fig. 10 The code fragment of BlockMapper related to if statement.

```
1  public List<Element> parseIf(UniIf ifexpr, Document document, Node parent){
2    Element ifElement = createBlockElement(document, "ifelse", ID_COUNTER++, "command");
3
4    List<Element> sockets = parseExpr(ifexpr.cond, document, ifElement);
5    List<Element> trueBlock = parseBody(document, ifElement, ifexpr.trueBlock);
6    List<Element> falseBlock = parseBody(document, ifElement, ifexpr.falseBlock);
7
8    return createBlockIfModel(ifElement, document, parent, sockets, trueBlock, falseBlock);
9  }
```

図 11 UniIf オブジェクトから BlockModel を生成する Java コード断片
Fig. 11 The code fragment of BlockGenerator related to if statement.

```
1  if(x > 3){
2          fd(100);
3  }
```

図 12 説明用サンプルの Java 表現
Fig. 12 The sample model represented by Java.

して扱っており，1 行目で `expt` 変数の型が `UniIf` かどうかを確認している．`UniIf` 型である場合は，条件式の評価を行い（3 行目），真の場合は真の時に処理するブロックを（4 行目），偽の場合は偽の時に処理するブロック（6 行目）を実行する．同様に，Uni-Model 表現の各要素に対して，実行時の振る舞いを記述して，インタプリタを開発した．

タートルグラフィックスライブラリは `Turtle` クラスを提供しており，そのオブジェクトを生成することで，タートルを表示するウィンドウを表示して，タートルを操作できるようになる．JUNICOEN のインタプリタがタートルグラフィックスライブラリを利用するプログラムを実行する際に，実行時のグローバル環境にタートルを操作するための関数を定義することで，インタプリタ上の Uni-Model 表現からタートルを操作できるようにした．

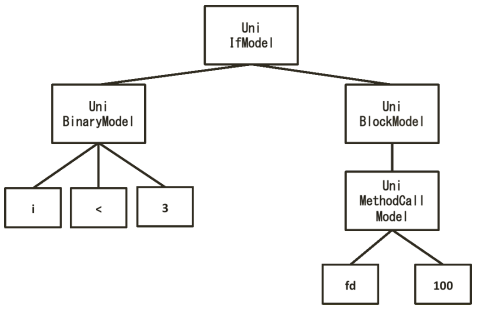

図 13 説明用サンプルの Uni-Model 表現
Fig. 13 The sample model represented by Uni-Model.

```
1  if (expr instanceof UniIf) {
2          UniIf ui = (UniIf) expr;
3          if (toBool(execExpr(ui.cond, scope))) {
4                  return execBlock(ui.trueBlock, scope);
5          } else {
6                  return execBlock(ui.falseBlock, scope);
7          }
8  }
```

図 14 UniIf オブジェクトを実行するインタプリタの Java コード断片
Fig. 14 The Java code fragment for executing a UniIf object in the interpreter.

5. おわりに

本稿では，Computational Thinking(CT) の育成を目指したプログラミング教育を促進するための，教育現場で様々な言語を選択できる，特定の言語に依存しないプログラミング教育環境を提案した．プログラミング教育における教育者が言語選択の歴史を概観し，問題を整理した上で，現代の技術を利用し，UNICOEN を利用した抽象 AST を用いた言語変換によるプログラミング教育環境構築の可能性を議論した．提案環境を利用して実際に既存のプログラミング入門現場で利用している環境を再構築した結果，Java とブロック言語の変換に関して，新システムで動作することを確認した．

今後は，第 3 の言語への翻訳（Javascript を予定している）とそのコスト評価の試みをとおして，抽象 AST である Universal Code Model の洗練，Web ブラウザへの対応を行う．現在，SSS2015 において，複数言語の相互変換の動作デモを実施することを目指して開発を行っている．

謝辞 本研究は JSPS 科研費 25730203，26280129 の助成を受けたものです．

参考文献

[1] School, C. A.: Computing At School, http://www.computingatschool.org.uk/ (2015.05.16 に参照).

[2] Wing, J.: Computational Thinking, *Communications of the ACM*, Vol. 49, No. 3, pp. 33–35 (2006).

[3] 内閣官房：世界最先端 IT 国家創造宣言，http://www.kantei.go.jp/jp/singi/it2/kettei/pdf/20130614/siryou1.pdf. (2013).

[4] Papert, S.: *Mindstorms: children, computers, and powerful ideas*, Basic Books, Inc., New York, NY, USA (1980).

[5] Kay, A.: Squeak Etoys authoring & media, *Viewpoints Research Institute Research Note*, (online), available from ⟨http://www.squeakland.org/resources/articles/article.jsp?id=1008⟩ (2005).

[6] Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B. and Resnick, M.: Scratch: a sneak preview, *Proceedings Second International Conference on Creating Connecting and Collaborating through Computing 2004*, pp. 104–109 (2004).

[7] 大岩元：識字教育としてのプログラミング，情報処理学会論文誌 教育とコンピュータ，Vol. 1, No. 2, pp. 1–6 (2015).

[8] 兼宗 進，中谷多哉子，御手洗理英，福井眞吾，久野靖：初中等教育におけるオブジェクト指向プログラミングの実践と評価，情報処理学会論文誌：プログラミング，Vol. 44, No. SIG 13(PRO 18), pp. 58–71 (2003).

[9] 西田知博， 原田章，中村亮太，宮本友介，松浦敏雄:：初学者用プログラミング学習環境 PEN の実装と評価，情報処理学会論文誌，Vol. 48, No. 8, pp. 2736–2747 (2007).

[10] 松澤芳昭， 保井元， 杉浦学，酒井三四郎：ビジュアル-Java 相互変換によるシームレスな言語移行を指向したプログラミング学習環境の提案と評価，情報処理学会論文誌，Vol. 55, No. 1, pp. 57–71 (2014).

[11] 坂本一憲， 大橋昭，太田大地，鷲崎弘宜，深澤良彰：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，情報処理学会論文誌，Vol. 54, No. 2, pp. 945–960 (2013).

[12] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M. and Paterson, J.: A Survey of Literature on the Teaching of Introductory Programming, *SIGCSE Bull.*, Vol. 39, No. 4, pp. 204–223 (2007).

[13] Hadjerrouit, S.: Java As First Programming Language: A Critical Evaluation, *SIGCSE Bull.*, Vol. 30, No. 2, pp. 43–47 (1998).

[14] Lewis, C.: How programming environment shapes perception, learning and goals: logo vs. scratch, *Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE '10)*, pp. 346–350 (2010).

[15] Lewis, C.: What do Students Learn About Programming From Game, Music Video, And Storytelling Projects?, *Proceedings of the 43rd ACM technical symposium on Computer science education (SIGCSE '12)*, pp. 643–648 (2012).

[16] Bieniusa, A., Crestani, M., Degen, M., Heideger, M., G.P, Klaeren, H., Knauel, E., Sperber, M., Thiemann, P. and Wehr, S.: HtDP and DMdA in the Battlefield ? A Case Study in First-year Programming Instruction, *Proceedings of the 2008 Workshop Functional and Declarative Programming in Education* (2008).

[17] Dann, W., Cosgrove, D., Slater, D., Culyba, D. and Cooper, S.: Mediated Transfer: Alice 3 to Java, *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pp. 141–146 (2012).

[18] Roque, R. V.: OpenBlocks: An Extendable Framework for Graphical Block Programming Systems, *Master thesis at MIT* (2007).

[19] Google Inc: Blockly:A visual programming editor, http://code.google.com/p/blockly/ (2013.03.17 に参照).

[20] Kolling, M. and Quig, B.: The BlueJ system and its pedagogy, *Computer Science Education*, Vol. 13, No. 4, pp. 249–268 (2003).

[21] Warth , A. and Piumarta, I.: OMeta: an Object-Oriented Language for Pattern Matching', *DLS 2007*, pp. 11–19 (2007).

[22] Warth, A., Yamamiya, T., Ohshima, Y. and Scott, W.: Toward a more scalable end-user scripting language, *Proceedings Second International Conference on Creating Connecting and Collaborating through Computing 2008*, pp. 172–178 (2008).

[23] Ohshima, Y., Lunzer, A., Freudenberg, B. and Kaehler, T.: KScript and KSWorld: A Time-Aware and Mostly Declarative Language and Interactive GUI Framework, *ACM Proceedings of the "Onward! 2013" Conference* (2013).

[24] Sakamoto, K.: A Framework for Analyzing and Transforming Source Code Supporting Multiple Programming Languages, *Proceedings of the 12th Annual International Conference Companion on Aspect-oriented Software Development*, AOSD '13 Companion, New York, NY, USA, ACM, pp. 35–36 (2013).

[25] 藤井春香，塩見彰睦：FA コントローラプログラミングの複数 DSL 間の相互変換に関する研究，第 76 回全国大会講演論文集 2014(1)，pp. 53–55 (2014).

[26] Allen, E., Cartwright, R. and Stoler, B.: DrJava: A Lightweight Pedagogic Environment for Java, *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '02, New York, NY, USA, ACM, pp. 137–141 (2002).

[27] FINDLER, R. B., CLEMENTS, J., FLANAGAN, C., FLATT, M., KRISHNAMURTHI, S., STECKLER, P. and FELLEISEN, M.: DrScheme: a programming environment for Scheme, *Journal of Functional Programming*, Vol. 12, pp. 159–182 (2002).

[28] Moreno, A., Myller, N., Sutinen, E. and Ben-Ari, M.: Visualizing Programs with Jeliot 3, *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, New York, NY, USA, ACM, pp. 373–376 (2004).

[29] Hendrix, T. D., Cross, II, J. H. and Barowski, L. A.: An Extensible Framework for Providing Dynamic Data Structure Visualizations in a Lightweight IDE, *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '04, New York, NY, USA, ACM, pp. 387–391 (2004).

[30] 松澤芳昭， 岡田健，酒井三四郎：Programming Process Visualizer:プログラミングプロセス学習を可能にするプロセス観察ツールの提案，情報教育シンポジウム (SSS2012)，pp. 257–264 (2012).

[31] 平尾元紀，松澤芳昭，酒井三四郎：Compile Error Collection Viewer:修正履歴分析によるコンパイルエラー学習支援システム，情報教育シンポジウム (SSS2014)，pp. pp.151–158 (2014).

[32] Parr, T. and Fisher, K.: LL(*): The Foundation of the ANTLR Parser Generator, *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, New York, NY, USA, ACM, pp. 425–436 (2011).