2009－SE－163 (7)
2009／3／18

三言語独立のオブジェクト指向記述言語OOJの実装と検証

大木幹生[†2] 片野克紀[†2] 三塚恵嗣[†1]
沼崎隼一[†1] 涌井智寛[†2] 加藤木和夫[†3]
池田陽祐[†2] 畠山正行[†1]

われわれの研究グループでは，以前よりプログラムの生成・取得を容易にするための一貫した記述言語とその環境であるOOJの研究を行ってきた．しかし，研究開始当初はいくつかの問題の障害からOOJ構築の第一段階として，分析，設計，実装の三段階に分けて開発 (三言語独立方式) を進めてきた．その結果，分析段階からプログラムの生成までの一貫した記述例を得る事が出来たため，三言語独立した方式においてOOJのある程度の成果を得たと言える．しかしながら，それと同時に記述内容の一貫相似性に関して問題が浮上してきたため，この問題に対処するべく研究の段階を進め，一貫相似性問題に対処した三言語独立方式のOOJの提案を行うと共にその開発を行った．その結果，三言語独立の方式に依っても一貫相似性のある記述やプログラムを得る事を立証した．

An Implementation and a Validation of the Three-Independent, Object-oriented Description Language OOJ

Mikio Ohki ,[†2] Katunori Katano ,[†2] Keishi Mituka ,[†1]
Yoshikazu Numazaki ,[†1] Tomohiro Wakui ,[†2] Kazuo Katogi,[†3]
Yousuke Ikeda [†2] and Masayuki Hatakeyama[†1]

We have been developing the Object-oriented (OO) description language OOJ. The OOJ is a language to realize the Object-oriented, Integrally Consistent Similarity in the various simulations with high similarity. The OOJ is, however, difficult to be developed in the singleton language formalism. Then, we have developed the series of three kinds description languages, say the OONJ, ODDJ, OPDJ in stead of the OOJ. In the present paper, then, we have implemented these three languages, and validated the similarity among these description languages by making use of the same description examples throughout the OONJ, the ODDJ, the OPDJ, and finally the Java program. Throughout the results, we have confirmed the integrally consistent similarity. In the future work, we will try to develop the singleton description language OOJ to verify the above described similarity.

1. はじめに

我々の研究グループでは，DU [*1]をターゲットユーザにOOJ(Object Oriented Japanese)[1] の開発を行って来た．しかしながら，現実世界における対象世界（分析段階），計算機で動かすことを前提にし特定プログラミング言語 (以降，**PL** と略) に依存しない対象世界の言葉で表現された計算機世界（設計段階），駆動命令や実行手順を示すPL等の言語表現の世界（実装段階），駆動させる動的な計算機世界（駆動段階），以上四つの段階の記述をを開発当初の段階において一つの言語仕様に収めることは困難であった．そこで，OOJを分析，設計，実装という三段階に分けて記述言語として開発する（三言語方式）ことでこの困難に対処したが，開発が進むにつれてこれら三つの段階間における一貫相似性の問題が浮上した[*2]．そこ

†1 茨城大学工学部情報工学科
Department of Computer and Information Sciences, Ibaraki University
†2 茨城大学大学院理工学研究科情報工学専攻
Graduate School of Science and Engineering, Ibaraki University
†3 茨城県立産業技術短期大学
Ibaraki Prefectural Industrial Technology Junior College

⋆1 DUとはDomainUser（特定分野の専門家）の意．ここでは非情報系の科学・技術分野専門家を想定している．

⋆2 本論文での**一貫相似性**とは，分析・設計・実装の三言語において同等内容の別表現となるプログラムが記述される性質を持つ事を指す．

で，三段階での開発においてある程度の実績を作った今，開発の段階を進め，相似性問題に対処した開発へと移行するにいたった．

2. 従来の三言語独立方式の開発

OOJ の開発初期段階において実装上の困難から，三段階に分けて開発している事は先に述べた通りである．**図 1** にこの OOJ における実現目標であるところの，オブジェクト指向一貫相似性モデリング/記述/変換の一連の流れを示してある．まず，DU は自身の再現したい対象世界を決定する (図 1 の左上部)．次に，対象世界をオブジェクト指向モデリングを適用して分析を行い，分析記述言語 OONJ[2] を用いて分析段階記述を作成する (図 1 の上部中央，対象世界の分析段階)．この時，DU が OOJ の初心者等で分析記述を行うことが困難である場合には，まず simpleOO 分析記述を行った後に OONJ 記述に昇華する (「対象世界の分析段階」内左下部)．続いて，分析段階の記述を基に設計段階にて計算設計記述言語 ODDJ[3] を用いて計算機世界での設計を行う (図 1 の上部右側)．続いて，設計段階の ODDJ 記述を基に実装段階にて実装記述言語 OPDJ[4] を用いて PL 記述世界への実装を行う (図 1 の下部右側)．最後に，実装段階の OPDJ 記述に対してトランスレータを用いて記述変換を行う事でプログラムのソースコードが生成される．現在のところ Java に変換する事を目標としている．

2.1 三言語独立方式の開発方針

開発においては，各段階に担当者が別々に存在し，それぞれがほとんど独立した形で開発が進められた．また，トランスレータを用いて各段階の記述を変換する事で三段階間をつないだ．分析，設計段階では記述内容の確認のしやすさなどから OOSF[5] 形式[*1]を用い，記述難易度を下げるために専用の支援エディタを開発した．実装段階では学習の容易性から OOSF 形式を用いずに独自の形式を採用し，通常のテキストエディタなどでも編集が行えるような保存形式を採った．この独立方式の開発はある程度進み，実際にプログラム生成まで至った記述例もいくつか作成された．

2.2 分析段階の OONJ の特徴概要

Object Oriented Natural Japanese(オブジェクト指向分析記述言語) の略．OOJ において，対象世界の分析を行う記述言語．記述には自然日本語を用い，OOSF の範囲内に限られるものの比較的自由に記述できる設計になっている．ただし，対象となる世界の分析のみを行うので計算機に特有の情報は記述してはならない．

2.3 設計段階の ODDJ の特徴概要

Object oriented Design Discription Japanese(オブジェクト指向設計記述言語) の略．OONJ による分析記述は，コンピュータ上でシミュレーションするために必要な情報が不足しているため，分析記述を基に計算機世界へ移行する為に必要な情報を付加し，分析記述を計算機世界の記述に変換，修正するための言語である．

2.4 実装段階の OPDJ の特徴概要

Object oriented Program Description Japanese(オブジェクト指向実装記述言語) の略．ODDJ による設計記述に対して，特定 PL の記述 (つまりプログラム) として完全かどうかのチェック，および，入出力関係の情報を付加する．また，OPDJ の記述を OPDJ トランスレータによって変換する事で，実際の実行可能にプログラムが生成される．

3. 従来の三言語独立方式における問題点

本来であれば，図 1 に示されている様に対象世界の結果と計算駆動の段階での結果の間に相似性評価が行われる必要がある (図 1 の左側,「一貫相似性評価」)．しかしながら，対象世界と計算駆動の段階間での相似性評価は以下の理由から困難であった．

- 実行結果の予測がつかない場合においては，相似性評価を行うにはプログラムのソースコードを理解する必要がある．
- 対象世界の全体構造は DU の理解としてのみ存在する．

したがって，対象世界から分析段階，分析段階から設計段階，設計段階から実装段階，実装段階から計算駆動段階と各段階毎に相似性を評価し，一定水準以上の相似性を確保する事で全体としての相似性 (一貫相似性) を実現可能であると考えた．しかしながら，前章の三言語独立方式の開発においては，

(1) 各段階の担当者が独立して開発を行っていた．
(2) 各担当者が自身の段階の開発に手一杯な状態にあったため，一貫相似性を保持する事に対しての

---

*1 OOSF とは Object Oriented Structured Frame(オブジェクト指向構造化フレーム) の意．OOSF では記述の構造の他に表示形式なども含めてその構造を規定している．

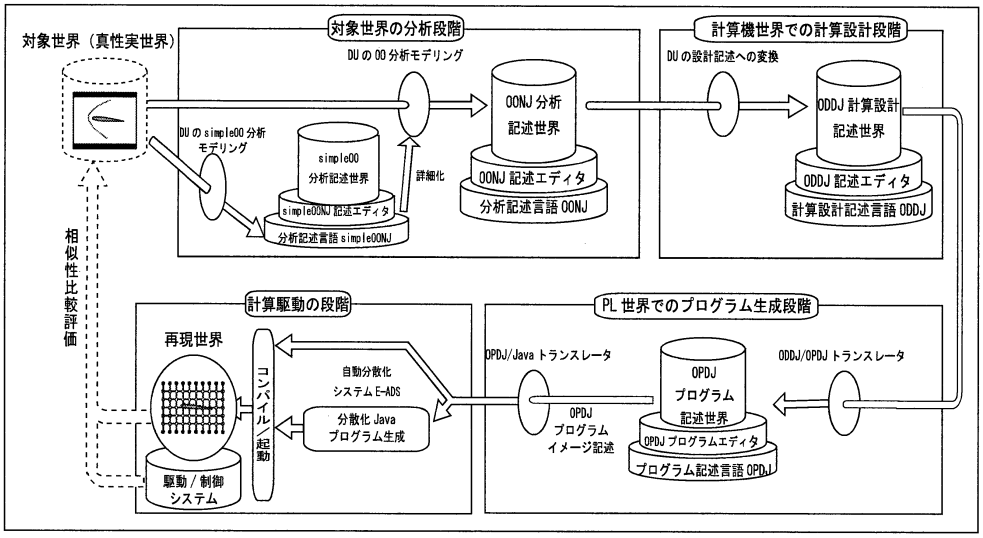

図 1 オブジェクト指向一貫相似性モデリング過程/記述/変換

意識が希薄だった．

という問題があった．それゆえに対象世界での現象の振る舞いに始まり，計算機世界でのプログラムに至るプロセス (複数段階を指す用語) を忠実に辿ると共にシミュレーション結果と比較しての一貫相似性の検証が困難な状況になっていた．一貫相似性の検証が困難であった主な原因をまとめると以下のような問題点があった．

(1) 分析，設計段階と実装段階の間で言語設計が異なる．

(2) 実装段階において表現形式が特定言語に依った形式になっている．

(3) 分析，設計段階と実装段階の間で記述の保存形式が異なる

(4) 各段階間のトランスレータによる変換規則が明示的でない．

(5) サブスロット*1における各段階間の日本語単文の変換が DU による手作業である．

これらについて詳細を説明する．

(1) は分析，設計段階では OOSF 構造を採っているが，実装段階では独自構造を採っていたために設計段階 ODDJ 記述と実装段階 OPDJ 記述の比較が難しくその内容の相似性の確認が困難であるという問題があった．

(2) は表現形式が特定言語に依っているため，その言語を知っているユーザ以外には記述内容の確認が困難であるという問題があった．

(3) は分析，設計段階では保存形式に XML を採用していたため，XML と DTD*2を用いた妥当性検証により記述規則と記述内容の妥当性を検証することができたが，実装段階では保存形式に独自のテキスト形式を採用しているために妥当性検証を目視点検にて行う必要があった．

(4) は先にも述べた通り，各段階間はトランスレータにより記述の変換を行っているが，この変換規則が明示的になっていないため客観的な妥当性検証が行うことができないという問題があった．

(5) はサブスロットにおける日本語単文変換を手作業で行うので，DU の負荷が大きいという問題があった．

本論文では (1)〜(4) の問題点についての対処を行い，一貫相似性を重視した三言語方式の OOJ の提案を行う．(5) については別論文[6] にて扱うため，本論文では問題の列挙に留め取り扱わない．

4. 相似性を重視した三言語独立 OOJ の提案

従来の三言語方式では一貫相似性に対して問題があった事は先に述べた通りである．そこで，一貫相似性の検証，すなわち，対象世界の分析から生成された PL の実行結果までの相似性を三言語方式において一

⋆1 OOSF の集約階層構造において末端の階層に当たる離散要素

⋆2 三言語独立方式，OOJ においては，計算機世界での記述規則として DTD 用いている

貫して検証する仕組みを提案する．それはつまり従来の三言語方式で相似性が問題となった最大の原因が，それぞれの段階が完全独立して開発されてきたことにある．各段階における記述支援するエディタなどが独立して開発された点は，使い勝手の点やDUの操作負担が大きいと言う点を除けば問題はない．むしろ大きな問題点は，各段階の言語仕様までもが独立して開発されたことに問題がある．各段階の言語仕様が独立して設計された事，トランスレータを用いて各段階間の記述を変換する際の変換規則が示されていない事から，記述変換の妥当性を検証する事が困難な状態になっていたという点である．

そこで，相似性を実現するために以下を計画した．

(1) 各段階の記述をトランスレータによる記述変換の妥当性が検証できるよう言語仕様を見直す．

(2) 通常のPLでは文法に当たる構造記述規則を，対象世界と計算機世界そしてOOPL世界の三世界について可能な限り「同等内容の別表現」になるように調整する．例えばmpをメソッド呼び出しへの変換等がある．

(3) OPDJのエディタの表現形式がFortranに似通った独自の形式になっており，OOSF形式を採用していなかった．これでは直接目で見た際の相似性の検証が困難である．そこで，この独自形式もDUの利便性[*1]を考えて残すと共に，OOSF形式の構造化記述規則とエディタの開発を行う．

(4) 三つの記述言語における記述結果の直接の比較検討と共に，記述例をいくつかについてJavaプログラムに変換・生成して直接の比較を行う．

5. 相似性を重視した三言語独立OOJの設計

4章における(1)〜(4)の相似性を実現するための条件についての説明とそれに伴うOOJの設計を示す．

(1)については，各段階のトランスレータの変換規則を作成すれば，規則段階で客観的な妥当性の判断が可能になるため，トランスレータにて変換を行った記述内容の変換は，規則段階の妥当性を超えない範囲に収める事が出来る．

(2)については，設計，実装段階で構造記述規則の再設計を行った．分析段階は，

- 一貫した流れの中で先頭の段階である
- 現時点ではDUの実現したい事を三言語の中でもっともカバーできている

ため，再設計を行わなかった．したがって，いかに分析段階の内容を計算機世界に持ち込めるように再設計するかが設計，実装段階での課題となった．

また，計算機世界での構造記述規則であるDTDの設計は，従来は三言語とも構造記述規則との差が大きかったため，構造記述規則上は妥当であっても，エディタを用いて記述する際にはその内容が妥当でなくなることがあった．そこでDTDに関して構造記述規則とその内容がほぼ同じになる様に再設計を行った．

(3)については，まず，OPDJの記述規則をOOSF化する必要があった．また，エディタが初めからの作り直しになるのでDUの使い勝手も考慮し，すでに稼動状態にあった分析段階のエディタを参考に設計，実装を行った．

(4)については，OOJを用いてJavaプログラムまでの一貫した記述例を作成し，三言語間の変換結果やJavaプログラムへの変換結果，実行結果について一貫相似性や妥当性を検証した．

また，本設計においては以下の問題点があることがわかっている．

1 **mpが計算機世界では複数の意味をもつ可能性がある**

分析段階では通常mp[*2]は，mpの相手オブジェクトのメソッド呼び出しに用いられているが，OOPLにおいてはリターン文等もmpと見なすことが出来る．しかしながら，現状ではメソッド呼び出し以外として扱う仕組みが設けられていないため，その設計段階以降でmpの種類の区別が出来ない．

2 **SDR内容で実現されていない機能がある**

実装段階の構造記述規則において，現在出力先として想定しているJavaには存在しない機能が存在するなど特定言語に依存する部分に付いては実装を見送った．

3 **設計段階で採用するデータ型の範囲が厳密に定まっていない**

設計段階で採用するデータ型に付いて多言語の型について調べたが，各言語に特有のデータ型を何処まで導入するかについての議論が完了していない．現在は，多くの言語に対して採用されているデータ型を設計段階でのデータ型として採用している．

4 **設計段階で採用するスコープが厳密に定まってい**

---

*1 DUにはFortranを常用PLとする方が多い．

*2 message passing(相互作用伝達)の意．OOJではオブジェクト間の情報伝達に用いている

ない
現時点では最終的にJavaに変換することが分かっていたため，システムとしてのOOJの完成を重視するためにスコープの設定を見送った．今後開発が進めばJavaのみならず多言語への変換が求められるため，スコープの設定を行う必要がある．

6. 三言語独立OOJの実装

分析，設計，実装の全てにおいて，実装は以下の環境にて行われた．

- OS:Windows2000
- 開発言語:Java SE6
- 開発環境:Eclipse3.4

従来どおり今回の再設計においても各段階ごとに担当者がそれぞれ付き実装は独自に行われたが，従来と異なる点は設計作業において協調歩調がとられた点である．これにより，設計時点でそれぞれの段階が果たすべき役割の明確化，実現すべき機能が明確化され相似性確認の向上に貢献したと言える．

また，記述環境である支援エディタの実装においてもすでにある程度の成熟度に達していた分析段階のエディタをベースとする事で，GUIおよび操作性の統一化を図ることが出来た．これにより，従来は各エディタ毎に操作方法に統一性が無かった(特に設計が大きく異なっていた実装段階の)エディタの操作性に関して大きく向上したと言える．

7. 三言語独立OOJの実装結果とその成果

図2〜図4に本設計における各段階の構造記述規則の一部を掲載する．図2は分析段階の記述規則の一部である．図3は設計段階の記述規則の一部である．図4は実装段階の記述規則の一部である．それぞれの図は全て，共通の部分を示している．

図2〜図4において例えば，分析段階において図2の(16)番の規則は「スロット記述」は「スロット総称文」または「注釈」と，「サブスロット」から成り立つ事を示している．『lf』はLine Feedの略で行変えを意味している．

次に，同じ部分で設計段階(図3の(6)番の規則)ではアクセス修飾子となる「制約」，仮引数を定義するための「仮引数定義」，スロット内部変数を定義するための「スロット内変数記述」が計算機世界に必要な新たな情報として分析段階の情報に付け加えられていることが分かる．『rt』はRight Textの略で右寄せを意味している．また，設計段階において分析段階に存在した「注釈」が消えているのは，設計段階では「注釈」は処理されないためである．

実装段階ではまったく同じ構造(図4の(6)番の規則)になっているが，これは，実装段階は設計段階での記述内容の確認と，ユーザーインターフェイスとしての入出力（例えば，画面に出力等）の情報の付与が主な役割であるため，設計段階からの新たな情報の付加は非常に少ないため，同一構造で扱えるためである．この様に各段階において，極力前段階の情報に対して新たな情報を付加するという設計にする事で記述規則レベルでの相似性の向上を図ると共に，目視での相似性検証において負担を軽減するという効果を得た．

8. 記述例を用いた三言語OOJの評価と考察

図5〜図8にOOJを使用して作成した記述例の一部を掲載する．対象世界は「一次元衝撃波管内の流れ」[7]であり，それぞれ図5は分析段階の記述例の一部，図6は設計段階の記述例の一部，図7は実装段階の記述例の一部，図8は最終的に出力されたJavaプログラムの一部を示しており，それぞれは共通の部分を示している．

分析段階の例では，一行目で「駆動を開始する」というスロット名が示されており，以降順番に，反復試行内で「任意のステップ回数」という判定式の下，「衝撃派が伝播し流速が変化する」，「ステップ数の格子点速度を取得する」という振る舞いが行われることを示している．これらの処理は内部の他のスロットに対して処理を渡している事が相互関係により示されている．また，この「駆動開始する」という一連の処理は相互関係により「駆動シナリオ」よりmpを受け取っている事が分かる．

設計段階ではまず，「駆動開始する」に大して適切にデータ型とアクセス制約が設定された後，反復試行で使用する「反復回数」という変数が用意されデータ型に「整数」，アクセス制約に「私有」(privateの意)が設定されている．そして，反復回数に0が代入され初期化されている．反復条件式の「反復試行¡ステップ数」の「ステップ数」という変数は別のところで定義されている変数である．反復試行内では「衝撃波が伝播する」，「格子点速度を記録する」という処理が行われている．-6行目は-5行目における「格子点速度を記録する」という処理が引数として「反復回数」をと

15 <スロット>	:=:(≪短冊形のスロット実線外枠線≫《<スロット記述>
	| <引用スロット記述>≫) | <フレームヘッダスロット>
20《 》	:=:≪短冊形のスロット実線外枠内への収容を指示する記号.
16 <スロット記述>	:=:(<スロット総称文> | <注釈>)(< lf ><サブスロット>)*)
( ) <引用スロット記述>	:=:<引用スロット総称文> (< lf ><引用変更記述>)*
3 <フレームヘッダスロット>	:=:<フレーム番号>< sp ><ファセット記号>< sp >
	<フレーム名> [<行内注釈>] [<フレーム間構造記述子>]
	(注) 本スロットだけは短形の<フレーム>枠外上部にスロット枠無しで置く.

図 2 OONJ の構造記述規則 (一部抜粋)

スロット要素集合			
(5)	<スロット>	:=:	≪ 短冊形のスロット実線外枠線 ≫《<スロット記述>》
(6)	<スロット記述>	:=:	<スロット総称文>< rt ><制約> (< lf ><仮引数定義>)*
			(< lf ><スロット内変数記述>)*(< lf ><サブスロット>)*
(7)	<変数スロット>	:=:	≪ 短冊形のスロット実線外枠線 ≫《<変数スロット記述>》
(8)	<変数スロット記述>	:=:	<変数スロット総称文> (< lf ><スロット内変数記述>)*
(9)	<初期化スロット>	:=:	≪ 短冊形のスロット実線外枠線 ≫《<初期化スロット記述>》
(10)	<初期化スロット記述>	:=:	<初期化スロット総称文>< rt ><制約> (< lf ><仮引数定義>)*
			(< lf ><スロット内変数記述>)*(< lf ><サブスロット>)*
(11)	《 》	:=:	≪ 短冊形のスロット実線外枠内への収容を指示する記号 ≫
(12)	<フレームヘッダスロット>	:=:	<フレーム番号>< sp ><ファセット記号>< sp ><フレーム名>
			<フレーム間構造記述子> *

図 3 ODDJ の構造記述規則 (一部抜粋)

スロット要素集合			
(5)	<スロット>	:=:	≪ 短冊形のスロット実線外枠線 ≫《<スロット記述>》
(6)	<スロット記述>	:=:	<スロット総称文>< rt ><制約> (< lf ><仮引数定義>)*
			(< lf ><スロット内変数記述>)*(< lf ><サブスロット>)*
(7)	<変数スロット>	:=:	≪ 短冊形のスロット実線外枠線 ≫《<変数スロット記述>》
(8)	<変数スロット記述>	:=:	<変数スロット総称文> (< lf ><スロット内変数記述>)*
(9)	<初期化スロット>	:=:	≪ 短冊形のスロット実線外枠線 ≫《<初期化スロット記述>》
(10)	<初期化スロット記述>	:=:	<初期化スロット総称文>< rt ><制約> (< lf ><仮引数定義>)*
			(< lf ><スロット内変数記述>)*(< lf ><サブスロット>)*
(11)	《 》	:=:	≪ 短冊形のスロット実線外枠内への収容を指示する記号 ≫
(12)	<フレームヘッダスロット>	:=:	<フレーム番号>< sp ><ファセット記号>< sp ><フレーム名>
			<フレーム間構造記述子> *

図 4 OPDJ の構造記述規則 (一部抜粋)

るため実引数としてこの変数を指定していることを表している．そして最後に条件判定である「反復回数」を 1 増やしている．これらの追加された情報は計算機世界で必要な情報であり，分析段階で記述された処理（日本語単文）を書き換えてはいない．以上の事から，分析段階の記述と設計段階の記述の間に相似性があると言える．

実装段階では設計段階とほぼ同様の記述内容であるが，設計段階で-6 行目に指定されていた実引数が-5 行目の「格子点速度を記録する」という関数呼び出しの仮引数部 (関数名の後の () 部) に直接指定されている．実装段階ではより計算機世界での記述に近い記述に直されているだけで新たな情報の付加や記述の書き換えはされていない．実装段階の位置づけからもこれは妥当な記述である (実装段階では設計段階の記述内容に問題が無いかの確認が主な作業であるため確認の行いやすい，より計算機に近い記述になっている)．以上の事から，設計段階と実装段階には相似性があるといえる．

この結果出力された Java プログラムが図 8 である．プログラムのソースコード内では実装段階で対応する部分に対してコメントが付与されている．これらの

コメントは自動的に付与されてソースコードが生成される．また，半角英数の名前はエディタ内部で名前テーブルが作成され自動的に変換される．このJavaのソースコードではまず,「tempVar0」という名前で反復回数の変数を作成し，0で初期化を行っている．次に「tempVar0¡Cls4.instance.glb2」という反復条件式の下にwhileループを行っている．whileループ内では「衝撃波が伝播する」というメソッド呼び出し処理に対応する「this.mthd2();」,「格子点速度を記録する」というメソッド呼び出し処理に対応する「this.mthd3」というメソッドを引数「tempVar0」で呼び出している．そして最後に反復回数を1増やしている．したがって，このJavaプログラムは分析段階から実装段階までの記述内容と同じ処理を行っている事が分かる．以上の事から，実装段階と出力されたソースコードには相似性があると言える．

これらの結果より，分析，設計，実装を経てJavaプログラムまで生成されたが，分析から実装までの相似性の確認が容易である事，そして図5〜図8に挙げた記述例の相似性が確認された．以上から図5〜図8の記述例については分析段階のOONJ記述からJavaプログラムに至る一連の変換と記述が相似性を保持しつつ実現されたことが分かった．それはつまり，一貫相似性の実現例が図5〜図8であることを明示した事に他ならないことが分かる．

しかしながら，まだ指摘すべき点も残る．それは，分析段階で記述される内容は日本語単文が殆どで設計段階へ自動的に変換される情報は現在のところごく少数であるという事である．分析段階での記述はその殆どがコメントとして移行され，そのコメントを見ながら設計段階で再入力する必要がある．よって，今後この点に関しての改善をする必要がある．また，Javaのソースコードを理解して相似性を確認できるDUが殆ど居ないため，実装段階からのトランスレータによる変換の相似性の高さが問題となる．この点については日本語単文変換機構JTM[3]を導入する事で解決を図りつつある．

9. 結論と今後の展望

本研究における分析，設計，実装の各段階の再設計により，各段階間の記述内容の相似性確認は格段に容易になったと考えられ，また，エディタのGUIを統一する事でシステム全体の利便性が向上し，単一的な操作方法で記述例を作成することが可能となった．これにより，三言語独立方式のOOJとして図5〜図8の記述例で示した通り，本論文の目的である一貫相似性の向上を図ることが出来た．

今後は，三言語独立方式でのOOJとして各段階の要求仕様に対する完成度を高めつつ，今回の再設計，実装にて得られた成果を基に，本来の実現目標である現実世界における対象世界から駆動させる計算機世界までを一つの言語仕様に収めたOOJ（単言語方式）の設計，実装へと移る予定である．

参考文献

1) 畠山正行，オブジェクトベース一貫日本語記述言語OOJに基づくプログラムの自動生成，第58回情報処理学会研究報告，2006-MPS-58 pp.55-58，2006年3月16日
2) 畠山正行，オブジェクト指向自然日本語記述言語OONJの設計とその記述力の評価，第58回情報処理学会研究報告，2006-MPS-58 pp.55-58，2006年3月16日
3) 片野克紀，オブジェクト指向設計記述言語ODDJの設計と実装，平成20年度修士論文，茨城大学大学院理工学研究科 (2008).
4) 加藤木和夫，畠山正行，オブジェクト指向プログラム自動生成記述言語OPDJとその記述開発環境，第61回MPS研究会報告，2006-MPS-61，pp. 65-68, (sept. 15, 2006) .
5) 畠山正行，オブジェクト指向分析自然日本語構造化フレームOOSFの設計と表現技法，日本シミュレーション学会誌，Vol.22，No.4，pp.195-209，Dec.，(2004).
6) 池田陽祐「単言語方式のオブジェクト指向プロセス記述言語OOJの設計」，第163回情報処理学会ソフトウェア工学研究会，平成21年3月18日
7) 峯村吉泰，流体・熱流動の数値シミュレーション 第3章，森北出版株式会社，ISBN4-627-91751-1.

番号	FN	NJ	相互関係
1	fn3.3	駆動開始する	≪mp≪2:駆動シナリオ[1-4]
2	fn3.3	| (反復)任意のステップ数回	
3	fn3.2	| 衝撃波が伝播し流速が変化する	≫[3]
4	fn3.2	| ステップ数の格子点速度を取得する	≫[4]

図 5 分析段階の記述例 (一部抜粋)

番号	FN	NJ	データ型	アクセス制約	相互関係
1	dfn3.3	駆動開始する	void	共有	
-1	dfn 2.8	和名「反復回数」英字名「J」	整数型	私有	
-2	dfn 3.1	反復回数=0			
-3	dfn 3.1	| while (反復回数<ステップ数)			
-4	dfn 3.2.1	衝撃波が伝播する ＞ 一次元衝撃波管			
-5	dfn 3.2.1	格子点速度を記録する ＞ 一次元衝撃波管			
-6	dfn 2.2	和名「反復回数」英字名「J」	整数型		
-7	dfn 3.1	反復回数=反復回数+1			

図 6 設計段階の記述例 (一部抜粋)

番号	PFN	OPD_記述	アクセス制約	データ型	相互関係
1	pfn3.3	駆動開始する	public	void	
2	pfn2.8	反復回数=0	private	int	
3	pfn3.1	while(反復回数<ステップ数)			
4	pfn3.2	衝撃波が伝播する()			
5	pfn3.2	格子点速度を記録する(反復回数)			
6	pfn3.1	反復回数=反復回数+1			

図 7 実装段階の記述例 (一部抜粋)

```
// 処理スロット
public void mthd1() {   // 駆動開始する
    int tempVar0 ;  // 反復回数
    tempVar0=0;

    while(tempVar0<Cls4.instance.glb2) {

            this.mthd2();  //  一次元衝撃波管.衝撃波が伝播する
            this.mthd3(tempVar0);  //  一次元衝撃波管.格子点速度を記録する
            tempVar0=tempVar0+1;

    }
    ;

}   //  ***** メソッド終了
```

図 8 Java プログラムの記述例 (一部抜粋)