情報処理学会論文誌　数理モデル化と応用　Vol.6　No.3　23–42 (Dec. 2013)

離散・構造化モデル記述言語系OOJの構築と科学技術計算教育への適用

—分析からプログラムまでの一貫開発とV&V評価実現の検討—

畠山 正行[1,a)]　池田 陽祐[1]　三塚 恵嗣[2]　大木 幹生[3]　加藤木 和夫[4]　上田 賀一[5]

受付日 2013年1月30日，再受付日 2013年3月21日，
採録日 2013年4月14日

概要：本論文では分析・設計・実装・プログラムの4つの段階を順次追って一貫したプログラム開発ができる記述言語系OOJの開発を報告する．OOJの適用分野は科学技術計算分野であり，この分野に適した離散・構造化モデルを開発した．そして4つの段階間の記述が必ず「同等内容の別表現」となる特性，すなわち一貫相似性，を実現する設計とした．この特性の実現により分析記述は忠実にプログラムに反映され，開発過程の正しさや成果の妥当性が確保される．この特性は別視点から見ればV&Vの実現でもあり，プログラムの信頼性向上に貢献するゆえにプログラム開発にも有効かつ有用な構築指針になる．OOJがこの特性を実現していることは論証された．想定ユーザ評価は大学院生と学部3年生の記述レポートを分析して得た．院生全員と3年生の想定ユーザはOOJを十分に理解し容易に使いこなし，OOJの理解と記述が容易であることが結論できた．以上からOOJは一貫した開発過程の実現によるプログラム開発の容易化，信頼性の向上やV&Vの評価の実現，および理解と実記述の容易性という3つの特長のゆえに，想定ユーザには有用かつ簡潔に使える言語系であると結論できた．

キーワード：離散・構造化モデル，記述言語系，一貫相似性，ソフトウェア開発，ソフトウェアの信頼性

A Design of Discretized and Structured Model Descriptive Language System OOJ and Its Applications to Scientific and Technical Calculation Education

Masayuki Hatakeyama[1,a)]　Yousuke Ikeda[1]　Keishi Mitsuka[2]　Mikio Ohki[3]
Kazuo Katougi[4]　Yoshikazu Ueda[5]

Received: January 30, 2013, Revised: March 21, 2013,
Accepted: April 14, 2013

Abstract: In the present paper, we will report a descriptive language system OOJ that can be applied from the analysis stage up to the program stage throughout the design and the implementation stage. OOJ is designed based on the discreted and structured model and applied in the fields of science and engineering calculations. In OOJ, the corresponding descriptions among four stages are integrally similar. That is, the descriptions in these four stages have some different phrases but the equivalent contents. These characteristics realize the concept of the V&V, and contribute to the upgrade of the reliability of the program. The assumed user estimations have been performed by the report of the graduate school students and the third grade undergraduate students. As the results, all third grade students and the graduate school students have gotten the sufficient recognition for OOJ and well-qualified Java program. We got the conclusion with high feasibility that OOJ contributes to the upgrade of the program reliability, to the evaluation of the V&V concept, and finally to the easiness and usability for the assumed users.

Keywords: discrete and structured model, descriptive language system, integral consistency and similarity, software development, reliability of software

1. はじめに

自然現象の解明や工学的問題の解決のために，着目する問題領域の分析やモデリング [1] を行って目的に沿った忠実な再現シミュレーションを行う計算ニーズが近年飛躍的に増加している．また工学的なシステムや工業製品の性能・安全性を確保・保証するためのシミュレーション計算も必要性が非常に高まり，活発化が著しい．いずれの場合にも計算量が莫大で再現精度の高いシミュレーション，そして計算自体の信頼性も必須となっている [2], [3], [4]．これらは現象の詳細を数値的にとらえてその本質に関わる知見を主張・検証したり，工業製品の設計データとして用いたりする際の重要な根拠データとなる．

それにともなってその計算プログラムもますます高度化・複雑化しており，プログラム開発[*1]技術の大幅な向上も社会的な要請となっている．具体的には

(1) **第 1 に**シミュレーションの数値精度や計算（式）の妥当性の確保と，計算プログラムの信頼性の評価という問題の解決への要請である，
(2) **第 2 に**プログラムを開発するユーザの方からの要請で「分析からプログラムまでを一貫して作成できる仕組みの提供」であった．

それらの要請の一半に応えるべく，主として科学技術計算のプログラムの開発を目的とした記述言語系 OOJ[*2]の構築を行ってきた [7]．狙いは当初の分析記述がプログラムに忠実・正確に反映する仕組みの構築にあった [8], [9]．それはプログラムの信頼性 [10], [11] の確保といい換えられる．

その解決策は基本的に，最初の分析段階の記述がどのように変換されてプログラムにまで達するのか，その全段階の記述と変換の詳細をホワイトボックス化し，任意の詳細度で追跡（トレース）できる仕組みを提供すればよいと考えた．なぜならばこの仕組みがあれば，科学技術計算プログラムの作成や運用に多少の経験があるユーザを想定すれば，彼らは自身で詳細に記述された変換過程の各段階の記述の妥当性の確認とミスの訂正ができるからである．であれば同分野の第三者からの客観的な意見等も同時に可能になる．そしてこれらの構築結果を，現在よく使われて実績のある MATLAB [12] や SysML [13]，ソフトウェア工学分野の MDA [14] との比較評価を通じて OOJ の特質を明らかにする．

以上の狙いに沿って以降，2 章では離散・構造化モデルを，3 章では OOJ 全体の構成と設計方針を述べ，4 章では OOJ の詳細設計を提示する．5 章と 6 章では従来よりもより効果的な分野と利用法としてプログラムの信頼性の問題と，一貫したプログラム開発法を教育分野に適用した結果を検討する．7 章では関連研究との比較と考察を，8 章では結論と今後の課題を述べる．

2. 離散・構造化モデルと一貫相似性の狙い

2.1 OOJ の背景：対象世界と想定ユーザ [5], [6]

想定対象世界は主として科学技術分野の計算，つまり自然現象の忠実な再現シミュレーションや工学・産業製品の目的を絞った再現シミュレーション計算である．それらは計算式・論理式の計算処理に帰着することが多い．そのような対象世界の典型例として衝撃波管内を伝播する衝撃波 [15] の再現計算である**図 1** の「一次元衝撃波流れ」を取り上げた．この記述例は提案する OOJ に必要なすべての基本要素を含むので，以降一貫して用いる．

表 1 には OOJ の利用想定ユーザの想定や特性等をまとめた．彼らはいわゆる数値計算ユーザである．ゆえに表 1 のユーザ特性を反映させて OOJ の設計を行う．

2.2 離散化モデル，OO モデル，離散・構造化モデル

図 1 の例では連続媒体である空気の満ちた一次元空間を “セル” と呼ぶ微細な空間（厚さ Δ x）に区切る[*3]．微分方程式も，たとえば空間幅 Δ x のスケールで近似された差

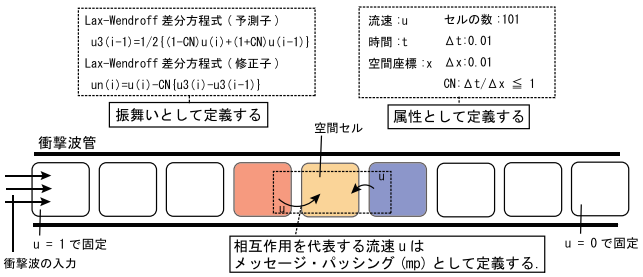

図 1　一次元衝撃波流れの離散化モデルの概念図

Fig. 1　A conceptual illustration of discretized model for one-dimensionnal shock wave flow.

1 茨城大学工学部
Faculty of Engineering, Ibaraki University, Hitachi, Ibaraki 316–8511, Japan
2 株式会社日立システムズ
Hitachi Systems, Ltd., Shinagawa, Tokyo 141–8672, Japan
3 群馬工業高等専門学校教育研究支援センター
Technical Support Center for Education and Research, Gunma National College of Technology, Maebashi, Gunma 371–8530, Japan
4 茨城県立産業技術短期大学校
Ibaraki Prefectural Industrial Technology Junior College, Mito, Ibaraki 311–1131, Japan
5 茨城大学工学部情報工学科
Department of Computer and Information Sciences, Ibaraki University, Hitachi, Ibaraki 316–8511, Japan
a) htkyma@mx.ibaraki.ac.jp

*1 本論文で多く用いる「プログラム開発」という用語は個人で開発する小規模なものを指す用語として用いる．「ソフトウェア開発」という用語は特に，ソフトウェア工学に基づく大規模な開発を意味する用語として用いる（文献 [5], [6] の表 2 参照）．またプログラミング言語（**PL** と略）の記述のみを**プログラム**と呼び，それ以外は**記述**と表現する．

*2 Object Oriented Japanese の略．

*3 図 1 は空間一次元のモデルであるが，多次元空間，時間軸やある種の位相空間を座標時空にとる技術も以前からある [15]．

表 1　想定ユーザの定義・特性・要求
Table 1　Definition, characteristics and requirements of assumed users.

(A) **ユーザイメージ：**OOJ の想定ユーザの中心（設計ポイント）は理工系の大学院生とする．彼らは主として科学技術計算に関わる研究や技術開発，解析業務計算に携わるための訓練を受け始めたプロの卵である．彼らは対象世界に関する一応の水準の知識やモデルをすでに持っており，分析も自力で記述する．彼らは分析の記述内容 [5], [6] が忠実にプログラムに反映されることを前提に記述する．
(B) **許容範囲：**それ以外にも理工系の高学年以上から，すでに現役のプロの研究者や技術者 [16] までが想定ユーザの許容範囲である．分野は学術の専門分野に限らず，数式等の計算を必要とする実用的な，あるいは趣味や興味のある分野も入る．
(C) **最大目標は数千行程度（10,000 行未満）の小規模プログラム**を個人自力で開発できる技術の習得である．そのために「PL の教育」以上で「ソフトウェア工学の高度な知識や技術」未満の**プログラム開発過程**の基礎技術を経験して習得することである．
(D) **関心：**彼らの主な目的や関心はプログラムではなく，計算結果であり，対象世界の新たな知見や設計データ等を得ることにある．
(E) **作業と経験：**多くは手続き型のプログラミング言語 1 つを利用に必要な範囲で知っており，少なくとも数百行程度のプログラムは作成し計算した経験がある．最初から一貫して**個人単独**で，**自身利用**のプログラム開発を行う．本項目が全ユーザの共通必須条件である．
(F) **意欲：**ソフトウェア開発技術の膨大な知識やノウハウ，高度な OO 技術，専門外の学習や訓練はほぼ期待できない．プログラム作成までの負担は最小限とし，プログラミング・フリーを望む．日本語の利用は歓迎するが，見知らぬ PL や自然言語処理技術等は避ける．

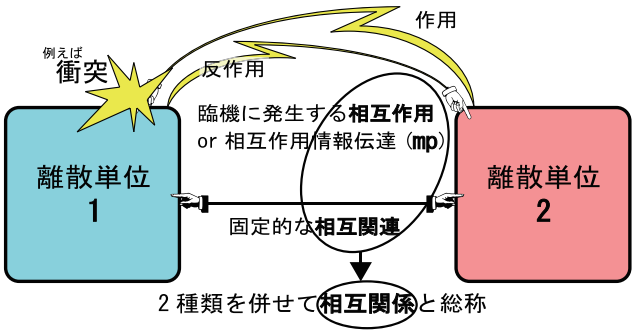

図 2　離散・構造化モデルの定義図
Fig. 2　A definiton diagram of discrete and structured model.

分方程式に変換する．この作業は離散化と呼ぶモデリングであり，**離散（化）モデル**といわれる．このモデルの単位である**離散単位**[*4]にセルという名前を付ける．各セルの物理量等を離散単位ごとの変数として設定して配列をとり，たとえば図 1 の Lax-Wendroff 法の差分方程式を使って各セルの物理量を計算するプログラムに組み上げてゆく．これが離散化モデルに基づく手続き型の計算プログラムであり，典型的には Solver と呼ばれる計算式の塊モジュールを作る [15], [17]．現在の数値計算では最も多く使われている数値計算用のモデルである．

一方本論文ではこの離散化モデルとオブジェクト指向[*5]モデルの目的に合わせて選択して組み合わせたモデルを構築する．まず図 1 のセルを改めて離散単位として着目する．離散単位個々の振舞いとしての数式（たとえば差分方程式）と振舞いに必要な属性を持たせる（図 1 上部に示す）．そして**図 2** のように複数の離散単位間では相互作用が発生するので，相互関係を付与する．この相互関係を付与することを**構造化**と呼ぶ [18], [19]．この「離散化」と「構造化」というモデリングを行った近似表現モデルを**離散・構造化モデル** [15], [17], [19], [20], [21] と呼び，離散単位間を相互関係で結んで構造化した近似モデルで対象世界を記述する

*4　ただし離散単位であることは特に強調的には意識されない．
*5　以降，必要に応じて **OO** と略す．

表 2　離散（化）モデル，離散・構造化モデル，OO モデルの比較
Table 2　Discrete model, Discrete and structured model, OO model.

離散モデル	離散・構造化モデル	OO モデル
配列の 1 単位	離散単位	クラス，インスタンス
変数	属性	属性
計算処理	振舞い	メソッド，ステートメント
（非明示的）	相互関係（図 2）	関係
×	×	カプセル化（情報隠蔽）
×	×	継承
（非明示的）	集約（相互関連）	集約
計算処理	相互作用（図 2）	メッセージ/シグナル
×	×	ポリモルフィズム
×	×	動的結合

という考え方である．

この離散・構造化モデルにおいては図 1 の差分方程式からセル間の相互作用を行う物理量として流速 u を抽出する．このモデルの特に特徴的な点はセル間でやりとりされる相互作用の情報を，輸送される相互作用情報である流速 u を伝達するためにメッセージパッシング[*6]という情報モデルの一種を使って他のセルに送る計算方式をとっているという点である．この方法を用いて衝撃波が伝播する計算を実現した [21]．この計算方式をとることにより離散化モデルに基づく手続き型の計算を OO 計算に変換したことになる [21]．つまり離散化モデルの計算式（たとえば Lax-Wendroff 法の差分方程式）を mp に変換して置き換えることによって OO パラダイム準拠のモデルにしたのが離散・構造化モデルである．

2.3　離散化，離散・構造化，OO の 3 つのモデルの比較

離散化モデル，離散・構造化モデル，OO モデルの「3 つのモデル」の主要な構成の比較を**表 2** に示した．離散化モデルと離散・構造化モデルの違いは，第 1 点は離散単位が配列の 1 単位になり，振舞いの主体ではなく単なるデータ

*6　message passing．略して **mp** と記す．図 2 参照．

の集まりになってしまうこと，第 2 点は（たとえば）差分方程式がデータに対する計算処理になるか離散単位の振舞いになるかの違いが出る．離散単位の振舞いの一部が相互作用であるが，離散・構造化モデルでは図 2 のようにメッセージパッシング（mp）で行われるようになる点が大きく異なる．つまり離散化モデルからは離散化された物理量や単なる計算式である差分方程式を振舞いである mp に変換して受け取る．

離散・構造化モデルと OO モデルとを比較すると，離散・構造化モデルの事項は OO モデルには必ず対応事項があるが，逆は必ずしもそうではない．OO モデルには設計や実装段階のプログラミング関わる事項もかなり入っているが，離散・構造化モデルにはまったく入れていない．したがって離散・構造化モデルは OO モデルのサブセットすなわち，物理世界に対する分析段階に特化したモデルであることが分かる．つまり離散・構造化モデルは分析段階限定の内部構成要素を物理世界の構成構成である**離散単位，属性，振舞い，相互関係**に読み替えて受け取ったことが分かる[*7]．

以上の離散・構造化モデルの構成に関わる 3 つのモデルを**図 3** に示す．図 3 の (1) 離散化モデルと (2) OO モデルから (3) 離散・構造化モデルが構成される方法が分かる．離散・構造化モデルは離散化モデルと OO モデルの特長を失わずに統合できることに着目し，それを想定ユーザが実利用ができる形に再構成した点にある [21]．

以上，離散・構造化モデルについて現時点でいえる特徴は，以下の 2 点である．

(1) 離散・構造化モデルは離散化モデルの延長線上にあるので手続き型の科学技術計算を行ってきたユーザにも理解しやすいモデルである．

(2) 離散・構造化モデルは OO モデルでもあるので，分析からプログラムまで一貫して使える．

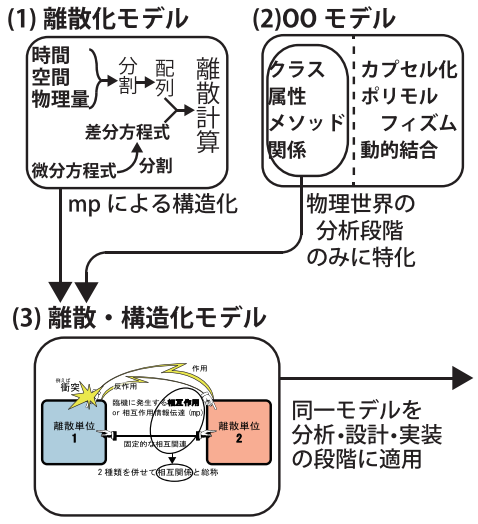

図 3　離散化モデルと OO モデルに基づく離散・構造化モデルの構築
Fig. 3　Discreted and structured model based on discreted model and OO model.

[*7] 表 2 において離散・構造化モデルにはない事項，たとえばプログラミングに関わる事項は離散・構造化モデル以外で記述される．

2.4　分析からプログラムまで一貫した記述言語系の必要性

1 章で述べたように OOJ 開発の狙いは「客観的に信頼できるプログラムとその数値データ」つまりは客観的な信頼性（reliability）の裏付けである．簡単なあるいは効率的な（開発期間の短い）プログラムの開発方法ではない．

より具体的にいえば，基準となる分析段階の記述と同等内容の処理を行うプログラムを得たい，ということであり，その実現には，分析からプログラムの段階までの記述と変換の詳細を「信頼性を確保・保証」しつつ追跡（トレース）できる仕組み」の構築であると考えた．その仕組みを具体化すると，以下のようになる．

(1) 分析段階から一気にプログラム段階にゆくのではなく，複数の中間段階も新たに設けて，その段階においても本節冒頭の狙いを実現するためのチェックと適切な変換を行う仕組みを構築する．具体的には設計と実装の 2 つの段階を新たに設けて，分析から設計・実装・プログラムまで合計で 4 つの段階を設ける．

(2) 数値計算ユーザが現状で使っている離散化モデルを拡張してプログラムまで一貫して使えるモデルを構築する．そのためには，同じモデリングパラダイムで一貫して使える OO パラダイムが必須である．それはすでに前節で構築した離散・構造化モデルである．

以上の OOJ の基本構成によって 4 段階の変換に従って一貫して追跡するという概念である一貫追跡性（integral traceability）の概念は実現できる．しかしさらには，4 つの段階間の記述間あるいは変換に対してある「特性」を持たせることが必要である．それが（一貫）相似性である．この一貫相似性という特性を離散・構造化モデルと組み合わせることで，OOJ の基幹構成ができあがる．

2.5　一貫相似性の概念と用語

本論文の特別な用語である（一貫）相似性の定義を行う．

相似性：ある段階と次の段階の **2 段階の間**の対応するすべての離散単位[*8]（群）と離散・構造化モデルにおいて**表現は異なっても同等/同値の記述や構造になる特性**を指す用語であると定義する[*9,*10]．

[*8] 最小から最大までのすべて（every）の離散単位を指す．通常典型例としては，オブジェクトに相当する離散単位を指すが，厳密には相互関係自体も離散単位である．

[*9] ただし完全に同等/同値でなくても，近似的に同等/同値であることも許す場合がある．その場合は相似性の高さを数値等で評価する仕組みを構築すればよい．

[*10] 本論文の「相似性」という用語の概念は，分野は異なるが類似性，同値性，相同性，双模倣性 [22], [23] 等の用語の概念にも比較的近い．しかし用法や意味，適用分野等が多少異なる．そこで誤解を避けるためあえて特別な用語を採用した．（付録参照）

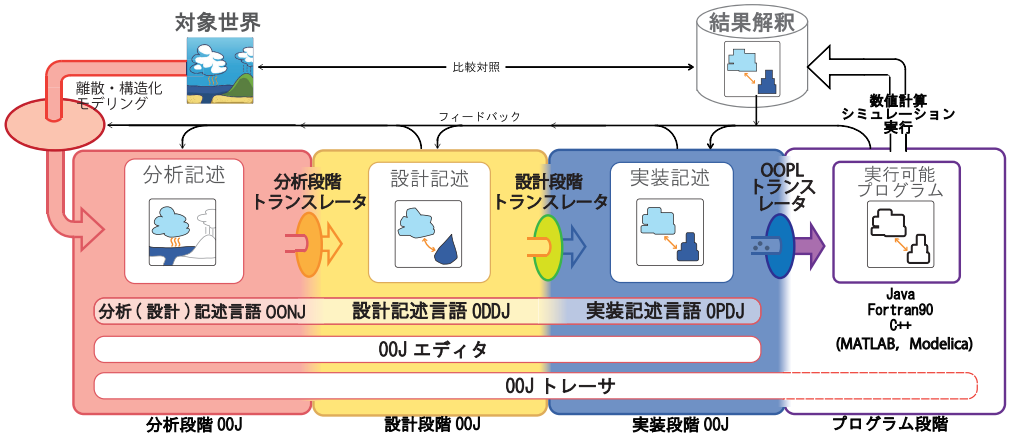

図 4　記述言語系 OOJ を基盤としたプログラム開発過程の全体概念図

Fig. 4　A whole concept diagram of program development processes based on OOJ.

一貫相似性：分析・設計・実装・プログラムの **4** 段階の間の対応するすべての離散単位（群）と離散・構造化モデルが上記の相似性を持つことを指す．この特性を強調するために「一貫」という言葉を冠して使う[*11]．

相似性の静的側面と動的側面：静的側面とは記述そのものを指すので，2 つの段階間の記述を直接比較し，変換されていなければそれでよし，変換されていても同等/同値の記述や（静的な）構造を持つか否かを検討して相似性を判定すればよい．動的側面としては mp も含めた離散単位ごとの振舞いや動き，相互作用等の等価性を検討して相似性を判定する．理論的には双模倣性の概念を導入することで実現するが，この点については今後の課題とする．

2.6　一貫相似性を持たせた記述言語系 OOJ 構築

2.4 節の基本構成 (1) と (2) に 2.5 節の一貫相似性を加えた言語系を構築すればそもそもの狙いを実現できる，と考えた．

(1) 分析からプログラムの全段階で使うために PL ではなく記述言語系を採用する．
(2) 離散・構造化モデルを適用し，一貫相似性という特性を与えるための記述環境 [24] や，記述の変換の前後関係をチェックする仕組み [25] を新たに構築する．

以上 2.4 節と合わせて 4 項目が具体的な方針として計画された．その OOJ の全体概念図を**図 4** に示す．上記の項目が組み込まれている図であることが見て取れる．

3.　OOJ の構築方針と概念設計

前節の計画に基づいて OOJ のより具体的な構築方針を**表 3** に示す．根本方針はまず先に目的に沿った OOJ 全体の設計，その後サブ言語，というトップダウン設計にある．

[*11] 一貫相似性の模式図を用いた直観的な説明は，後の 5.2 節においてその検証手段であるトレーサの節で行う．

3.1　OOJ 内部の 3 つのサブ言語構成の設計（表 3 の【I】）

表 3 の【I】の (1)〜(4) は各々異なった観点からの要請から導かれた方針であるが，この 4 項目を単一言語とするのは不適切と考え，(1)〜(3) 各々をサブ言語構成にする方針とした．そして便宜上分析，設計，実装という 3 つの名称で段階分けし，各段階での作業を特定し，明確な区切りを持たせる方針とした[*12]．【I】の (1) が分析段階に，(2) が設計段階に，(3) が実装段階に相当する．(4) は OOJ を支援する記述環境である．

3.2　各段階のサブ言語の機能分担と特徴（表 3 の【II】）

サブ言語の分担と特徴を表 3 の【II】に示す．OOJ では分析段階が「対象世界の記述の作成」で，設計と実装の段階をまとめて「プログラムへの変換過程」である．

(1) 既存の分析段階の記述言語 OONJ [5], [6] [*13]は【II】(A) の条件，すなわち離散単位と構造 [15], [17], [19], [20] に忠実な記述を作成できる言語に設計された．
(2) 設計段階言語（ODDJ [*14]，同【II】(B)）は，離散・構造化モデルを忠実に引き継ぐとともに OONJ 記述との相似性を確保・保証しつつ，計算機処理できる記述規則に設計する．この段階で新たに記述されるデータ型とアクセス属性は相似性を崩さない[*15]．
(3) 実装段階言語（OPDJ [*16] [26]，同【II】(C)）特定の OOPL に関わる事項の扱いはこの段階に限定される．

[*12] これは実際のプログラム開発時への考慮とともに，(4) で分析からプログラムに至る全段階のトレースに必要だからである．
[*13] OONJ（Object-oriented Natural Japanese）の特性は分析設計記述言語と呼ぶ方がより正確であるが [5], [6]，段階としては分析段階なので，便宜上こう呼ぶ．
[*14] Object-oriented Design Descriptive Japanese
[*15] なお離散・構造化モデルから意図的に外れ，より複雑で高度なオブジェクト指向モデルを使った技術の適用も可能ではあるが，想定ユーザは望まない（表 1 の (F) 項）ので本論文では扱わない．
[*16] Object oriented Program Descriptive Japanese

表 3　OOJ の設計方針，各サブ段階言語の役割分担，OOJ 概念設計
Table 3　Design principles, role sharing of each sub languages, concept design of OOJ.

【I】OOJ の全体的な設計方針（設計者の立場から）

(1) 分析記述はユーザの分野特有の記述法・用語・用法・式をそのままユーザ・イメージに近い形で記述できる言語を実現する．：対象世界の知識や情報のほとんどは分析段階で記述に起こすので，ユーザ向けに**記述力の強い言語**が必須である．そこで主要言語を任意の専門用語を許しすべてのユーザが自在に使いこなせる**自然言語としての日本語**とする．日本語文は内容の概略が分かるだけでもよい．
(2) 設計や実装の段階ではユーザでなくては書けない・判断できない情報のみの提供を求める．：これにより想定ユーザの心理的な負担感を含めてプログラミングの負荷・負担を軽くするためである．それ以外は分析段階の情報を自動変換する方針とする．
(3) 最後にはそのままで実行可能なプログラムへ自動変換して出力する．：これももちろん，ユーザ負担を少なくするためである．ユーザはプログラミングの文法の微細な判別が必要な記法に神経を消耗させられるのを避けたい気持ちが強いので必須である．
(4) 記述環境としてエディタ，トランスレータ，トレーサを構築する．：エディタは物理的なモノ表現のイメージを扱うために，トランスレータは PL も含めて 4 つの記述間の変換のために，トレーサは 4 段階の記述を分析・追跡し，一貫相似性の検証のために構築する．

【II】内部の各サブ言語の機能分担と特徴

(A) 分析段階記術言語 OONJ	対象世界に関する情報で，プログラム変換に至るまでに必要となる情報（離散単位や構造についての情報）をすべて記述すること．最小離散単位の日本語文は単文であることが推奨される．そしてその離散単位の構造化を要求される．数式は各記述者の考えや通常使う記号と方式で記述してよい．この段階では計算機世界に関わる情報は記述しない．
(B) 設計段階記述言語 ODDJ	計算機特有な要素の追加（データ型やアクセス属性の追加）し，属性を変数と呼び変えて，離散変数に配列やリストを導入する．ユーザは日本語文を数学関数やライブラリ表現に変換する．設計段階では特定の PL 表現は使わない．
(C) 実装段階記述言語 OPDJ	特定 OOPL 向けのプログラムに自動変換するための準備記述を作る．現時点では Java，C++，Fortran90 に変換するためにこれら 3 つの OOPL 共通の言語仕様を満たす仕様を導入している．
(D) 記述言語系 OOJ	上記 3 つを統合したのではなく，一貫相似性という狙いを実現するためにまず先に OOJ を設計した．その目的に合わせてサブ言語に機能と役割をユーザに分かりやすく区分して分割設計した．（必要上個別のサブ言語を先に発表）

【III】OOJ の概念設計

1. サブ言語 3 つすべてで**離散・構造化モデル**（図 2）を用いる．設計段階以降ではこのモデルを特化した OO モデル（表 2）として使う．
2. この離散・構造化モデルを正確に表現できる分析記述専用の記述言語を設計する．（上記【I】の (1) の実現）
3. この離散・構造化モデルまたは特化した OO モデルを表現する設計および実装段階の記述言語を設計する．
   - 3.1　設計と実装の段階では，分析記述と等価な離散・構造化モデルとプログラムに変換し表現できる言語を設計する．
   - 3.2　設計段階の記述言語では，計算機世界の共通な表現に変換を行うための言語設計にする．
   - 3.3　実装段階の記述言語では，3 つのターゲット言語の共通な事項の変換を行う言語設計にする．
   - 3.4　各段階での記述を完成すればトランスレータに掛けて次の段階の記述に変換する．（【I】の (2) の実現）
   - 3.5　実装段階のトランスレータにおいては実装記述を実行可能なプログラムに自動変換を行う設計にする．（【I】の (3) の実現）
4. 以上の記述言語系，記述エディタ，トランスレータ，トレーサ等はすべて共通なデータ形式（XML）で定義しておき，改めて各段階 “内” および各段階 “間” の記述の任意な分析と変換前後の記述追跡を実行可能にして，将来ニーズに備える．（【I】の (4) の実現）

3.3　OOJ の概念設計（表 3 の【III】）

表 3 の【I】の設計方針を具体化し，想定ユーザの要求と【II】の言語機能の分担上の制約等を検討した結果，本論文では表 3 の【III】に記す概念設計とした．

(1) OONJ の記述規則は【III】の 1. と 2. を組み込んだ．
(2) 【III】の 3.1 と 3.2 の項目の離散・構造化モデルを OO モデルとして使って設計段階の言語を設計する．
(3) 実装段階では 3.3 から 3.5 の項目の目標とする複数の OOPL に共通の表現に変換する．
(4) 実装段階の記述完成形は 3.1 の項目の分析段階の記述と等価的になっていることが必要である．

4.　記述言語系 OOJ の詳細設計

4.1　OONJ の概要 [5], [6]

分析段階である OONJ では離散・構造化モデルを直截に記述できる仕様とした．それは**図 5** で示すように，図左上の離散単位を中心とし，相互関係を用いて離散単位間を構造化する仕組みを持っていることを指す．その仕組みは図下部のすべての離散単位に引き継がれて（継承されて）いる．それはすなわち離散・構造化モデルの構造を OONJ が持つことの証左である．つまり OONJ は図 2 の離散・構造化モデルを直裁に表現した記述言語であることが分かる．この図 5 の離散・構造化モデルを BNF 形式の記述規則に直したのが**表 12**[*17]（都合により 41 ページに置く）である．

これらの事実をユーザ視点から視覚的にも確認するために，空間セルの記述例を**図 6** に示す [28]．図 6-(A) は図 1 のセルを具体的に図 6-(A-1) のように四角形の枠線を用いて離散単位のイメージ風に記述する．図 1 の内部には四角形の枠線を設けてその内部に振舞いや属性の集まりを記述する．これを中間離散単位と呼び，図 6-(B) に差分方程式の計算を 1 つの振舞いととらえた記述をまとめて示してある．この中間離散単位は図 6-(B-1) のように細長い四角形の枠線で囲み，その内部に図 6-(C) の日本語文が複数集約

[*17] OOJ の記述規則は “離散単位” であることを強調するため，離散単位を包む枠線を設けており，その枠線表現を記述規則に含めている．そのため記述規則においては枠線を含めた最大離散単位を＜フレーム＞[27]，同じく中間離散単位を＜スロット＞，同じく最小離散単位を＜サブスロット＞と記述している．

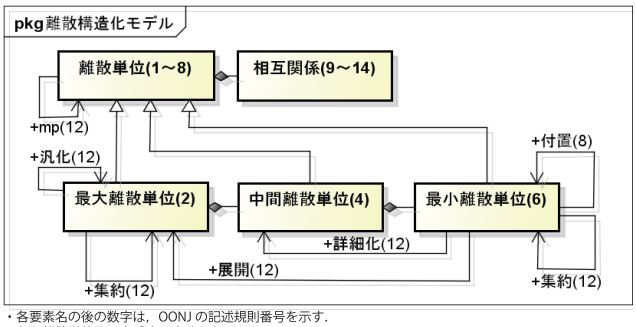

図 5　離散・構造化モデルに基づく OONJ の全体構造

Fig. 5　Whole structures of OONJ based on the discrete and structured model.

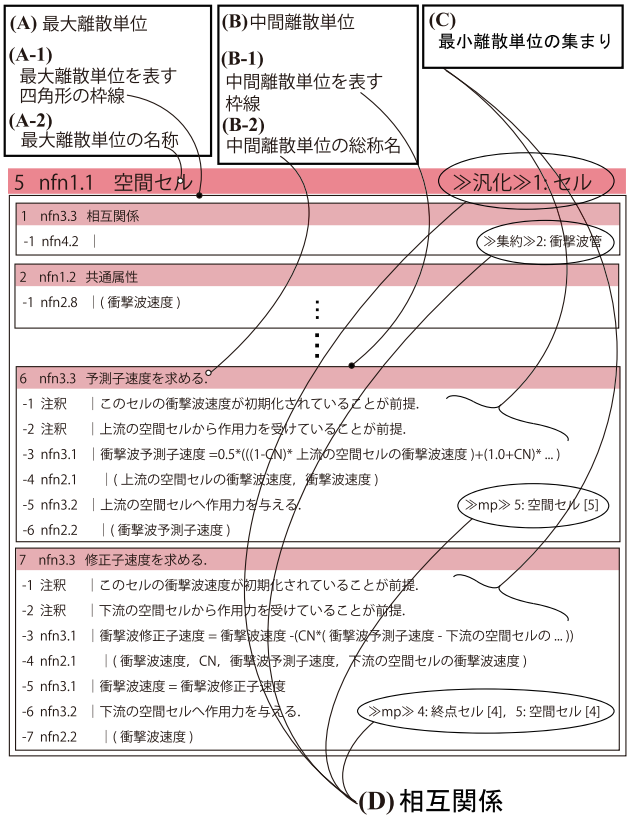

図 6　最大離散単位の空間セルの記述例（記述画面のみ切り出し）

Fig. 6　Description example of the maximum discrete unit “Space Cell”.

される．図 6-(D) は相互関係で，離散単位の相手方を指すことで構造化を表現している．この例からも OONJ が離散・構造化モデルを直截に表現していることが分かる．

4.2　ODDJ や OPDJ への変換の考え方と扱い方

図 5 を参照しながら**表 4** に記述規則の変換方法の考え方の概略を示した．全体の設計方針は分類 0. に従って計算機世界内部の表現に変換する，すなわち表現形式を変更すること以外の変換を避けることである．その典型例が，分類 1. のように対象世界の全体構造を表現するための集約と汎化の階層構造を「OPDJ まで維持する」という非変換ルールである（分類 1.1，1.2，1.3）．同様に分類 4. ～6. でも変換は行われない．離散・構造化モデル（OONJ の記述規則（表 12 の 1～14）から分かるとおり，表現形式の変換はあるが，計算内容に関わる変換はない．

設計段階では，分析段階の最小離散単位の日本語文を変換する．最小離散単位は分類 0. の原則に沿って 5 種類の命令文（の集まり）に変換する処置（分類 2.1）で対応する．この命令は式（計算式，評価式，条件式，break も含む），変数宣言，仮/実引数，コメント，メソッド呼び出し（return も含む）の 5 つである．初めの 4 種類は離散単位内部の処理であり，メソッド呼び出しは外部への処理であり，これにデータ構造を含めると離散・構造化モデルを計算機で表現したものになることは明白である[*18]．

計算機内部の処理には引数をともなうことが多い．その引数には属性が相当し，分析段階であらかじめ付置されているはずなので，それを充てる（分類 2.2）．最も大きく変換されるのは相互作用である（分類 3.）．OONJ では対象世界において可能な限り広範な分野で書ける一般的な形式で設計した．特にメッセージパッシング[*19]の場合がそうであり，相互関連の場合は変更はない（分類 3.1）．mp の場合はまずその記述形式が大きく変わる（分類 3.2）のでそれに対応する記述規則を設計した．

実装段階の追加と変換の作業は上述の考え方とほぼ同じ方式で行われる．ただしその作業内容に関する方針は前節と異なり，相似性を持たせた「プログラミング言語表現（すなわちプログラム）」に一意に変換できる記述規則に変更する必要がある．OPDJ は複数種 OOPL の文法を抽象化した設計を取り入れる．

4.3　ODDJ や OPDJ の記述規則への変換

前節の考え方と方法で行った具体作業を示す．そのために，**表 5** と**表 6** に 3 つの言語の記述規則番号とその変換の対応関係を示す．まず OONJ の記述規則に存在する 1～34 までを表 5 の最も左の段に置く．表内で “←” の記号は前の段階の言語と記述規則が同じ（変わらない）ことを意味する．記述規則 34 までの大半が変化しないので，残りを相似性を確保・保証する規則に設計すればよい．

検討すべき記述規則は**表 7** にある．規則番号に付された D や P は設計段階や実装段階での規則であることを示し，その実体は**表 8** に示される．番号のない記述規則は変更されないことを示す．これらの変換作業の内容を OONJ の記述規則番号と照らし合わせ相似にあるいは離散・構造化モデルを使って変換されているか否かを個々に検討した．その結果は表 4 の最右欄に示した．

[*18] 他の PL では，この種類以外に計算を簡潔にするための処理等が定義されているが，この 5 種類を特化した処理であるため，OOJ では拡張可能な設計にして対応している．Modelica [29] でもこの 5 種類を基本構成とした設計をしている．

[*19] mp と省略して使う．離散単位間の相互作用情報の伝達に用いる．

表 4　OONJ から ODDJ への記述規則の変換ルールの考え方と方針
Table 4　Concept of transformation rules of descriptive rules from OONJ to ODDJ.

分類	変換ルールの概略　（表中の括弧付きの番号は表 12 の規則番号である）	規則番号	相似変換
0.	根本方針は，OONJ 記述規則の**相似な変換**または可能な限り変更なしでの OPDJ までの維持である．		○
1.	集約階層と汎化階層の構造記述は設計および実装段階でも変更はなく，OPDJ まで維持する．		○
1.1	＜対象世界 (1) ＞の全体構造はそのまま OO プログラムに変換されるので，変更なしで OPDJ まで維持される．	1	○
1.2	＜フレーム (2) ＞はクラスかモジュール相当に変換されて，OPDJ までそのまま維持する．	2	○
1.3	＜スロット (4) ＞も OPDJ まで維持する．ただし，関数（メソッド）に変換され，＜スロット総称文 (5) ＞は関数名（メソッド名）とする．関数名の後ろに引数とデータ型，アクセス修飾子を追加する．	4，5	○
2.	＜サブスロット (6) ＞の実体は，規則 7 と 8 である．	6	○
2.1	＜サブスロット主文 (7) ＞は＜数式サブスロット＞，＜駆動 mp サブスロット＞，＜戻り mp サブスロット＞，＜両向き mp サブスロット＞，＜ライブラリ呼び出し＞の命令文のいずれかに変換する．	7	○
2.2	＜付置属性文 (8) ＞は命令文の引数（argument）に変換し，データ型とアクセス修飾子を追加する．	8	○
3.	＜相互関係 (9) ＞はその意味内容からユーザ判断で計算機の命令へ対応する．	9	
3.1	＜記述規則 (9〜14) ＞までの中で変換は mp の場合のみで，他は OPDJ まで変更はない．	9-14	○
3.2	mp 記述は受信側の記述は削除し，発信側のみに呼び出し（call，駆動 mp）か戻り（return，戻り mp）のいずれかに変換する．あるいは送受信を表す両向き mp“≪ mp ≫” に変換する．		○
4.	各離散単位の属性（各離散単位ごとの一連番号，ファセット記号，名称/総称）はそのまま OPDJ まで維持する．	3	○
5.	＜要素特定子 (25-30) ＞と＜記述位置指定子 (31-34) ＞は表現上の規則なので，そのまま OPDJ まで維持する．	25-34	○
6.	記述規則中には明示されていない＜対象世界共通要素フレーム群 (23) ＞と＜フレームヘッダスロット (24) ＞がある．これらは種類としては通常の＜フレーム (2) ＞や通常の＜スロット (4) ＞であり，それぞれ (2) や (4) の規則が適用される．ただし特定の内容を記述するので，ガイドラインや標準スタイルを提供している．	23,24	○
7.	＜日本語文 (15) ＞は設計および実装段階でライブラリ等の新規な規則として定義される．	15-22	

表 5　OONJ から ODDJ，OPDJ への変換規則対応表
Table 5　Correspondence of OONJ to ODDJ and to OPDJ.

OONJ 規則番号	ODDJ 規則番号	OPDJ 規則番号	理由説明等
1〜3	←（注 1）	←	
4	4D	4P	
5	5D	5P	
6	6D	6P	
7	7D 新	×（注 2）	ODDJ 以降の新規則へ
8	8D	8P	
9〜10	←	←	
11	←	11P	
12	←	12P	
13	←	←	
14	←	14P	
15（注 3）	×	×	
16〜17	×	×	ODDJ 以降の新規則へ
18〜19	←	×	OPDJ 以降の新規則へ
20	20D	20P	
21	21D	21P	
22	22D	22P	
23〜24	←	←	
25〜30	←	←	
31〜34	←	←	

（注 1）左矢印は前段階と同じ規則であることを指す．
（注 2）×印は対応する規則が存在しないことを指す．
（注 3）規則 15 番の実体は 16 番から 22 番までに当たる．

次に OONJ には存在せず ODDJ から始まる記述規則がある．それが表 6 の “新” という文字を付した ODDJ 起源の記述規則番号である．その実体の記述規則は表 8 に示してある．このカテゴリの記述規則は個々に計算機処理の世界で特定する何かが OONJ 記述規則に反する，つまり相似性を崩す記述規則であるか否かを個々に検討する．

表 8 の【A】の 35D〜38D に関してはデータ型とアクセス修飾子なので相似性を崩す規則ではありえない．39D〜

表 6　ODDJ と OPDJ 起源の記述規則の対応表
Table 6　Correspondence of OONJ to ODDJ and to OPDJ.

OONJ 規則番号	ODDJ 規則番号	OPDJ 規則番号	理由説明等
×	35D 新	35P	計算機表現への変換規則
×	36D 新	←	同上
×	37D 新	←	同上
×	38D 新	←	同上
×	39D 新	39P	同上
×	40D 新	40P	同上
×	41D 新	41P	同上
×	42D 新〜	←	OONJ の日本語文をライ
	〜105D 新	←	ブラリに定義した規則．
×	×	42P 新〜	ターゲット OOPL への
		〜50P 新	変換規則
×	×	51P 新〜	3 種のターゲット OOPL
		〜82P 新	から抽象化定義した規則．

41D までは OONJ では抽象的であった数式やメッセージパッシング（mp）を具体化した記述規則であり，内容の変更ではないので，相似性を崩す規則ではないことが分かる．それは【A】の ODDJ 起源の記述規則は段階的に計算を詳細化する規則だからである．また 42D 新〜105D 新という記述規則は OONJ の日本語文を変換するためにあらかじめ準備したライブラリであり，ユーザは適切に選択して使えばよい．

最後に OPDJ 起源の “新” 記述規則群（表 8 の【B】）は，いずれもターゲット OOPL の記述規則（文法）対応に定義された記述規則である．【B】は制御構文や変数や引数等の詳細化を規定した記述規則で，【A】と異なるのは変換先の複数の OOPL（現状では Java，C++，Fortran90）の共通

的な文法に変換するための記述規則である．これらの記述規則の内容も相似的な変換になった記述規則か否かを個々に検討すればよい．詳細は煩雑になるので省略するが，結果としては同じく問題なく相似的な変換であることを確認した．また 51P 新～82P 新までの記述規則は，式，関数やライブラリ等の呼び出し記述規則が定義されている．これらも同様にユーザの選択に関わるので，各々の式や関数，ライブラリがその定義どおりの設計になっていればよい．

なお，記述規則の変換の実作業は手作業ではなく，記述規則自体の変換作業を行う専用の変換プログラムを作って機械的（形式的）に行っている [28]．特に表 6 はかなり多数で複雑でもあり，ルーチンワーク的な手作業はミスも多くなるので，変換ルールとして ODDJ では 105 個，OPDJ では 82 個定義して組み込んでいる．また変換プログラムの外部にも記述規則を定義する仕組みを設けて柔軟に規則を変更可能にした．言語間の変換を客観的に評価したり，変更や改訂を行ったりするには特に有効である．

表 7　OONJ から ODDJ へ，ODDJ から OPDJ への変換規則
Table 7　Transformation rules from OONJ to ODDJ and to OPDJ.

規則番号	**OONJ からの変換を定義する．**
4D	右辺に付置属性記述を付与．
5D	右辺にデータ型とアクセス修飾子を追加．
6D	右辺のサブスロット主文を複数の命令文へ変換．
7D 新	39D 新～41D 新の規則として新たに追加．
8D	右辺にデータ型を追加．
4P	右辺の付置属性記述を仮引数へ変更．
5P	右辺の相互関係を削除．
6P	右辺のサブスロット主文を複数の命令文へ変換．
8P	付置属性を仮引数や実引数に変換．
11P	複数の相互関係相手先を単一の相手先に変換．
12P	右辺の一部を削除（詳細省略）．
14P	右辺の一部を削除（詳細省略）．
20D	注釈文，つまり “コメント” 扱いになる．
21D	“nfn” の表記を “dfn” へ変換．
22D	ODDJ の離散単位の種類表（省略）の対応へ変換．
20P	注釈文，つまり “コメント” 扱いになる．
21P	“dfn” の表記を “pfn” へ変換．
22P	OPDJ の離散単位の種類表（省略）の対応へ変換．

4.4　OOPL プログラムへの自動変換

【OOPL プログラムへの変換方法】

OPDJ 記述はプログラム段階の OOPL トランスレータ（図 4 の右側）によって 3 種の OOPL（Java，C++，Fortran90）のプログラムに変換される．このトランスレータ [26] は OPDJ と 3 言語間の対応関係を基に unfold/fold 方式 [30], [31] を実際に適用処理する．本トランスレータにおいても OPDJ 記述と OOPL プログラム間の相似性を保つ設計と実装としたので，OPDJ の仕様の範囲内で相似

表 8　ODDJ および OPDJ 起源の新規な記述規則の定義
Table 8　Newly defined descrption rules and transformation rules.

【A】ODDJ 起源の規則

規則番号	**計算機での共通な表現への変換するための規則の新設や変更を行う．**
35D 新	＜付置属性記述＞:=:(＜ lf ＞＜付置属性文＞)$^{+}$
36D 新	＜計算機特定指定子＞:=:＜ sp ＞＜データ型＞＜ sp ＞＜アクセス修飾子＞
37D 新	＜データ型＞:=:(＜整数型＞｜＜実数型＞｜＜論理型＞｜＜文字型＞｜＜文字列型＞ ｜＜整数配列型＞｜＜実数配列型＞｜＜文字配列型＞｜＜ void ＞｜＜フレーム名＞)
38D 新	＜アクセス修飾子＞:=:(＜対象世界共通＞｜＜フレーム内共通＞｜＜スロット内共通＞)
35P	付置属性文を仮引数に変換．
39D 新	＜数式サブスロット＞:=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞＜数式＞＜相互関係＞$^{?}$
40D 新	＜スロット呼び出しサブスロット＞ :=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞＜日本語文＞＜相互関係 (駆動 mp) ＞＜付置属性記述＞$^{?}$
41D 新	＜戻りサブスロット＞ :=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞＜日本語文＞＜相互関係 (戻り mp) ＞＜付置属性記述＞$^{?}$
39P	＜数式サブスロット＞:=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞＜式＞
40P	＜スロット呼び出しサブスロット＞:=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞＜スロット呼び出し＞
41P	＜戻りサブスロット＞:=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞ “return” ＜単純式＞
(42～105) D 新	ユーザが日本語文を数学関数やライブラリで記述する規則．＜三角関数＞，＜乱数＞，＜双曲線関数＞，＜誤差関数＞等．

【B】OPDJ 起源の規則

規則番号	**ターゲット OOPL のプログラムへの変換のための規則の新設や変更を行う．**
42P 新	＜反復脱出文＞:=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞ “break”
43P 新	＜反復飛ばし文＞:=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞ “continue”
44P 新	＜変数＞:=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞ (＜変数名＞｜＜実値オブジェクト名＞) (“[” ＜要素数＞ “]”)$^{?}$(“=” ＜単純式＞)$^{?}$ ＜ rt ＞＜計算機特定指定子＞
45P 新	＜コメント＞:=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞ “(コメント)” ＜日本語文＞
46P 新	＜そのまま出力文＞:=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞＜ユーザ自由記述＞
47P 新	＜仮引数＞:=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞＜仮引数名＞＜ rt ＞＜計算機特定指定子＞
48P 新	＜実引数＞:=:＜単純式＞
49P 新	＜式＞:=:(＜単純式＞｜＜比較式＞｜＜論理式＞｜＜代入式＞)
50P 新	＜単純式＞:=:(＜文字列式＞｜＜算術式＞)
(51～82) P 新	3 種のターゲット OOPL から抽象した規則．＜加減法演算子＞，＜大小比較演算子＞，＜文字列定数＞，＜数字＞等．

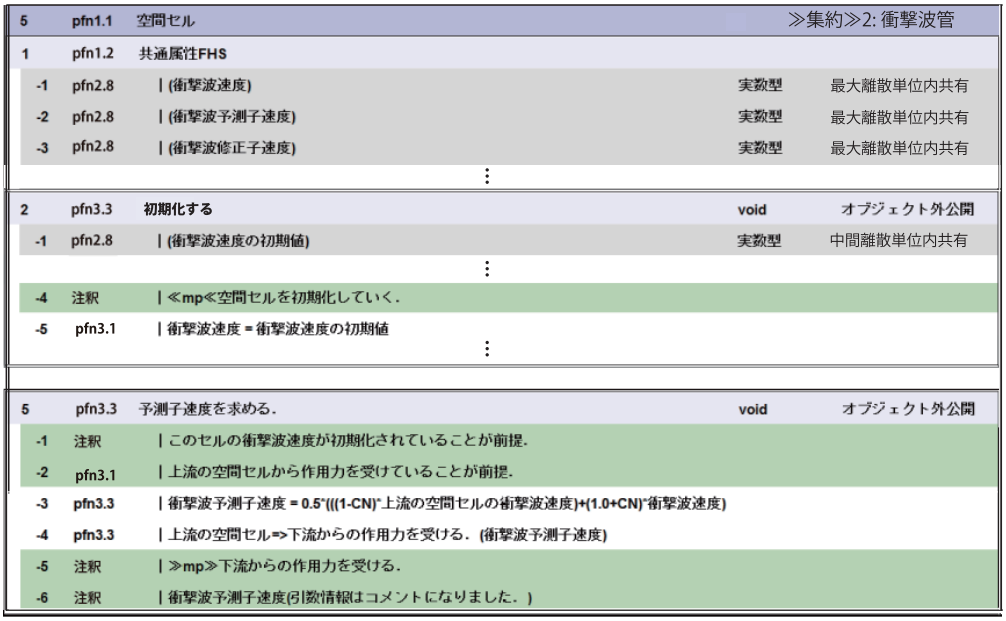

図 7　空間セルの OPDJ 記述例（日本語文が注釈に変換され，緑色で説明文になっている）
Fig. 7　Description example of the space cell using OPDJ.

性が保証される．ただし両段階間のトレーサによる検証機能は未提供である．

Java と C++はクラスベースのオブジェクト指向言語である．そのため，OPDJ におけるオブジェクトの定義と実体はそれぞれクラスとインスタンスに直接に対応付けできることで変換を実現している．一方，Fortran90 はモジュールをクラスの定義に，クラスの属性部分を構造体に対応付けることで OPDJ 記述を Fortran90 のプログラムへの変換を実現した[*20]．

【OOPL プログラムへの自動変換】

図 6 の OONJ 記述例を OPDJ 記述に変換した記述例を**図 7** に示す．図から変換先 OOPL のプログラムとの対応関係が明確に見えること，コメント化された日本語文から変換前の処理内容が明確に分かるのが特長である．ユーザはこの記述からプログラム変換前の最終確認を行える．

図 7 の OPDJ 記述例では全行数は（コメント行を除いて）80 行であった．図 6 と図 7 の記述例の場合は，人手をまったく加えることなく 3 種類の OOPL のプログラムに正常に自動変換された．Java では約 250 行，C++では約 320 行，Fortran90 では約 610 行のプログラムとなった．実際の計算結果は 3 言語ともに同じ計算結果を出した．

[*20] 本変換方法では別々のモジュールの関数どうしで循環参照が発生する場合がある．これを防ぐために，Java プログラムへの変換ではオブジェクトの定義を属性部分とメソッド部分に分け，オブジェクトの属性を構造体として対応付け，メソッドをプログラムの内部関数と対応付けている．C++プログラムへの変換では事前宣言によってクラス内のメンバ関数が宣言された後からその実体が定義され，Fortran90 プログラムへの変換ではオブジェクトを構造体と内部関数に分ける変換を行った．これらの処理機能はいずれもトランスレータに実装された．

4.5　記述言語系の記述規則の段階的詳細化

OOJ の規則体系を段階的詳細化という視点で見る．

【段階的詳細化と一貫相似性】　そもそも OONJ の記述規則では離散・構造化モデルによって対象世界の全体構造を規定している．この離散・構造化モデルに関わる規則は表 5 から明らかなように（最小離散単位の実体である日本語文を除いては），OONJ から OPDJ までは変更はない．したがって OOJ は離散・構造化モデルについては一貫相似性を実現していることが分かる．

一方，最小離散単位の内容である日本語文はそもそもの設計方針の 1 つとして，OONJ 記述開始当初は日本語文では概略の内容を記述[*21]していればよく，プログラムに変換可能な記述の完成は ODDJ 以降でよい，という方針を表 3 の【I】の (1) において提案した．表 5 の理由説明欄に「ODDJ 以降の新規則へ」とあるが，これらが “概略の日本語文” から変換すべき ODDJ と OPDJ からの新規な規則である．これらの規則の大きな特徴はすべてが最小離散単位の日本語文を数学関数やライブラリ，計算式等にするための記述規則である点である．そこで OOJ エディタを使って日本語文の内容と同等な数学関数やライブラリを使った計算式等にユーザ自身が書き直せば相似性を実現できる．

【段階的詳細化の方法】　OONJ 記述規則の 7 と 16～20 は ODDJ と OPDJ の “新” 記述規則で規定される．“新” 規則は表 6 にあげた “D 新” と “P 新” が付された番号の記

[*21] OOJ では最初から最後まで同一人の個人だけで作業する，という前提で設計している（表 1 の (E) 項）ので，OONJ では概略だけ記述し，後で必要に応じて詳細化したり正確に式に書き直したりすることが可能かつ十分であるとしている．

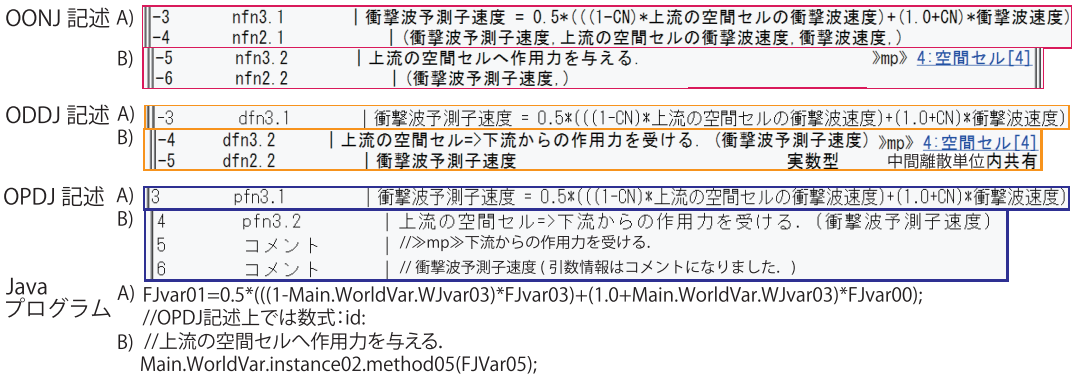

図 9　最小離散単位ごとの日本語文の一貫相似性を持った記述例

Fig. 9　An integrally consistent and bisimilar description examples of the Japanese sentences.

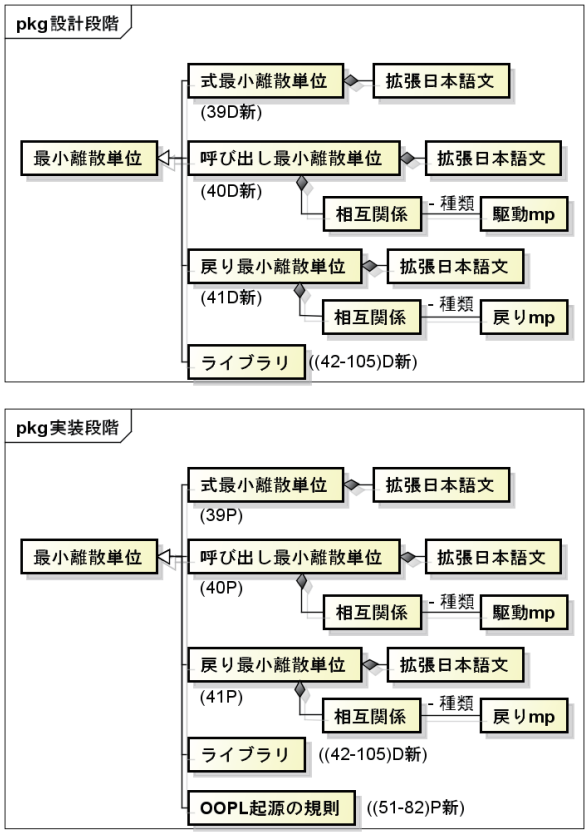

図 8　設計と実装の段階の最小離散単位の段階的詳細化

Fig. 8　Stepwise refinement of the minimum discrete unit in the design and implement stage.

述規則である[*22]．つまり OONJ では日本語文が「その趣旨や内容だけが分かればよい」記述をしたのに対し，表 6 と表 8 は日本語文のみを扱う完成記述用の規則とその実体を定義した．

【段階的詳細化を表すメタ表記図】　以上の段階的な詳細化に基づく設計と実装段階の規則を図に表したのが図 8 である．図 8 の左端の最小離散単位は OONJ の図 5 右端の最小離散単位に相当する．設計と実装の段階において図 5 の他の部分は内容的にはほぼ変化がなく（表 5 と表 7 参照），最小離散単位は，式，mp，ライブラリ，OOPL 起源の規則に特化されたことが分かる．

まとめると分析段階では離散単位間の構造のみを正確に記述し，日本語文は概略的な記述にとどめておく．設計と実装の段階では最小離散単位の日本語文の部分のみを OOPL に変換可能な記述にするという方式の段階的詳細化を採用した．それが特長である．

4.6　OOJ に組み込まれた一貫相似性の仕組みの検証

(1) **離散・構造化モデル（規則番号 1〜14）の一貫相似性**は，表 5 と表 7，これらに完全準拠で設計された OOJ エディタとトランスレータによって保証される[*23]．これを**規則保証**と呼ぶ．

(2) **日本語文の一貫相似性**は，表 5 から分かるように，規則 15 <日本語文>に関わる規則 16〜19 によって “新” と名前付けされた ODDJ と OPDJ の規則に準拠したエディタ上で提供される数学関数やライブラリを使った式や制御構造文に変換することで一貫相似性を保証される．これを**記述環境保証**と呼ぶ．

(3) **ユーザ記述の計算式や処理文に対する一貫相似性**　日本語文の変換には意味内容（semantics）の一貫相似性も必要である．これには形式的な 100%変換は不可能[*24]であり，ユーザ自身がエディタの支援の下で日本語文ごとに相似性を保つ変換を行う．ユーザが起こす

[*22] ODDJ 起源の規則（D 新）は OONJ 記述を計算機表現に移行するための（OONJ からの影響の強い）規則である．OPDJ 起源（P 新）は OOPL のプログラム特有の記述にするための（OOPL からの影響の強い）記述規則である．両方向からの記述規則は段階を分けて記述すべきだと考えて設計した．

[*23] OPDJ から OOPL への変換は OOPL トランスレータ（図 4 右側参照）によって保証される．4.4 節参照．

[*24] 自然言語処理を用いた変換理論やシステムは存在するが，実用上 100%完全な変換はできず，表 1 の (E) 項によりユーザには拒否されたこともあって採用しなかった．対象世界と OOJ に関するユーザの知識と能力を考慮すれば，ユーザ自身がやるのが最も合理的かつ効率的であると判断した．

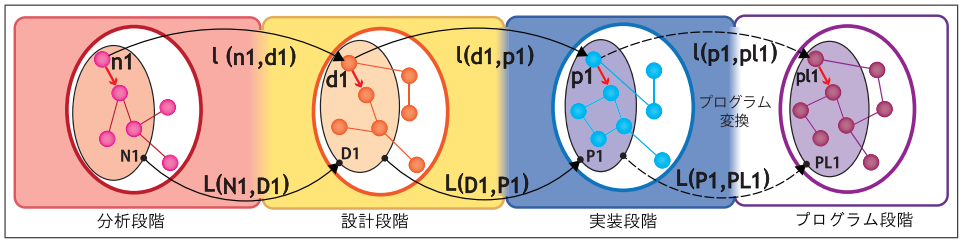

図 10　離散単位間の対応関係リンクの模式図
Fig. 10　Correspondence links among discrete units.

ミス等は特殊なトレーサ [25], [32], [33], [34] *25によってカバーする．これを**ユーザ・トレーサ保証**と呼ぶ．

以上をまとめると，一貫相似性は特定の記述規則によるのではなく，4 つの各段階前後の記述規則間の関係の特性，すなわち内容が実質的に分析段階の記述と同等内容を維持するという特性の実現によって確保・保証される．具体的には上記の (1) と (2) で形式的に扱える部分は OOJ の言語仕様と OOJ エディタで実現し，(3) では意味内容 (semantics) の変換の一貫相似性をユーザ自身の変換と特殊なトレーサを使って行う．

最小離散単位の日本語文の一貫相似性を実現している 2 つの変換記述例を**図 9** の (A) と (B) に示す．(A) の式計算は表現形式が変わるだけで内容は変わらず一貫相似性が実現している．(B) の日本語文は相互作用情報伝達 (mp) に順次変換されて Java プログラムに至る．両記述例ともにそれらの内容を想定ユーザが見れば一貫相似性が十分に保たれていることが分かる．

5.　分析記述の再現性の検証と信頼性向上
—OOJ 一貫相似性の従来より効果的な利用法—

5.1　プログラムとそれに至る過程の 2 つの信頼性

科学技術計算のプログラム開発で最も重要な点は，

(1) プログラムがユーザの分析記述 (OOJ では OONJ 記述) を正しく反映した開発過程 (開発手順・プロセス) を経ているか*26という**開発過程の信頼性**，

(2) 計算結果の信頼性つまり**分析記述の忠実再現性**，

という 2 点にある．この 2 点を合わせた概念を本論文では仮に**プログラムの信頼性**と呼ぶことにする．現状では経験的に判断したり，参考文献の計算結果との合致が良かったり，同じ条件の実験との比較で一致したりすればプログラムの信頼性を得たとされる．しかし，それらは上記 (1) と (2) についての直接的な解決ではなく別手段で間接的に判定している．開発過程はブラックボックスのまま残り，分析記述の忠実再現性は中間段階の変換の対応関係が不明なゆえにそのまま残り，問題は解決されていない．

OOJ では記述も開発過程も完全なホワイトボックス化し，分析記述のすべての離散単位に対する詳細な追跡性 (traceability) [36] と直接的な比較評価性を実現することで少なくともその一部は解決するのではないか，と考えた．そこで OOJ の目標を上記 (1) の開発過程 (プロセス) の信頼性の問題の解決を優先目標とした*27．ただし表 1 に沿って個人・小規模での実現を目標とした．

5.2　OOJ トレーサによる開発過程の一貫相似性の検証

4.6 節の (1) において離散・構造化モデルの一貫相似性を確保・保証できていることを論じた．残るのは 4.6 節の (2) の一部と (3) の日本語文である．つまり純粋にユーザ記述になる部分である．その解決には記述環境を使って作業過程をすべてデータとして残しているので個々の離散単位 (群) の変換過程を追跡 (トレース) し，個々の離散単位 (群) ごとにその変換前後の記述を比較して相似性を検討・評価する．それをすべての段階間のすべての離散単位 (群) 間で行えば一貫相似性を明らかにできる，と考えた．

図 10 に 4 つの段階の離散単位間の相互関連リンクを模式的に示す．各段階の丸印は離散単位，各段階内部の丸印と同色の線は図 2 の意味での相互関係を表現する線である．この相互関係線は各段階内部の記述に対する通常のトレーサ [25] の対象である．一方，黒の線で代表される 4 つの段階の離散単位間を結ぶ線は離散単位の変換前後の対応関係を示す．この線は特殊なトレーサである **OOJ トレーサ** [25] が扱う．これら 2 種類の情報は OOJ エディタにおいて default で作成されて別ファイルに保存される．

具体的には変換前後の 4 つの離散単位 n1，d1，p1，pl1 間の相互関連を示す黒のリンク “l” の変換前後の記述を取り出して比較し，3 つの相似性の高さを検討・評価すればよい．それら 3 つの相似性が同時に高ければ一貫相似性が高いあるいは成立しているという．同じように図 10 の N1，D1，P1，PL1 は構造化された離散単位群であるが，離散単位群間のリンク “L” について同じく一貫相似性評価が可

*25 この特殊なトレーサは，プログラム内部の実行手順を追跡する [35] のではなく，分析からプログラムに至る各離散単位 (群) 間の変換前後の関係を追跡するトレーサである．5.2 節に詳述．
*26 業務システムであれば概念設計や顧客の要求仕様等が使われるが，科学技術計算においては分析記述が出発点となる．
*27 分析記述の再現性は想定ユーザによって比較的容易に確認できるからである．なぜならば分析記述を行ったのは計算結果の再現性を望む本人自身だからである．また専門性が高くなればその確認は本人が最適である．

能である．なおOOJトレーサの特性は4段階の離散単位の一貫追跡性の実現でもある．

次章で述べる記述実験の際にも，OOJエディタ [24] を使って手作業で3つの段階の記述を並べて表示させ，比較検討して一貫相似性を確認している例も多かった．被験者の院生は全員，学部生にも複数いた．OOJの一貫相似性の確認機能のニーズはやはり大きいようである．OOJトレーサを用いれば短時間で効率的に評価可能である．詳しくは参考文献 [25] に譲る．

5.3　OOJ の V&V 評価システムへの適用の検討

5.1節の(1)と(2)のプログラムの信頼性は，現在さかんに検討されているV&Vの概念 [37], [38], [39] の一部とほぼ同じ概念である．V&Vとは当初の要求仕様や設計が正確な手順と過程（プロセス）を踏んでプログラムに正しく反映されていることの検証（Verification）を指す概念であり，開発の狙い・要求と意図が達成されたプロダクトとしての妥当性の確認（Validation）を指す概念である[*28]．これらはそれぞれ5.1節の(1)と(2)に相当する．

その観点でOOJの一貫相似性とV&Vとを比較すると，最終目標は同じであると考えてよいが，そのために用いる手法が大きく異なる．大きな相違点の1つは専用言語の構築の有無と汎用性，もう1つは検証プロセスにおける判断基準である．第1点はOOJという専用記述言語がある点とV&Vの求める汎用性ではなく表1の特化した目標に絞っている点である．第2点についてOOJにおいてはつねに一意に「一貫相似性」を確保・保証した変換作業プロセスを求められる．一方一般のV&Vの場合は一貫相似性に限定せず当初の要求仕様や設計条件，途中の制約条件等を新たに設定したうえでの「一貫整合性」が求められる．これらが大きく異なる．

そこでOOJを利用してV&V評価をしようとする場合は，設計方針を決め検証や確認の評価基準を決めて分析記述を作る．その際OOJエディタは離散・構造化モデルに対してつねにより強い制約である一貫相似性を確保・保証しようとするが，ユーザ側での変更は容易であり障害にはならない．それは設計や実装の段階においても同様である．OOJトレーサによる開発過程の信頼性に関する検証機構はそのまま利用可能である．したがってV&Vの一貫整合性を検証するのはOOJにとっては十分容易である．

つまりOOJの想定範囲内においてはV&V評価のための基礎準備はすでに存在しており，現用のOOJトレーサをV&V評価システムに転用することは十分に容易であり，実現は可能であると分かった [25]．ただし現状のOOJトレーサで実行できる機能は主として記述の同値性評価を行う静的なトレースである．ユーザは動的な側面の相似性も考慮して相似性を評価するゆえに一応は使えるが限定的であり，現状で出せる評価は定性的な結論に近い．本来はたとえば双模倣性の概念を適用して動的な振舞いや計算の側面について定量的で詳細な相似性評価が必要である．

[*28] 他にも解析業務の計算シミュレーションのV&V [4] や，工業製品の品質に関わるV&V [40] もある．

6.　従来より効果的なプログラム開発法の提案
—OOJ の一貫プログラム開発法の適用と考察—

本章ではOOJの主要な利用法の1つである「一貫したプログラム開発」へ適用してOOJのより効果的な利用方法の考案とその有効性を考察する[*29]．

6.1　OOJ の記述作業の特徴とその利用技術

【OOJとユーザの記述義務区分】　表5～表8，および表12までの記述規則のうち，専用のOOJエディタ [24] とトランスレータ [26] により，規則15～24までの日本語文に関わる規則以外はすべて（どの程度かを別にすれば）自動変換に関わっている．

したがってユーザが関わるのは離散単位名を含めた表12の規則15の＜日本語文＞の部分（つまり，図8）と考えてほぼ間違いない [24]．それらの多くの記述規則は数式や既開発のライブラリ（表9の（42～105）D新とOPDJ起源の規則（P新）の規則群）で形式的に変換可能なので，OOJエディタにより（半）自動処理できる．

【OOJ記述環境を使った記述作業の概要】　OOJエディタは表1の想定ユーザの要望や特性を反映して設計されており，OOJの使いやすさの多くの部分をOOJエディタに負っている．必要ならば変換結果の妥当性をOOJトレーサ [25] で確認できる．これまでの利用経験ではユーザ自身が理解して使える必要があるのは図2の離散・構造化モデルのみである．記述規則の参照はまず不要である．ユーザはマニュアルとドキュメント，OOJトレーサ等を使い，多くの記述例 [28] を参考に，図4の流れに沿って記述する．それが積極的に推奨されている．

以上のようにユーザが直面して習得すべきOOJの言語仕様の規模（記述規則の数）が小さく（見えるように）作られており，簡単で学習が容易なように設計された．その設計が特にOOJの一部であるOONJの記述性の良さ（[5], [6]）につながっている．さらには，3段階に分割されたエディタの各々の記述間はトランスレータで半自動変換され，最後のOOPLトランスレータによりOOPL（Java）に自動変換されてユーザの負荷を低減している．この間の変換の妥当性等はOOJトレーサを使えばよい．このOOJトレーサは第3者の他人が書いた記述例を読む際にも客観的な検証を行う際にも非常に有効であることが経験的に記

[*29] 本章はあくまでもOOJの適用有効性を検証するのが目的であり，教育方法の改善策等を目的とはしていないので被験者データの詳細な分析等は行わず，平均値を使って評価する．教育方法の改善策等については別の機会（文献 [41] 等）に譲り本論文では扱わない．

表 9　OOJ に対する大学院生の理解と記述力，記述しやすさに関するアンケート結果
Table 9　Results for the questionaire concerning the recognition, the descriptiveness, and the usability of OOJ for the graduate students.

アンケートの質問内容の概要と補足説明	OOJ の理解	記述ができた	記述のしやすさ
1. 離散・構造化モデルの概念（図 2 や図 4）や通常の OO モデルとの違い（表 3）を理解できましたか？.	5	5	4.4
2. 離散単位とその表現法について，図 5 のように離散単位に相互関係を付与する仕組みや最小離散単位までを一意に特定する方法，対象世界共通フレーム群の記述法等は理解できましたか？	5	5 △	5
3. 構造化とその表現法について，相互関係を付与する構造化の表現法を理解できましたか？特に最小離散単位間の構造化や（付置）属性の構造化の表現法が理解できましたか？	5 △	5	4.6
4. 日本語文の記述には特別な文法が不要なことを十分に理解できましたか？日本語文はモノや振舞いを表現すると同時に，離散単位として扱えるようにした趣旨は理解できましたか？	5	5	5 △
5. 一貫相似性離散・構造化モデルの概念（図 3）は理解できましたか？　3 つのサブ言語すべての段階で使われており，各段階間で表現は違っても内容的には同等/同値表現となるという特性を理解できましたか？ OOJ は分析からプログラムまでの「一貫プログラム開発方式」の言語系である事実を理解しましたか？	5 △	5 △	5
6. 離散・構造化モデルを使い対象世界を日本語文を使って OONJ 記述にする作業負担は軽かったですか？	5	5	4.2
7. 設計・実装段階のプログラミング作業は容易だったですか？	5	4.4	3.4 △
8. プログラムの信頼性の向上と検証の仕組み，トレーサの試行的利用	5 △	5 △	4.7 △

△印は質問のあったところや拡張規則の部分の説明不足な点について院生に補充説明を行った項目である．

述実験とアンケートから分かった．

6.2　院生の記述実験とアンケートの分析と考察

被験者の院生は 3 年生のときに OOJ の方法をすでに学んでいたので，OOJ トレーサを含めてここ 2 年間の発展分の補充説明を 90 分程度 2 回ほど行った後に記述実験とその後のアンケートを行った．ただ，初めて OOJ を学習し記述実験に加わった院生がいたので，改めて 90 分 4 回相当の講義と，4 回相当の演習をいずれもゼミ形式で行ったところ，他の 4 人の院生と同等あるいはそれ以上の理解と記述成果をあげた．被験者の対象世界はたとえば「アルコールの蒸溜実験」,「斜方投射と跳ね返り」,「赤血球の生成」,「水の大気循環」等の科学技術計算の世界であった．

院生たちの記述実験は対象世界の分析も設計・実装段階の記述も問題なくこなし，OOJ トレーサも積極的に併用してプログラム出力，即実行して，計算結果の物理的/化学的な考察も行った．それゆえにその過程の詳細分析をする必要がなくなった．その代わりに離散・構造化モデルや OOJ に関するインタビュを交えた詳細なアンケートを行った．その結果を**表 9** に示す．被験者院生の数は 5 人であった．アンケートは各主項目とそのサブ項目からなっており，成績評価風に 5 段階で答えてもらい，それらの平均値を出しさらに 5 人の値の平均値を出した．

院生たちの理解度ののばらつきは少なく各項目は 5.0 であることが圧倒的に多い．5.0 ではない場合の理由は特に汎化階層についての記述方法の説明が不足である，日本語文が自由すぎて逆に何をどう書くかが分かり難いということであった．これ等はより細かい説明を行うことで解決した．不足点としては第 7 項のエディタおよび第 1 項と第 3 項の構造化記述の慣れの問題であった．しかし記述する実力そのものは表 9 の「記述ができた」という項目に示すとおり十分に書かれていた．不足点はテキストの改訂とエディタの改良で容易に対処でき，現時点では解消している．

次に自由記述欄に多かったコメントは，魅力としては「自然言語に近い OONJ から始めて，それを順次ルールに従って変換すれば自然にプログラムに到達するというスムースな流れ」であり，これは「一貫したプログラム開発の方式が良かったのか？」と聞いたところ院生全員が「プログラム開発過程の流れが一セットの仕組みとして見えるのでこの方式の知識と経験は大きく役に立つ」という趣旨のことを答えた．

もう 1 点はトレーサについての感想で「変換前後の記述比較を直近眼近に見ながらやる経験は初めてであったけれど，直接比較できるので記述ミスがあるか否かを発見しやすくてとても有効な方法だと思う」と 3 人ほどが答えた．つまり一貫プログラム開発の方法，一貫相似性に基づく OOJ トレーサの 3 つの項目の有効性は彼らの 3 年生のときの経験も有効に働いて，相乗的にその実力向上に貢献したと考えられる．

以上の分析と考察から院生に関しては OOJ の設計で狙った主項目のテーマ，すなわち離散・構造化モデル，一貫プログラム開発の方法と OOJ トレーサに関する理解は十分行き届き，実記述力は十分に向上しており，以前の経験も有効に作用する結果，彼らの OOJ の実力は十分なレベルに達したと評価できる．ゆえに大学院生あるいはそれ以上のユーザであれば，OOJ を自身の仕事に対して実用的に十分使いこなせる段階に達したと結論した．

6.3　学部 3 年生のプログラム開発の授業の分析と考察

本節では前節よりも初級者に近い（表 1 の (B) 項）の適用下限ユーザに適用した結果を報告する．目的は OOJ の開発・整備に基づく利用可能ユーザの拡大と最低限の実力

表 10　学部生の最終レポート評価
Table 10　Marks of the final reports for the faculty students.

グループ名（人数）	対象記述	分析平均	設計実装	実行結果	全平均
G1（6 人）	5.00	5.00	5.00	4.67	4.79
G2（6 人）	3.67	4.6	3.67	2.83	3.69

の見極めである[*30]．

被験者の離散・構造化モデルと言語系 OOJ の理解度と彼ら自身の記述力向上の有効性を見るために最終レポートとアンケートからデータを抽出して分析した．結果は高評価順に 5 段階表示にして**表 10** に示す．評価は当該科目のシラバス上の習得目標に従い，表 3 の【II】の 3 つのサブ言語の設計仕様と OOJ エディタの指示に準拠して想定ユーザが分析，設計，実装の各段階で然るべき記述を行えたかどうかで判断する．評価項目は日本語での対象世界の記述，分析段階（OONJ）記述，設計段階（ODDJ）と実装段階（OPDJ）の記述，OOPL（Java）プログラムへの自動変換とその実行結果，そして最右欄に全平均とした．レポート部分である表 10 の評価値が 3.5 以上であれば一定レベル以上の理解と実力を持つと評価する．一定レベルとは科目の全体成績が A か A+となる可能性が高いことを指す．

離散・構造化モデルの理解度と利用する実力については，表 10 の両方のグループの「分析平均」が 4.6 を超えていること，この 2 つのグループの筆記試験（OOJ に関わる概念定義と用法を記述式で問う試験）の平均が 86.3 点であることから 12 人全員が十分な理解と記述の実力を得たと判断した．同じく OOJ を使って記述する実力については，筆記試験成績のほかに，G1 と G2 が設計・実装段階まで良好な成績（5.00，3.67）を示しているゆえに，一定の理解度と良好な記述力を得たと判断した．

次にプログラムへの変換と実行の項目については，2 つのグループの設計と実装の段階の評価値の差は 1.33 と，Java プログラムへの変換と実行の項目の差が 1.84 となっている．これは 2 つのグループ各個人のプログラミングの実力の差もさることながら，G1 の 6 人中 5 人までが OOJ トレーサを使っていたが，G2 では皆無であった．つまり OOJ トレーサの利用の差が大きな効果として出ていると考えられる．

表の右下部分のグレーに塗った部分は平均点が 3.5 を下回っている．そこで実行結果が 5.0 でなかったレポートを精査した結果，評価が下がった理由のほとんどは以下の 2 つのどちらかであった．

1. 設計と実装段階の細かい記述ミス．
2. 日本語文で書いた振舞いの抽象度が高すぎて目的に応じた（計算）式や mp，制御構造等に変換ができなかった．

項目 1 の細かい記述ミスとは (a) while 文の条件判定が誤っており無限ループに陥った，(b) データ型の指定ミス，(c) 離散単位の名前の指定ミス等であるが，これらは OOJ 特有のミスではなく，ヒューマンエラー，ケアレスミスであるといってよいミスで，1 記述例あたり数個以内であり十分に軽微なミスであった．

項目 2 の第 1 のミスはたとえば，振舞い文の式への変換，mp 文の関数呼び出し式への変換，条件判定文の式等が「式として」間違っているケースである．第 2 のミスは日本語文の再分析から必要なケースである．第 1 のミスの数は各記述例に対して 1 種類ずつ数個以内であり，第 2 のミスは 2 人が 3 カ所で起こした．しかしこれらのミスは記述例の長さ（数百行）に比べると十分に小さいと考えられる．G2 のレポートについて第 1 のミスの部分を著者らが 5 分〜20 分程度の改訂を行ったところ全員分について Java プログラムに達し起動ができた．

以上から OOJ の適用下限である表 1 (E) 項を満たせば被験者全員が十分有効に使えるようになったことが判明し，ユーザ層下限（表 1 の (B) 項）への拡大が実現した．この結果から OOJ の想定ユーザの条件には，1 つの PL 利用の実績（1 本で数百行）と離散・構造化モデルの理解が重要と判断できる．

本章の結論として，プログラム開発法が系統的に仕組まれてあるので想定ユーザ（表 1 (B)，(E) 項）でも確実にプログラムを開発できる方法が検証データとともに示された．

7. 関連した研究や言語との比較と考察

7.1 比較すべき関連研究や言語の選択

本章ではプログラム/ソフトウェアに関する以下の特徴を比較評価の基準とする．

［I］一貫した開発の仕組みか実績
［II］V&V あるいは信頼性に関わる仕組みか実績

この 2 点は 1 章で OOJ 開発の発端になった想定ユーザの要望を考慮するとともに，有効性を主張できる重要点であるからである．この基準に沿って選ばれたのが MATLAB，SysML，そして MDA である．これらを OOJ と比較対照を行って最終的には OOJ の特性を明らかにすることを狙う．これらと OOJ との基本特性の比較を**表 11** に示した．

7.2 MATLAB との比較と考察

シミュレーションプログラム開発環境としての MATLAB [12], [42] と比較する．MATLAB は行列計算を得意とした数値計算，可視化，プログラミングのための高水準の技術計算言語であり，Simulink [42]（シミュレーションとモデルベースデザイン環境）と組み合わせて，すでに用意されている微積分ブロックや論理演算ブロック等のブロッ

[*30] 学部 3 年生の受講者は 18 人であったが，アンケートを精査した結果，6 人が表 1 (E) 項をかなりの程度満たさないことが分かったので，被験者を 12 人に絞った．被験者のほぼ全員が前年度に C を学んでいた．離散・構造化や対象世界の分析はほぼ全員が初経験であった．

表 11　OOJ との比較対照のターゲット
Table 11　Comparison targets with OOJ.

ターゲット	一貫開発	V&V/信頼性	開発規模	ユーザ技量	開発人数	主適用分野	自然日本語（注）
OOJ	4 つの段階	個人保証	小規模限定	低	個人単独	科学技術計算	自由で制限なし
MATLAB	実績あり	実績による保証	小～大規模	中～高	個人～複数	科学技術計算	原則不使用
SysML	実績あり	実績による保証	大規模以上	高	複数・多数	組込みシステム	原則不使用
MDA	仕組みあり	実績による保証	大規模以上	高	複数・多数	業務システム	原則不使用

（注）：自然言語のままの日本語の利用

クダイアグラムを画面中に配置するだけでプログラムの自動生成技術を用いて円滑なプログラミング環境を提供しており，プログラムもしくは結果を即座に得ることが可能である [43].

MATLAB にはモデルをシミュレーションしテストするモデル検査の機能が組み込んであり，これにより誤りを少なくするする目的でプログラムの信頼性に対する検証や確認を行っており，実績等からも信頼性を保証していることが分かる．OOJ では分析記述の再現性に焦点を合わせたプログラムの信頼性を想定ユーザ個人で行うための手法と環境を提供している．この点は MATLAB と異なる．

また，MATLAB は DSL（Domain Specific Language）の 1 つなので提供ライブラリは特定分野[*31]に特化している．それゆえにプログラム構築が簡単にできて実行できる分野は限定される．サポートのない分野や対象世界については OOJ と同じスタートラインに立つことになる．

一方，表 11 からも見出せるように，OOJ は日本語を用いる等自由な記述を行うことが可能であり，簡潔なモデルを一貫して用いてプログラム開発を行っていく．このような手法を基盤とした開発を行うため要求される技術力も MATLAB と比較すると低くても運用可能であり，手法の習得やプログラム開発の経験を積むための環境が整っているといえる．

以上のように，ライブラリが整備されている分野ではプログラム開発を効率良く行えるという点では MATLAB が優位である．一方，プログラム開発の過程を簡単に学べ，オブジェクト指向プログラミングの理解が容易になるように構成されている教育効果の点と，一貫相似性の検証機能によるプログラムの信頼性の確保という点で想定ユーザにとっては OOJ の方が優位である．

7.3　SysML との比較と考察

SysML [13], [45] は組込みシステムを定義，分析，検証するための汎用的なダイアグラム形式のモデリング言語であり，UML の一部を選択し拡張を施した言語である．さらに Simulink や Modelica 言語を用いて/組み込んで，モデル駆動開発における仕様の記述 [46]，検証等に用いられており，モデルの動的検査手法 [47] 等の研究が行われており，開発環境も構築されている．また個々の適用分野に適したダイアグラムも定義されており，ドキュメンテーションの仕組みと実績に特徴がある．

適用分野としては航空機や自動車等の組込みシステム分野等に特化して用いられていることが特徴である [45],[13]．したがって UML と比べると適用分野（工業と科学技術）という点において OOJ との共通点が多い．一方 OOJ はクラス図やアクティビティ図等の UML の特徴の一部も保持しており [5]，UML を拡張設計された SysML のブロック定義図等との類似点もあるので，記述性の点で共通点も見出せるゆえに一部に OOJ との共通の仕様も持つといえる．

しかし SysML は企業が主な利用ユーザであり複雑な組込みシステムの開発を目的としている．したがってユーザには高度な技術力が求められ，開発を行う環境や条件が大きく異なる．また SysML はモデリング言語ではあるが，記述法という側面も強く開発手順を基本的には含んでいない．OOJ が言語系と開発手順をセットで一体的に持っているという点で SysMl と異なっている．

7.4　MDA との比較と考察

MDA [14] とは，ソフトウェア工学におけるモデル駆動開発（Model Driven Developments，以降 MDD と略記）の手法の 1 つであり，OMG で正式に採択されている．MDD では業務過程とプラットフォームを独立して記述し，最後に結合し Implementation Model を得ると，C/C ++，Java，Ada95 等のソースコードを自動生成によって得ることができる．さらに前節で示した SysML や MATLAB/Simulink を合わせて用いる方法やアジャイル MDA として Excutable UML [48] があり，そのツールとして BridgePoint がある．

具体的な流れとして，クラス図やコミュニケーション図等のいくつかのダイアグラムと Action Language を用いて業務過程を記述し，各プラットフォームごとのアーキテクチャを記述する．そしてそれらを統合し，最終的にはソースコードを自動生成する．以上の共通点は OONJ で対象世界を記述しそれを OPDJ を経て実行可能プログラムを得るという点に限れば開発環境として視点からみた開発の過程を抽象的には類似しているといえる．また，開発規模が大きく関わってくるが，モデルを中心とした開発手法であることにも類似点がある．

一方，MDA や MDD は企業で行われる製品開発や組込

[*31] ユーザ事例 [44] を見ると，信号処理，通信，制御システム，画像と動画の処理，計測等がほとんどを占めている．

みシステムの開発 [49] で使用されており，OOJ が対象としている個人向けの科学技術計算分野向けとは大きく異なる．さらに MDA の実施においては要求される知識と経験と技術が実質的にプロ並みに高いことが要求されており[*32]，こういうスキルを持つべくもない理工系の大学院生が個人で使うには無理であると評価できる．

また，磯田 [50] はモデリング法そのものについて，科学技術（計算）分野と製品開発とでは大きく異なっていることを指摘しており，OOJ と MDA は必要となるモデリング法が異なる．これらを考慮すると想定ユーザには MDA/MDD に必要な技術力は望めず，MDA の利用は難しいであろう．

以上から本章の結論として，OONJ が上記の 3 つの開発環境よりも，より小規模で個人向きであり，より学生・院生向きであると分かる．したがって想定ユーザにとっては OOJ が他より有効性の高い言語であると結論できる．さらに，OOJ の教育適用の経験をもとにすれば MATLAB や MDA の本格開発のフロントエンドや成功経験（ベスト・プラクティス）を用いた教育に使うことが可能であり，双方に有益な考えとして期待できよう．

8. 結論と今後の課題

結論

(1) 離散・構造化モデルは OOJ の根幹として一貫相似性を実現する機能を備えていることを確認した．

(2) 3 章と 4 章における OOJ の設計により 4.6 節で一貫相似性を実現する仕組みが具体的に示され，検証の方法も明らかにされた．

(3) 5 章で一貫相似性がプログラムの信頼性の向上に貢献することと OOJ トレーサによって信頼性を確保することを論証するとともに，OOJ が V&V の評価支援システムとして簡易な利用が可能なことを論証した．

(4) 6 章でプログラム開発法として科学技術計算の教育に適用して有用であることを確認した．プログラムの信頼性向上にも有効であることが確認できた．

(5) 7 章で想定ユーザ向けに限定すれば，比較した開発環境等よりも優れた点をいくつか持つことを論証できた．

(6) 以上から OOJ はプログラムの信頼性向上とプログラム開発に関わるより効果的な利用法を実現しているという 2 点において従来よりもより優れた特性を持つ言語系であることが結論できる．

今後の課題

実用規模（表 1 の (C) 項）の記述例の開発，ライブラリやトランスレータの機能強化，MATLAB 等のトランスレータの開発，OOJ の製品化検討がある．

[*32] 具体的には UML（各種開発環境を使いこなす能力）を十分に使いこなす能力，対象となるプラットフォームのアーキテクチャを記述できる能力，そしてソフトウェア開発の経験が必須であり，多人数による開発が前提である．

参考文献

[1] 畠山正行：オブジェクト指向分析モデリングの明示形式化・詳細化・手順化，シミュレーション学会誌，Vol.21, No.4, pp.295–309 (Dec. 2003).

[2] 渡邉一衛：ものづくり・サービスづくりのシミュレーション，日本シミュレーション学会論文誌，Vol.28, No.1, pp.41–45 (2009).

[3] 加藤　廣：自動車開発のデジタル化はシミュレーションが主役，日本シミュレーション学会論文誌，Vol.27, No.2, pp.50–53 (2008).

[4] 日本計算工学会（編）：工学シミュレーションの標準手順，JSCESS-HQC002:2011，日本計算工学会 (May 2011).

[5] 池田陽祐，三塚恵嗣，加藤木和夫，大木幹生，上田賀一，畠山正行：UML との比較に基づくオブジェクト指向分析設計記述言語 OONJ の評価，情報処理学会論文誌：数理モデル化と応用，Vol.5, No.3, pp.63–78 (2012).

[6] 池田陽祐, 三塚恵嗣, 上田賀一, 畠山正行：UML との比較評価に基づくオブジェクト指向分析設計記述言語 OONJ の記述技法の特徴，情報処理学会論文誌：数理モデル化と応用，Vol.6, No.1, pp.1–16 (2013).

[7] 池田陽祐，大木幹生，三塚恵嗣，畠山正行：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 SE 研究会報告，2009-SE-163, pp.57–64 (2009).

[8] 畠山正行：高度なシミュレーションのためのオブジェクトベース一貫モデリング過程論とその駆動支援環境，第 1 回情報処理学会 MPS 研究会研究報告 (May 1995).

[9] 畠山正行：一貫モデリング過程論に基づく物理世界のオブジェクトモデル，情報処理学会第 20 回 MPS 研究会研究報告，pp.7–12 (July 1998).

[10] 電子情報通信学会，入手先 ⟨http://www.ieice-hbkb.org/files/01/01gun_12hen_06.pdf⟩（参照 2013-03-21）.

[11] ISO/IEC 25000, Software engineering, Software product Quality Requirements and Evaluation (SQuaRE), Guide to SQuaRE (2005-08-01).

[12] MathWorks MATLAB (online), available from ⟨http://www.mathworks.co.jp/products/matlab/⟩ (accessed 2013-03-21).

[13] サンフォードフリーデンター，アランムーア，リックスタイナー（著），西村秀和（訳）：システムズモデリング言語 SysML，東京電機大学出版局 (May 2012).

[14] OMG, MDA Specifications, available from ⟨http://www.omg.org/mda/specs.htm⟩ (accessed 2013-03-21).

[15] 数値流体力学編集委員会（編）：数値流体力学シリーズ 2，圧縮性流体解析，東京大学出版会 (1995).

[16] 畠山正行，池田陽祐，三塚恵嗣：ドメインユーザ・プログラミング：記述言語をベースとする仕組みの提案，第 125 回 HPC 研究会報告，2010-HPC-125 (2010.06.17).

[17] 峯村吉泰：流体・熱流動の数値シミュレーション，森北出版株式会社 (2001).

[18] 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 (2004).

[19] 青木　淳：オブジェクト指向システム分析設計入門，第 2 章：広義のオブジェクト指向，（株）SRC (1993).

[20] メイヤー，B.（著），酒匂　寛（訳）：オブジェクト指向入門—原則・コンセプト，第 2 版，（株）翔泳社 (Jan. 2007).

[21] 畠山正行，池田陽祐，三塚恵嗣，加藤木和夫：メッセージパッシングモデルに基づく差分方程式の計算方式とその実行例，第 86 回 MPS 研究会，2011-MPS-86 (Oct. 2011).

[22] 佐藤一郎，所真理雄：プロセス代数によるリアルタイムオブジェクト指向プログラミング言語の意味論，情報処

理学会論文誌，Vol.35, No.11, pp.2355–2365 (1994).
[23] 磯部祥尚，佐藤　豊，大蒔和仁：準弱双模倣性をもとにした仕様の段階的合成方法，電子情報通信学会技術研究報告，COMP，コンピュテーション，Vol.97, No.627, pp.33–40 (1998-03-23).
[24] 池田陽祐，三塚恵嗣，上田賀一，畠山正行：実世界を直截に記述可能な OOJ 記述環境の開発とその利用効果，第 92 回 MPS 研究会報告，2012-MPS-92-E (Feb. 2013).
[25] 大木幹生，池田陽祐，三塚恵嗣，畠山正行：記述言語系 OOJ 上で実現する個人向け V&V 環境の提案，第 92 回 MPS 研究会，2012-MPS-92-F (Feb. 2013).
[26] 三塚恵嗣：オブジェクト指向一貫記述言語系 OOJ における変換機構と実装記述言語 OPDJ の開発，平成 22 年度茨城大学大学院修士論文 [28] (Feb. 2011).
[27] Friedentahl, S. et al.: A Practical Guide to SysML—The Systems Modeling Language. 西村秀和（監訳）：システムズモデリング言語 SysML，p.74(§ 4.3)，東京電機大学出版局 (2012.05.10).
[28] 畠山研究室，入手先 ⟨http://gaea.cis.ibaraki.ac.jp/⟩（参照 2013-03-21）.
[29] Tiller, M.M.：Modelica による物理モデリング入門，オーム社 (Nov. 2003).
[30] 吉田紀彦：自動プログラミングハンドブック第 4 章「プログラム変換手法による自動プログラミング」，pp.109–120, オーム社 (1989).
[31] Burstall, R.M. and Darlington, J.: A Transformation System for Developing Recursive Programs, *J. ACM*, Vol.24, No.1, pp.44–67 (1977).
[32] 永松泰成，畠山正行：オブジェクト指向日本語一貫記述環境 OOJDE の開発，情報処理学会研究報告，01-SE-136, pp.25–32 (2002).
[33] 沼崎隼一：オブジェクト指向一貫記述言語系 OOJ における一貫相似性の検証，平成 22 年度茨城大学大学院修士論文 (Feb. 2011).
[34] 沼崎隼一，池田陽祐，三塚恵嗣，畠山正行：オブジェクト指向一貫記述言語系 OOJ におけるトレーサの開発，第 171 回 SE 研究会報告，2011-SE-171-6 (Mar. 2011).
[35] 小倉　崇，藁谷大輔，櫻井孝平，古宮誠一：オブジェクト指向プログラムのためのデバッグ/テスト支援システム：プログラムトレーサー機能について，信学技報，KBSE 知能ソフトウェア工学，Vol.104, No.588, pp.7–12 (2005-01-18).
[36] IEEE Std. 830-1998, IEEE Recommended Practice for Software Requirements Specification.
[37] ASME: Guide for Verification and Validation in Computational Solid Mechanics, V&V10-2006 (2006).
[38] ASME: Standard for Verification and Validation in Computational Fluid Dynamics and Heat Transfer, V&V20-2009 (2009).
[39] What is Verification and Validation, 2009-July-24, available from ⟨http://www.nafems.org/publications/brows_buy/qa/vandv/⟩ (accessed 2013-03-21).
[40] 日本計算工学会（編）：工学シミュレーションの品質マネジメント，JSCESS-HQC001:2011，日本計算工学会 (May 2011).
[41] 池田陽祐，岡田信一郎，上田賀一，畠山正行：離散構造化モデルを用いた一貫したプログラム開発実践教育への OOJ の効果的な適用，情報処理学会第 90 回 MPS 研究会研究報告，No.18 (2012/09/20).
[42] MathWorks Simulink, available from ⟨http://www.mathworks.co.jp/products/simulink/description1.html⟩ (accessed 2013-03-21).
[43] MathWorksinc (online), available from ⟨http://www.mathworks.co.jp/verification-validation/index.html⟩, (accessed 2013-03-21).
[44] MathWorks，ユーザ事例—ユーザーインタビュー (online)，入手先 ⟨http://www.mathworks.co.jp/company/user_stories/osaka-kyoiku.html⟩（参照 2013-03-21）.
[45] Object Management Group: formal-12-06-01, ver1.3, Object Management Group (online), available from ⟨http://www.omg.org/spec/SysML/1.3/⟩ (accessed 2013-03-21).
[46] 加藤秀明，上田賀一：組込みソフトウェア開発における SysML 適用事例，信学技法，IEICE Technical Report, SS2009-34 (2009-10), pp.31–36 (2009).
[47] 小野康一，中村宏明，石川　宏：時間/機能制約による仕様に対する実行可能な UML/SysML モデルの動的検査手法，コンピュータソフトウェア，Vol.27, No.2, pp.33–49, 日本ソフトウェア科学会 (2010).
[48] OMG (Obejct Management Group): Building and Implementing Concurrent Specifications, available from ⟨http://www.omg.org/news/meetings/workshops/RT_2005/00-T2_Starrett-Mellor.pdf⟩ (accessed 2013-03-21).
[49] Vincent, A. and Brain, A.: available from ⟨http://anu.academia.edu/ZoeBrain/Papers/133897/Simulation_Case_Study_-_xtUML_in_agile_development⟩ (accessed 2013-03-21).
[50] 磯田定宏：実世界モデリング有害論—オブジェクト指向モデリング技法の解明，電子情報通信学会論文誌，Vol.J83-D-I, No.9, pp.946–959 (2000).

付　　録

本論文の一貫相似性と双模倣性との関係

OONJ，ODDJ，OPDJ，OOPL で書かれた 4 つの状態遷移系の OONJ，ODDJ，OPDJ の各記述および OOPL プログラムが双模倣性を持つということは，4 つの言語で書かれた離散単位の振舞い（動き，プロセス）の等価性，すなわち離散単位の動的な側面の相似性を意味することになる．したがって 4 つの記述とプログラムが発生させる振舞いや動き（プロセス）の双模倣性，すなわちプロセスの等価性を検証することで動的な一貫相似性の検証を実現できる．

(A) 4 つの段階の離散単位間の振舞いの入力と出力が等しいかどうかを評価する．離散単位が外部から観測不能なブラックボックスの場合，あるいは簡易検証の場合にあたる．
(B) 3 つの段階の離散単位間の振舞いの途中の状態まで含めてすべてが等しいかどうかを評価する．

まずは第 1 段階として (A) の弱双模倣性について，それが満たされれば，(B) の双模倣性について調べる手順になる．

表 12　OONJ 記述規則
Table 12　OONJ descriptive rules.

離散単位	（規則 2 の【　】と，4 の《　》はそれぞれが容器であることを表す幾何学的な枠線を指す）
1 ＜**対象世界**＞	:=:＜対象世界共通要素フレーム群＞ (＜ lf ＞＜フレーム＞)$^{+}$
2 ＜**フレーム**＞	:=:＜フレームヘッダ＞【(＜ lf ＞＜フレームヘッダスロット＞)$^{+}$(＜ lf ＞＜スロット＞)$^{+}$】
3 ＜フレームヘッダ＞	:=:＜フレーム要素特定子＞＜ sp ＞＜フレーム名＞＜相互関係＞$^{?}$
4 ＜**スロット**＞	:=:《＜スロット総称文＞ (＜ lf ＞＜サブスロット＞)$^{+}$》
5 ＜スロット総称文＞	:=:＜スロット要素特定子＞＜ sp ＞＜日本語文＞＜相互関係＞$^{?}$
6 ＜**サブスロット**＞	:=:＜サブスロット主文＞ (＜ lf ＞＜付置属性文＞)$^{?}$
7 ＜サブスロット主文＞	:=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞＜日本語文＞$^{?}$ ＜相互関係＞$^{?}$
8 ＜付置属性文＞	:=:＜サブスロット要素特定子＞＜ sp ＞＜階層構造記述子＞ (“(” ＜属性名＞ (“,” ＜属性名＞)* “)”)
構造記述子	（相互関係を付与することが構造を表現する方法であることの仕組みや符号を指す）
9 ＜**相互関係**＞	:=:＜ rt ＞＜相互関係記号＞＜相互関係相手名リスト＞
10 ＜相互関係記号＞	:=:((“≫” ＜相互関係名＞ “≫”) ｜ (“≪” ＜相互関係名＞ “≪”) ｜ (“≪” ＜相互関係名＞ “≫”))
11 ＜相互関係相手先リスト＞	:=:＜相互関係相手名＞ (“,” ＜相互関係相手名＞)*
12 ＜相互関係名＞	:=:(≪ mp ≫｜≪集約≫｜≪汎化≫｜≪特化≫｜≪実値化≫｜≪引用≫｜＜その他の相互関係名＞)
13 ＜相互関係相手先＞	:=:(＜フレーム番号＞ “：” ＜フレーム名＞)$^{?}$(“[” ＜スロット番号＞ (“-” ＜サブスロット番号＞)$^{?}$ “]”)$^{?}$
14 ＜階層構造記述子＞	:=:((＜ mr ＞＜ sp ＞ “｜”) ｜ (＜ mr ＞ “｜｜”))
日本語文	（自然言語としての日本語の単語や文の種類を推奨するあくまで原則的な規則）
15 ＜日本語文＞	:=:(≪自然言語としての日本語の文法のままで定義・記述する文≫｜＜拡張日本語文＞｜＜注釈文＞)
16 ＜拡張日本語文＞	:=:≪式 (数式，計算式，化学式等) ≫｜＜和文単語＞｜≪日本語文や式等に変換可能な表現≫
17 ＜和文単語＞	:=:(＜属性名＞｜＜フレーム名＞｜＜相互関係名＞｜＜相互関係相手先＞｜＜ファセット記号＞)
18 ＜属性名＞	:=:≪多くの場合は名詞を使うが，制約条件等を書く場合には日本語文でも良い．≫
19 ＜フレーム名＞	:=:≪多くの場合は名詞を使うが，振舞いのフレーム等の場合には日本語文でも良い．≫
20 ＜注釈文＞	:=:＜サブスロット要素特定子＞＜ sp ＞≪任意の日本語文や和文単語を書いて良い．≫
21 ＜ファセット記号＞	:=:“fn” ＜ファセット番号＞　（fn は facet number の略．要素種類の分類記号）
22 ＜ファセット番号＞	:=:≪ OONJ の離散単位の種類表に記載されている種類に対応する番号 (**f**acet **n**umber) ≫

特別なフレームやスロット	（2 つは規則というより右辺が左辺の標準スタイルで提供する具体的な種類を示す．詳細省略）
23 ＜対象世界共通要素フレーム群＞	：＜初期/境界条件フレーム＞，＜共通属性フレーム＞，＜シナリオフレーム＞，・・・
24 ＜フレームヘッダスロット＞	：＜フレーム内共有属性スロット＞，＜相互関連スロット＞，＜ mp コントローラスロット＞
要素特定子	（相互関係を付与するための要素を特定する仕組みや記号を指す）
25 ＜フレーム要素特定子＞	:=:＜フレーム番号＞＜ sp ＞ $^{+2}$ ＜ファセット記号＞
26 ＜スロット要素特定子＞	:=:＜スロット番号＞＜ sp ＞ $^{+2}$ ＜ファセット記号＞
27 ＜サブスロット要素特定子＞	:=:＜ sp ＞ “-” ＜サブスロット番号＞＜ sp ＞＜ファセット記号＞
28 ＜フレーム番号＞	:=:≪各フレーム単位の一連番号で，通常は 1 から始まる昇順の整数≫
29 ＜スロット番号＞	:=:≪各スロット単位の一連番号で，通常は 1 から始まる昇順の整数≫
30 ＜サブスロット番号＞	:=:≪各サブスロット単位の一連番号で，通常は 1 から始まる昇順の整数≫
記述位置指定子	（記述位置を指定する仕組みや記号を指す）
31 ＜ rt ＞	:=:≪その右側の要素を右詰め (右寄せ) して記述する．≫
32 ＜ lf ＞	:=:≪次の離散単位に移り，その左先頭に戻る．フレーム間の場合は任意の間隔を空ける．≫
33 ＜ sp ＞	:=:≪全角一文字空白，space ≫
34 ＜ mr ＞	:=:≪直上のサブスロットに「階層構造記述子」または NJ 単文の先頭があればそこまで右寄せ．≫

拡張 BNF 記号
(1) ＜・・・＞はケット (アングルドブラケット) で，要素の内部を展開すべき/展開可能な離散要素であることを表す．
(2) ≪・・・≫はギュメ (ギルメット) で，内部要素の展開は完了した離散要素であることを表す．
(3) ＜・・・＞^{n} は n 個 (n ≧ 0，n=1 は省略可) の要素並びを表す．　(4) ＜・・・＞$^{+n}$ は n 個以上の要素の並びを表す．
(5) ＜・・・＞* は 0 個以上の要素並びで，＜・・・＞$^{+0}$ と同等．　(6) ＜・・・＞$^{?}$ は 0 個/1 個 (無し/有り) の要素を表す．
(7) “｜” はこの記号の左右にある離散単位の片方の選択を指示する．OONJ では同一記号で集約や付置属性も表す．
(8) ダブル・クオテーション “” は挟まれる文字列や記号等をそのまま記すことを指示する．

畠山 正行　（正会員）

1947 年生．1978 年東京大学大学院航空学専攻博士課程修了．工学博士．東京都立航空工業高等専門学校を経て，1990 年 10 月より，茨城大学工学部情報工学科専任講師，同助教授，後に准教授．2013 年 4 月特命研究員，現在に至る．超音速凝縮流れ等の非連続気体流れの力学の研究に従事．次いで数値シミュレーション用の計算機環境，オブジェクト指向日本語記述言語系，パーソナルソフトウェア開発環境の研究・開発に従事．日本機械学会会員．

池田 陽祐　（正会員）

1984 年生．2008 年茨城大学工学部情報工学科卒業．2010 年同大学大学院情報工学専攻博士前期課程修了．2013 年同大学院情報・システム科学専攻博士後期課程修了．2013 年同大学工学部博士特別研究員，現在に至る．パーソナルソフトウェア工学における開発環境の研究・開発に従事．IEEE 会員．

三塚 恵嗣

1986 年生．2009 年茨城大学工学部情報工学科卒業．2011 年同大学大学院情報工学専攻修士課程修了．同年日立情報システムズ（現，日立システムズ）入社．現在に至る．製造業の物流管理システムの開発・保守に従事．

加藤木 和夫　（正会員）

1947 年生．1970 年北海道大学理学部物理学科卒業．1971 年日立エンジニアリング入社．制御用プログラミング言語，ソフトウェア開発環境の研究・開発に従事．技術士（情報工学）．2002 年加藤木技術士事務所．2006 年茨城県立産業技術短期大学校，現在に至る．

大木 幹生

1982 年生．2008 年茨城大学工学部情報工学科卒業．2010 年同大学大学院情報工学専攻修士課程修了．同年群馬工業高等専門学校教育研究支援センター．パーソナルソフトウェア工学における品質評価の研究・開発に従事．

上田 賀一　（正会員）

1961 年生．1989 年名古屋工業大学大学院工学研究科電気情報工学専攻博士後期課程修了．工学博士．同年同大学工学部電気情報工学科助手．1990 年茨城大学工学部情報工学科講師．2002 年同大学助教授，後に准教授．2012 年同大学教授，現在に至る．ソフトウェア工学，組込みソフトウェアに関する研究に従事．電子情報通信学会，ACM，IEEE Computer Society 各会員．