オブジェクト指向日本語一貫プログラミング環境

加藤木 和夫† 畠山 正行††

オブジェクト指向に基づいた一貫した日本語系記述を用いたプログラミング方式およびこれを基礎にした個人規模用のプログラミング環境を，コンピュータ以外を専門とするドメインユーザをターゲットとして設計・開発した．本プログラミング環境は，モデリングとプログラミングの各段階でオブジェクト指向的に記述する相似形の3種類の日本語記述言語系（総称してOODJ）を用意して，これで記述する．この記述は最終的には本環境のトランスレータによりC++へ変換される．ドメインユーザは要求記述からプログラム開発までを一貫した相似表現の下にシームレスな記述を行うことができ，C++のようなプログラミング言語を直接使う必要がなくなる．また，本環境では具体的な作業手順を示すNステップ開発手順とそのGUIガイダンスシステム，OODJ記述を支援する一貫支援エディタ，および統合的な開発・再利用のためのリポジトリ環境も設計・構築した．本環境を用いてスカッシュゲーム，植物の成長シミュレーション，画像処理システムを記述した結果，本環境はシームレスな一貫プログラミング環境として十分機能していることが実証された．OODJの記述力不足が指摘されたが，これは特定分野向けの日本語ライブラリや辞書コンテンツをリポジトリに蓄積することで十分補える．結論として，本プログラミング環境は当初の目的を一応実現し，実用化・製品化の見通しも得た．

Object-oriented, Integrally Consistent Japanese Programming Environment

KAZUO KATOUGI† and MASAYUKI HATAKEYAMA††

We have designed and developed a programming environment that the domain users in various fields can develop their requirement programs by using three kinds of the similar Japanese description languages throughout the development processes. These languages are generically called the OODJ. The described Japanese program is transformed into the object-oriented program of the C++ by the specified translator. Therefore, the user can develop their computing programs without directly using some programming languages like C++. To support this programming environment, the following three subsystems have also been developed and implemented; the N-steps modeling and developing procedures and its actual guidance system with GUI environment, some integrated hierarchical editors, and an object-oriented repository. This programming environment have been applied to the fields of the squash game, the plant growth simulation and the image processing system. Throughout the long test uses and evaluations, the validity and the usefulness has been confirmed by the domain users in each field. The domain users have been able to develop their computing system easily and quickly by using only the OODJ. Though, they have pointed out that the functions of the OODJ are not yet complete and also the OODJ dictionary in the repository is insufficient, these points are to be made up. As the conclusion, the purpose of the integrally consistent OODJ programming environment has been realized.

1. はじめに

プログラミング環境を使うのは通常はソフトウェア開発者であるが，ドメインユーザ☆もプログラミング環境に対して強い利用希望を持っている．ドメインユーザの開発するソフトウェアは中小規模ではあるが，自

† 日立プロセスコンピュータエンジニアリング株式会社
Hitachi Process Computer Engineering, Inc.
†† 茨城大学工学部情報工学科
Department of Computer and Information Sciences, Faculty of Engineering, Ibaraki University

☆ 本論文でいうドメインユーザとは，ソフトウェア開発やプログラミングの専門家ではなく，自身の専門分野を持ち，自家用プログラムを自身の仕事のために必要最低限開発するユーザ．FORTRANユーザが最も多く，新規な言語修得には消極的で，一連のソフトウェア開発プロセスやプログラミング技法等を意識しては適用しない．分野としては，たとえば流体や構造の解析シミュレーション，画像分析，建築設計等であり，内容的には特殊で試行錯誤の多い数百行から数千行のプログラムを組むことが多い．

身専用（自家用）のプログラムに対して，その仕様の決定や設計・実装まで開発過程のすべてを，自身1人，あるいはせいぜい数人でやるという特徴がある．このようなドメインユーザのプログラミング環境へのニーズや要求は，言語やシステムに通暁していないだけに余計，ソフトウェア開発を専門とする者以上に強く多様である．たとえば，プログラムにドメイン側の知識やアルゴリズムについて完全に記述することはいとわないが，記述言語は自身のドメインに近い自然言語と用語であって欲しい，プログラミングフリーを望む，といった強い本音要求がある．しかし，このようなドメインユーザの要求に応えるプログラミング環境は，ビジネスドメインのユーザに対するものを除けば，ないに等しい．これに応えようというのが，本研究の動機である．

本論文では，ドメインユーザはオブジェクト指向の概念[1),2)]およびその体系は理解しており，自然言語での対象世界のオブジェクト指向記述は可能と想定している．また，対象とするソフトウェアの規模は本環境を利用するドメインユーザの特徴から考え，当面，個人規模[3)]の開発向けとした．このようなドメインユーザが自分でソフトウェアを開発するには現状の開発環境では次の2つの点が不足していると考える．1つは，対象世界の概念モデルの作成からプログラム実行までのモデル化過程を一貫して概念モデルと相似的な形で支援する環境が整備されていないという点，もう1つはそのような過程において記述する表現言語系に一貫性がないという点である．

そこで本研究においては，上記の2つの問題点のうち特に表現言語系の面から一貫した開発（プログラミング）環境を構築することを狙いとする．本環境の狙いを実現する具体的な方法とは，最初の概念モデルの記述言語を自然言語とし，そのモデル記述表現をプログラム記述へと段階的に一貫した相似変換を実現することである．そこでこのモデル記述からプログラム記述への段階的相似変換を実現するために，モデルとプログラムモジュールが密接な関係にあるオブジェクト指向パラダイム[1),2)]を用いることとした．そこで，開発フェーズの各段階の表現言語系としては，まず要求記述に対しては自然言語を用い，分析・設計には新たにオブジェクト指向日本語を定義し，設計・実装に対してはオブジェクト指向表記のできる日本語プログラム記述言語を開発する．

以下では最初に2章で筆者らが採用したアプローチ法を述べ，3章では本モデル化の上流の記述言語について，4章では本環境の中核となる日本語プログラミングの構文と特徴について述べる．5章では記述を支援するエディタとリポジトリ，6章では本環境の実装とソフトウェア構成，そして7章では適用事例について記述する．最後に8章と9章で本環境の考察，評価，結論および今後の課題について述べる．

2. 日本語一貫プログラミングへのアプローチ法

最初のモデル記述からプログラム記述までを相似な日本語表現を用いて連続的に変換していく方法を日本語一貫プログラミングと呼ぶこととする．

2.1 日本語一貫プログラミングとNステップ開発手順

対象世界の「モノ」をモデル化の単位とし，ソフトウェア世界のオブジェクトへ写像する技術はオブジェクト指向が本来持っている特長である．我々はこのオブジェクト指向技術を用いて日本語一貫プログラミングを具体的に実現する手順として，段階的詳細化の方法である**図1**に示すNステップ開発手順[4)]を導入する．

本手順の導入は抽象的な日本語一貫プログラミング過程をNステップ開発手順として明確に具体的な手順の形で定義することにより，プログラミング技術にさほど詳しくないドメインユーザに，対象分野（ドメイン）の設計およびプログラムを日本語相似表現により書き換えて詳細化する方法論を提供することを狙うものである．実装環境としては階層化されたGUI画面群が実際のガイド役としてユーザをガイドするように構成する（5章参照）．

ここでは，図1に沿ってその手順を示す．概念モデ

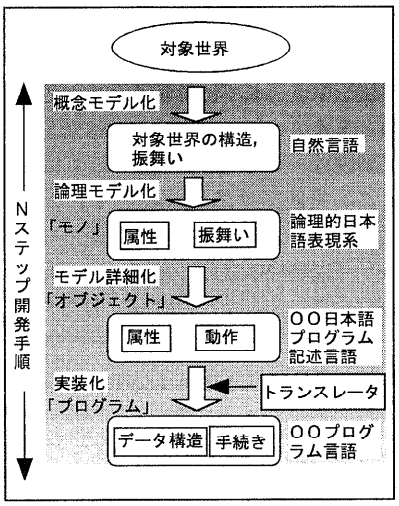


図1 Nステップ開発手順
Fig. 1 N-steps development procedures.

ル化（要求記述段階）ではドメインユーザが対象世界の構造と振舞い等を自然言語で記述する．論理モデル化（分析・基本設計段階）では概念モデルを「モノ」単位で分析し[5]，その属性，構造および振舞いの概要をオブジェクト指向の論理的日本語表現系を用いて記述する．モデル詳細化（詳細設計段階）では論理モデルの「モノ」をオブジェクトとし，その属性，構造および動作（インタフェース，アルゴリズム）をオブジェクト指向日本語プログラム記述言語を用いて詳細に記述する．実装（プログラミング段階）ではプログラミングフリーというドメインユーザの要求を満たすべく，トランスレータによりC++プログラムへと，自動変換する．

2.2 日本語一貫記述の言語系

日本語一貫プログラミングの表現言語体系の設計方針として，次を立案した．

(1) モデル化からプログラミングまで，オブジェクト指向の概念で一貫して構成された日本語風記述法を全過程で採用する．

(2) 日本語記述の最後の段階では制約された構文を用いた日本語風プログラムは既存のオブジェクト指向プログラム言語表現へトランスレートされる．

このような戦略を実現する表現言語系として，以下のような一貫して相似な系列の記述言語系を考案した．

(a) 要求記述段階の言語としては**自然言語**（書法に緩やかな制限を設けた日本語）を用いる．

(b) 分析・基本設計段階の言語としてはオブジェクト指向の論理モデルを記述するための「オブジェクト指向日本語」（**OOJ**：Object Oriented Japanese）を設計，開発する．OOJは複雑な表現も許すため，記述・解釈・変換は自然言語処理の手法を取り入れず，ドメインユーザ自身が行うものとする．

(c) 詳細設計・実装段階の言語としては既存のオブジェクト指向プログラム言語へトランスレート可能な「オブジェクト指向日本語プログラム記述言語」（**日本語SMDL**：Software Modeling Description Language）を設計，開発する．

なお，(a)，(b)，(c) を総称してオブジェクト指向記述日本語（**OODJ**：Object Oriented Description Japanese）と呼ぶこととする．

2.3 オブジェクト指向日本語一貫プログラミング環境

上記の表現言語系を支援するために次の開発環境を開発する．

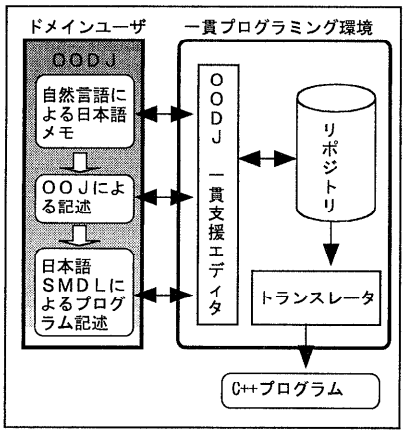


図2 オブジェクト指向日本語一貫プログラミング環境の概念図
Fig. 2 Conceptual constitution of Object Oriented Japanese programming environment.

(1) トランスレータ

日本語SMDL記述を既存のC++プログラム処理系へ渡すツール．

(2) 記述編集支援環境

OODJを一貫してガイド・記述編集支援するエディタ群．蓄積した単語集を自在に組み合わせるGUIガイダンスシステム．

(3) 限定日本語の蓄積と引用機構

モデル記述やプログラムおよび日本語の単語（クラス名，メッセージ名等）を蓄積するリポジトリ[3),7)]．

以上の考察に基づいた本環境の完成図を**図2**に示す．図2は図1を記述言語名とそのプログラミング環境で置き換えたものであり，論文タイトルを具現化した環境そのものである．

本論文の手法は，各ドメインユーザが持つ各々のドメイン（専門分野）に対する通常の日本語を用いての強くかつ高精度な記述力をプログラム生成に最も正確に生かした形で反映させようとする意図から企図された方式である．

3. 自然言語記述とオブジェクト指向日本語

3.1 自然言語を用いた日本語メモ

「日本語メモ」はオブジェクト指向言語Eiffelの「ショッピングメモ」[6)]に相当するもので，単なる備忘録ではなく自由記述形式の要求記述である．記述は対象世界の分析モデリング，特に概念モデリング段階の記述に等しく，要求記述の立場から見れば対象世界のモデル化し実現すべき範囲の（自己）要求の全貌の記述に等しい．

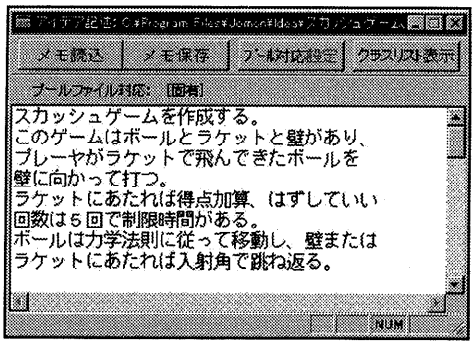


図3 自然言語による記述例
Fig. 3 Described example using natural language.

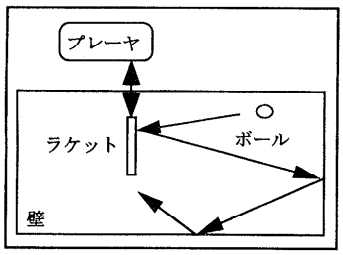


図4 スカッシュゲームのラフスケッチ
Fig. 4 A rough sketch of squash game.

(1) ドメインユーザのモデル化の目的に応じて，対象世界から抽出すべき対象世界の主体（モノ）となるべき要素を決定し，その主体名をすべて記述して取り出す．
(2) 各主体（モノ）をモデル化単位として，各単位ごとの属性や振舞いを記述する．
(3) モノとモノとの間の構造や相互作用を記述する．
(4) 記述の定量化が難しく実装不可能な形容詞（たとえば,「美しい」）等の表現を避ける．
(5) 記述は最終的には，単文の集まり（連接）で構成するように変換する．

原則は以上のとおりであり，ごく常識的なオブジェクト指向のモデリング記述のルールにすぎない．これ以上に関しては，結果として論理モデル化され最終的に実装プログラム化されるモデルとの整合性・相似性を保てるような構成になるまで経験的・反復的に改訂を繰り返す必要がある，と現状ではいうべきである．現在では経験的な書法がいまだ大半を占めている．

図3 にスカッシュゲームの対象記述例を示す．スカッシュゲームはシンプルで全体の見通しが良いので具体例として本論文で通して取り上げる．文中の名詞を「モノ」，動詞を「相互作用」とするが，本環境ではユーザがこれらを識別する．なお，抽出にあたっては図形を用いて「モノ」と「相互作用」のラフスケッチ（**図4** の例）を描くことがモノ（=オブジェクト）抽出の助けとなる．図形表現は分析する思考上重要な手段であるが，あくまでも本開発プロセスでは補助手段と位置づける．OMT 表記等のダイアグラム（**図5** 参照）は日本語メモの記述の補助とする．なお，設計（実装）段階で初めて出現するテーブル等の実装に必要となるオブジェクトも気付いた時点で列挙する．クラスの継承はこの段階では考えない．クラス階層は次の段階への補助手段となる．

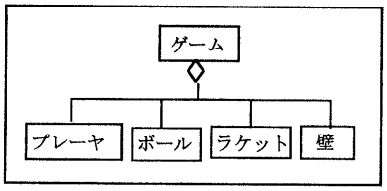


図5 オブジェクト図
Fig. 5 Object diagram by OMT.

3.2 オブジェクト指向日本語（OOJ）の仕様

本言語は論理モデルをオブジェクト指向表現するために日本語にオブジェクト指向の概念記述構成およびその表現力を強化したもので，以下のような特徴を持つ．

(1) クラス定義記述のためのキーワードを持つ

「クラス定義」，および「属性」と「動作」である．ユーザが入力したデータはマウスにより取り込み，リポジトリへ格納する．この方法により記述の自由性と任意のデータの取り込みを両立させた．すなわち，

A. 円**クラス**の**属性**は位置と半径である．

B. **クラス** 円

属性 位置，半径

A と B のどちらの表現も許す．図5のボールクラス記述の例を**図6** に示す．本例の OOJ を用いたクラス記述は図5の OMT 表記のクラス図を文章表現したものに相当する．

(2) アルゴリズム記述用オブジェクト指向記号を持つ

(a) インスタンス生成記号（=>）

クラス名 => インスタンス名

（例） ボール => スカッシュボール

(b) メッセージ通信記号（←）

オブジェクト ← 振舞い

（例） スカッシュボール ← 移動する

(3) プログラムの制御構造用英字キーワード

連接：□

選択：**IF～THEN～ELSE～ENDIF.**

繰返し：**LOOP**（判定条件）{ ～ }.

ただし，これらは構文チェックは行わない．

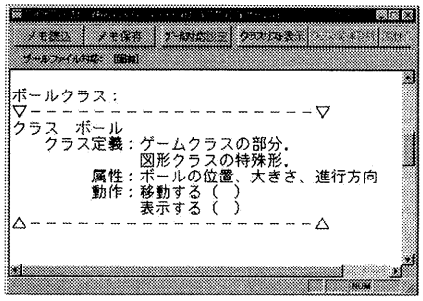


図6 OOJによるクラス記述例
Fig. 6 Class description example by OOJ.

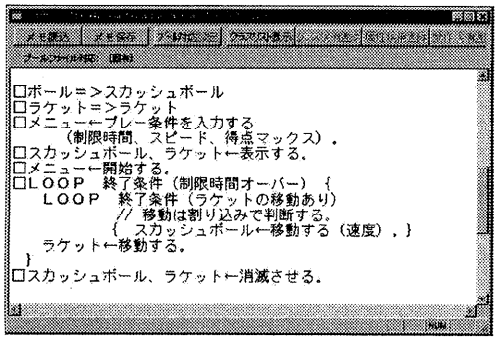


図7 ゲームクラスの動作（OOJ）
Fig. 7 Action descriptions of class "game".

図4にあるゲームクラスの動作のOOJ記述例を**図7**に示す．この記述例はドメインユーザにも容易に記述できるものであり，かつ同時にゲームクラスのオブジェクトの動作記述そのままであることも自明である．これがOOJによる記述の特長である．

4. 日本語SMDL

4.1 日本語SMDLの定義

オブジェクト指向日本語プログラム記述言語は，以前に開発した解析シミュレーション用の言語SMDL[4]を基に，通常のプログラミング言語に近い日本語SMDL[8]を定義・開発した．日本語SMDLはOOJで記述されたクラスの概要，動作のアルゴリズムをさらに詳細に記述する，ちょうど詳細設計段階から実装段階の記述言語である（詳細な構文は文献8)を参照)．日本語SMDLの記述単位はクラスである．クラス記述の構成はクラス定義，属性，動作の3セクションからなる．

4.2 クラス仕様

4.2.1 クラス定義セクション

(1) クラス名

クラス名は日本語で記述する．対応する英字名も記述すれば，C++へ変換されたときのクラス名として使われる．英字名を記入しなければシステムが自動的に割り当てる．

(2) 関連クラス

このクラスが関連するクラスを示す．クラス間の関係は汎化，集約，関連の3種類を設けた．

4.2.2 属性セクション

(1) 属性名

属性名はクラス名同様に日本語と英字名を対で記述できる．英字名の記述がなければ自動割当てとなる．

(2) 属性のデータ型

データ型は基本データ型，配列型とクラス型があり，基本データ型には整数型，実数型，論理型，文字型とその組合せ構造体のバリエーションがある．また，クラス型は属性として基本データ型とクラス型を内包する．

(3) 属性の有効範囲

私有，限定共有，共有の3種類を設けた．

4.2.3 動作セクション

動作記述にはオブジェクト指向を特長付けるオブジェクト指向記号を導入するが，全体的にはドメインユーザのメンタルモデルと考えられる手続き型のプログラミング記述を踏襲する．また，その記述は外部インタフェースを記述する動作インタフェース部とアルゴリズムを記述する動作記述の本体部から構成し，動作インタフェース部に記述したインタフェースは基本的に他クラスに公開されるものとする．

(1) 動作インタフェース部

インタフェース名（メソッド名相当）および引数名は日本語，英字名の両方が記述でき，英字名が省略されれば自動割当てとなる．

(2) 動作本体記述部

動作本体の部分は抽象的な記述であるアウトライン記述（上位階層）と具象化した日本語SMDLプログラム記述（下位階層）の2階層に分けて記述する．

(a) 上位階層（アウトライン記述）

1行程度の自然言語またはOOJ表現による連接（先頭に□記号を付す）で，プログラム手順のアウトライン（プログラムスライスと呼ぶ）を記述する．必要ならその順序は後で入れ替える．プログラムスライスの中身でよく使用するものはコーディングイデオムとして最小部品とし，リポジトリに蓄積できる．なお，上位階層の記述はC++変換時はコメントとなる．

(b) 下位階層（日本語SMDLプログラム）

上記アウトラインの1行ごとに日本語SMDLプログラムを記述する．下位階層のプログラムはトランスレータの対象となる．

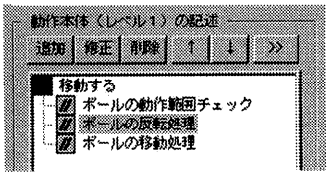


図8 アウトライン記述例（日本語 SMDL）
Fig. 8 Description example of outline of behaviors.

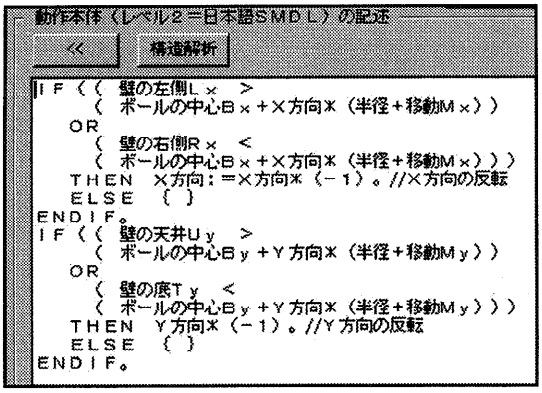


図9 日本語 SMDL の記述例
Fig. 9 Description example by Japanese SMDL.

(3) オブジェクト指向記号の定義

オブジェクト ← メッセージ

（例） スカッシュボール ← 移動する［速度］.

なお，OOJ のインスタンス生成記号に相当するものはない．インスタンスは C++と同様に属性に記述されれば生成される．

(4) 手続き型プログラミング

動作本体記述の基本形は理解しやすさを考慮して制御文，四則演算にはオブジェクトにメッセージを送る形式は採用せず，手続き型の考えに従う．

（a）制御文

識別のしやすさから OOJ 同様に英文字を混在させて使用する．

（例） **LOOP**，**IF**，**CASE**

（b）四則演算

手続き型言語を踏襲する．

（例） 距離 := 速度 * 時間.

4.3 動作本体の記述例

図8 にアウトライン記述の例として図5のクラス「ボール」の動作「移動せよ」の部分を記述例を示す．図8の記述の2行目の「ボールの反射処理」を日本語 SMDL でアルゴリズムを記述した例を**図9** に示す．本例は IF 文による衝突をしたか否かの判定処理，代入文，メッセージ送信を含む例である．

```
class　モデル定義
  属性：　//　下記を文字型で保持する
      クラス名
      クラス間関係
      属性データ
      動作インタフェース
      動作本体
  動作：
      トランスレートする
      リポジトリへの入出力
      ・・・・・
```

図10 〈モデル定義〉クラス概略
Fig. 10 Abstract constitution of 〈model definition〉 class.

4.4 トランスレータ

(1) モデル定義クラス

日本語 SMDL 記述は既存のオブジェクト指向プログラム言語 C++へ変換される．変換を行うトランスレータは本環境を構成するクラスの1つである〈モデル定義〉クラス[9]の1個のメソッドであるという位置付けにある．〈モデル定義〉クラスは日本語 SMDL 記述の管理・格納・変換を行う役割を担う．**図10** に〈モデル定義〉クラスの概略仕様を示す．〈モデル定義〉クラスは属性として日本語 SMDL のプログラム構造をそのままの形で保存し，動作としては日本語 SMDL から C++へのトランスレート，リポジトリへの入出力等の機能を持つ．トランスレータ機能をクラスの振舞いの1つとしたが，これはプログラミングツールのクラス化といえる．

(2) トランスレータ

本トランスレータは C++コンパイラのプリプロセッサで，日本語の単語（識別子）の英数字への置換，オブジェクト指向記号，制御文等の C++構文への変換を行い，既存の C++処理系へそのまま読み込むことのできるプログラムへ変換する役割を持つ．

図11 にトランスレータのアルゴリズムを示す．2行目の前処理の「対象世界探索準備」はトランスレートの対象クラス以外を必要に応じて探索できるようにあらかじめメモリ上に上げることを指している．また，トランスレータは大きくは2段階，すなわちプログラム構造チェックと構文解析に分かれており，C++への変換や日英単語の置き換えは構文解析時に解析をしながら処理する．構文解析時には〈識別子の処理〉を呼び出すが，この中でインクルード文等を自動生成するデータを収集する．

本トランスレータの重要な特長は，クラス間関係がリポジトリに蓄積されていることを利用して，トラ

```
前処理
  対象世界探索準備
  変換対象日本語SMDL読み込み
プログラム構造チェック
  制御構造と括弧の数，句点，セミコロンの
  合理性をチェックする．
  if (プログラム構造は正常か？)
    then  構文解析へ．
    else  処理終了へ．
  endif
構文解析
  各文のトップダウン構文解析処理を行う．
  識別子が現れたら<識別子の処理>を呼出す
後処理
  インクルードファイル名処理
  extern宣言等の処理
        ----------
<識別子の処理>
  if 識別子＝属性名
    then
      if 可視性＝OK  //継承クラスの探索
        then 日英置き換え
        else エラー出力
      endif
  endif
  if 識別子＝メッセージ名
    then   対象世界探索
           日英置き換え
           if メッセージ名＝C関数名
             then extern宣言前処理
           endif
  endif
```

図 11 トランスレータのアルゴリズム
Fig. 11 Algorithm of translator.

ンスレートの対象となっているクラスに関わるクラス（親クラス，メソッドのインタフェース）を含むファイルを自動収集してインクルードファイルとして出力することである．また，日本語SMDLプログラムは全角漢字，全角英数字，半角英数字の混在を許すが，ユーザがあまり入力形式を意識しなくとも済むように，合理的に変換する．本トランスレータにより自動変換され出力されたC++プログラムの一部を**図12**に示す．

5. OODJ一貫支援エディタ

5.1 一貫支援型エディタの概要

開発過程を通じて当初のドメインユーザのモデルイメージを一貫して保てるように，要求記述を順次，連続的に日本語SMDLプログラムまで書き換えていくことを支援する連結型のエディタ群を設計・開発した．**図13**に連続的に変化する記述過程を支援するエディタの構成を示す．矢印は上から下へと左から右へが具象化の過程，反対が抽象化の過程である．エディタは大きく分けてOOJまでを編集する日本語メモ記述エディタと，それ以降のすべての過程を支援する日本語SMDLエディタとからなる．

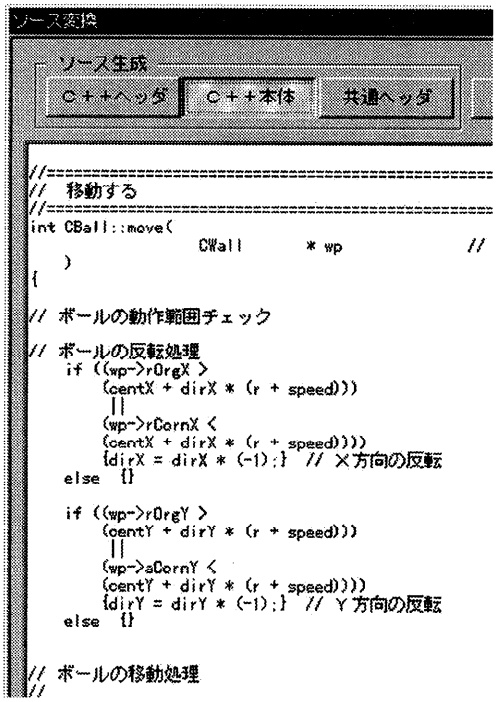


図 12 変換されたC++プログラム例
Fig. 12 Translated program example by C++.

5.2 日本語メモ記述エディタ

図14（図13の①）に示す本エディタは日本語およびOOJ記述を自由形式で記入するエディタである．ユーザは要求記述，あるいはクラス概要記述をこのエディタで行う．

本エディタの特長は通常のエディット機能に加えて，ユーザの記述した日本語の文章の中からクラス名候補，属性名候補，メッセージ名候補をマウスを用いて切り出し，リポジトリ中のデータベース（プールという）へ登録することができることである．切り出し，登録は通常のメニューとポップアップメニューの両方が用意されている．図14ではクラス名候補としてOOJで記述したクラス概要記述の中から「ボール」をクラス名候補として，マウスで指定し，登録しているところである．

5.3 日本語SMDLエディタ

プールへ切り出したクラス名，クラスの属性名，クラスの動作名候補等は次のステップの日本語SMDLエディタを起動したときに，リポジトリから自動的に取り出されて**図15**に見るようにエディタの主画面中に自動的に展開され，クラスフレーム（枠組み）を形成する．クラス名候補はクラス名の個所に，属性名候補は属性名の個所に各々リスト表示される．日本語SMDLエディタの主画面は図13の②に示すように次

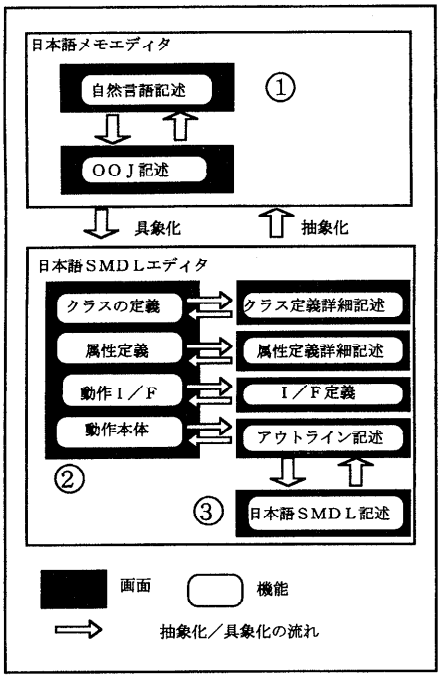


図13 一貫支援エディタの画面関連図
Fig. 13 Constitution of integrated support editor.

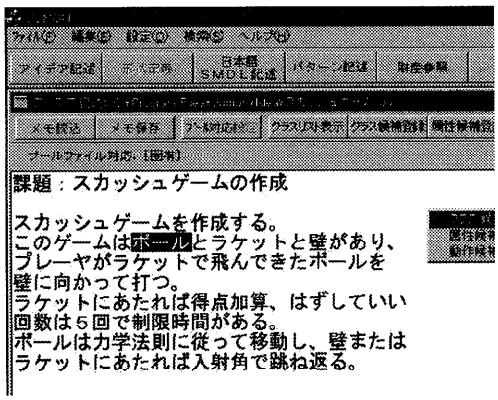


図14 日本語メモ記述エディタ
Fig. 14 Japanese memorandum description editor.

の4種類からなる．

(1) クラス定義，属性，動作I/F（インタフェース）

これらの部分はエディタの主画面と各々定義詳細を入力する補助画面からなり，ユーザは画面表示に従って単語をキーインしたり，リスト表示から必要事項を選択する方式となっている．たとえば，図15はクラス名として「ボール」を選んだ後，ボールクラスの属性として「ボールの位置」を選択しているところで，データ型はリストから選択する．

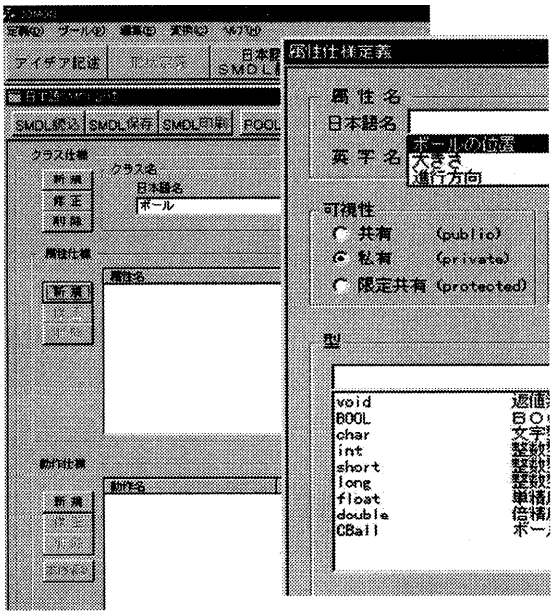


図15 日本語SMDLエディタへの自動展開
Fig. 15 Automatic extension to Japanese SMDL.

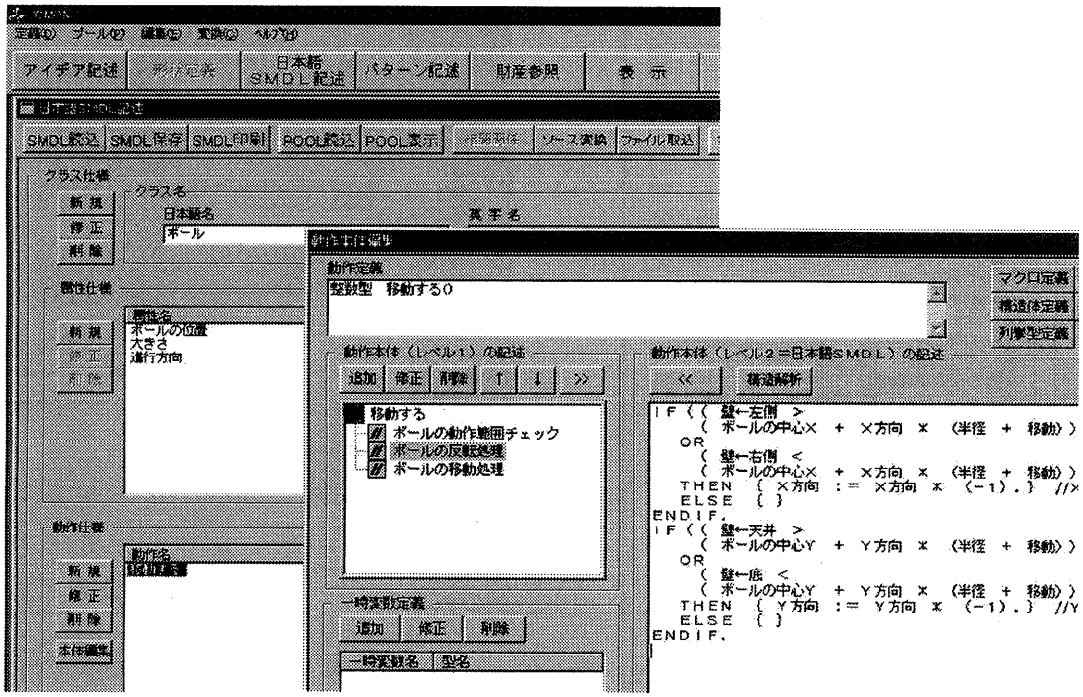
(2) 動作本体

図16 (a) は画面で動作インタフェースとして「移動する」を選択し，図16 (b) 画面でその本体を記入する例である．図16 (b) 画面の左側は上位階層で「移動する」のアウトラインを記入したところで，画面の右側はアウトラインの一部である「ボールの反転処理」の詳細を日本語SMDLで記入するところである．詳細記述はすでに図9で示した．

5.4 リポジトリ

日本語SMDLプログラムからC++プログラムを出力するためには作成中のプログラムにかかわるさまざまな情報，たとえば，上位クラスの情報，メッセージを送付している相手のクラスに関する情報等が必要となる．このため，各種生成物の蓄積と管理を行うリポジトリをWindowsのファイルシステムを利用して開発した[10]．

図17に対象世界単位にクラス管理を行うリポジトリの内部構造を示す．対象世界の管理表は日本語メモエディタで作成されたメモ（要求記述，分析モデル記述等），クラス名等を格納したプールおよび対象世界を構成するクラス管理表を記憶させる．各クラス管理表は属性，動作およびトランスレータ結果のC++プログラムを管理する．テキスト実体（日本語メモ，SMDLプログラム，C++プログラム）は管理表と分離してテキストファイル群として別に記憶させる．また日英辞書（対応表）は分離されて別に実体を持つ．



(a) (b)

図 16 日本語 SMDL エディタ
Fig. 16 Japanese SMDL editor.

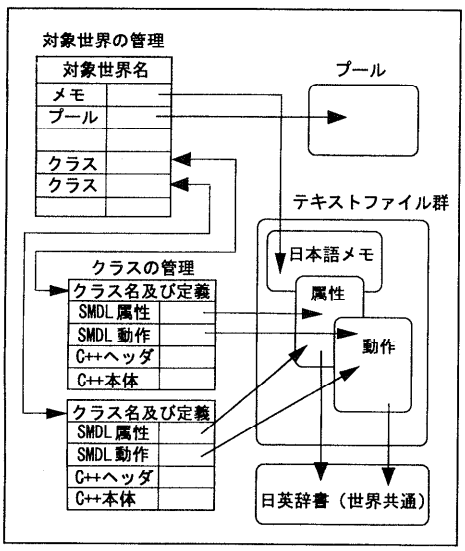


図 17 リポジトリの構成
Fig. 17 Constitution of object repository.

このような構成にすることで，対象世界の管理表から対象世界全体の検索，対象世界の管理とクラス管理の双方向リンクから親クラス等の関連クラスの検索，日英変換辞書の検索等を簡便に行える．

リポジトリ内の管理は植物の成長シミュレーション，スカッシュゲーム，画像処理といった対象世界単位に管理する．また，日英辞書も同じ言葉が対象分野で異なる意味を持つ場合があるので特定の対象世界単位で持つ．

6. ソフトウェアの全体構成

本環境自体もオブジェクト指向を用いて開発しており，約 80 個のクラスから構成される．ソフトウェア構成は図 **18** のようになり，グループ化された 4 つの層，すなわち，ユーザとのインタフェースを実現するインタフェース（I/F）層，日本語メモや OOJ，日本語 SMDL プログラムからオブジェクトを生成するオブジェクト生成層，これらの生成されたオブジェクトを管理し，リポジトリとのデータ交換を行うオブジェクト管理・アクセス層，および OS に依存あるいは利用するシステム層からなる．また，全体の構成は下の層が上の層のデータ（あるいはオブジェクト）を処理する形のアーキテクチャとした．

基本 OS は Windows95/NT で，GUI フレームワークとしては MFC（Microsoft Foundation Class）を利用して実装している．

表1 Nステップ開発手順に基づくプログラム作成
Table 1 Program development based on N-steps development procedure.

Nステップ開発手順			ユーザ操作（U表示）とシステム動作（S表示）	対応図
自然言語による要求記述			U：日本語メモエディタで要求を記述 S：エディット、メモをリポジトリへ格納	図 3 図 14
ラフスケッチ及びOMT等による分析			U：ペーパー等にゲームのラフスケッチ及びOMTのクラス図を必要に応じて描く	図 4 図 5
OO－J記述	クラス概略記述		U：日本語メモエディタでクラスの属性、動作の項目を記述し、プールへ登録する。 S：文字列をリポジトリ（プール）へ格納する。	図 6 図 14
	動作記述		U：日本語メモエディタで動作概略を記述する。（ポイントとなるアルゴリズムを中心に） S：文字列をリポジトリへ登録する。	図 7
日本語SMDL記述	クラス記述		U：日本語SMDLエディタを起動する。 S：プールからクラス名等を読み込み、表示する。	図 15
	属性記述		U：表示されたデータ型候補から選択する。 S：リポジトリ内から属性候補をサーチ、表示し選択されたデータ型を属性にセットする。	図 15
	動作インタフェース記述		U：動作インタフェースの詳細を記述する。 S：リポジトリに格納する。	図 16
	動作本体記述	アウトライン	U：アウトラインを1行化日本語で記述する。	図 8
		詳細手順記述	U：詳細アルゴリズムを記述する。	図 9 図 16
トランスレータによるC++変換			U：変換するクラス名を指定する。 S：C++へ変換する。ヘッダファイルと本体のファイルが生成される	図 12

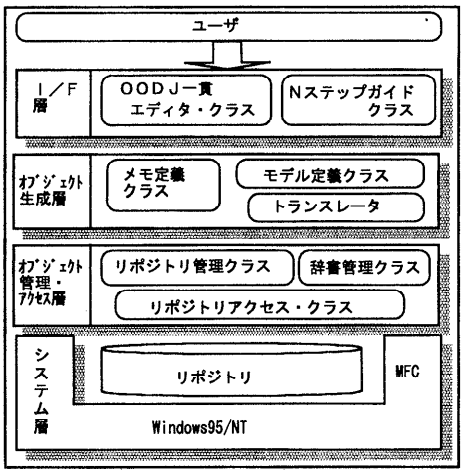


図18 ソフトウェアアーキテクチャ
Fig. 18 Software architecture.

7. 適用事例とその評価

本章ではまず，6章までにおいて一貫して取り上げてきたスカッシュゲームを例に日本語一貫プログラミング環境がシステム全体の方法論として実現していることをまとめて示し，本環境の有効性を定性的に評価する．次に本環境の有用性・実用性を客観的かつ定量的に評価するために行った記述実験を示し，その結果を評価する．最後にすでに2年前後利用されてきているJOMON[11]，JOMON+[12]（いずれも，本環境の社内コード）についての社内評価について述べる．

7.1 日本語一貫プログラミングの検証

日本語一貫プログラミングの記述例として，全体が見通しやすく単純なゲームであるスカッシュゲームを取り上げてきた．表1にこれらをまとめて示す．表からNステップ手順とその対応図（図3～図16）を対照させて巡覧すれば，図2の実現，すなわち，OODJ（総称）という日本語系自然言語を一貫して用いたプログラミング環境の実現例が一目で概観できる．

本適用事例は論文中に引用するため規模の小さい事例を用いた．しかし，表1の並びとその変換例の図を対応させて追っていけば，OODJの3種の日本語の記述とその変遷およびその変換の様子が，自然言語（風）であるだけに，ほとんど説明がなくとも抵抗なく理解できる．そしてオブジェクト指向記述日本語（OODJ）を用いた一貫プログラミングの実現は明らかであろう．また，表現系の中核である日本語SMDL記述も，対応図中のOODJの記述を順に追っていけば，複数（N）のステップを経ているので，各ステップ間の変換落差が小さくなったゆえに変換が容易にな

表2 本環境の方式と従来方式の比較
Table 2 Comparison present programming environment with current one.

比較項目	概念モデル（記述量）	論理モデル（記述量）	プログラム（記述量）	モデル～プログラム	テスト	全開発期間
本環境方式	日本語 ２０行	ＯＯＪ ５４行	日本語ＳＭＤＬ ２５０行 (生成870行)	７日間	６日間	約２週間
従来方式	日本語 ２０行	OMT １４ｸﾗｽ図	C++ ４９０行	１０日間	１０日間	約３週間

り，かつ相似変換が実現されやすくなったことが容易に推察できよう．以上から，不足機能はまだ残るとしても日本語一貫プログラミングはシステム方式としては実現できたと評価できよう．

またOODJ一貫支援エディタの機能は，たとえば図3，図6，図7，図8，図9において，枠組みと選択肢によりガイドをする形で記述支援されており，図14，15，16の作業工程を見れば，それら3種のOODJ間の記述支援が「一貫して」実現されていることが読みとれよう．

7.2 日本語SMDLの記述実験

本システムの開発効率を計測するために2つの記述実験を従来の環境と本環境の両方式で行い，比較，評価した．

7.2.1 記述実験1：新規プログラム作成

(1) 実験の概要

実験の記述テーマとして植物の成長シミュレーション記述を取り上げることとした．内容は緑色植物が光を受け，成長する様子をシミュレーションするものである．実験の比較項目は開発全体に要した時間，開発各段階の記述量である．

実験の対象者は2つのグループに分けた．本システムで記述する対象者（以降Aグループ，2人）としてはオブジェクト指向の概念を理解し，FORTRANプログラミングが多少できる者を選定した．一方，従来方式のOMT，C++で記述する対象者（以降Bグループ，2人）としてはオブジェクト指向の概念を理解し，かつC++プログラミングの多少経験のある者を選定した．Aグループの方がプログラミングの経験は浅い．

(2) 実験の手順

要求記述は実験課題として与えた．また，クラス分割のイメージはあらかじめ与え，概念モデル，分析段階での差が出ないように配慮した．AグループはOOJでクラスの概略，日本語SMDLで詳細な動作を記述した．BグループはOMTでクラス図の記述を行った後，C++でプログラミングを行った．

事前教育としてはオブジェクト指向の概念について共通のセミナー受講をさせた以外に，Aグループは日本語SMDLとエディタの教育を，BグループはOMTとC++の教育を行った．

(3) 実験結果と評価

クラス数は14個と小規模なプログラムがともに作成された．各項目について比較した結果を**表2**に示す．

モデル記述の時間は対象世界の内容がシンプルな点，そしてクラスイメージを与えたためか，ほとんど差がなかった．プログラムの記述量の差は，属性部分の記述量が違うこと（ヘッダーファイルの自動出力）およびextern，include宣言の自動出力が大きく差となって現れた．動作記述，クラスの振舞いおよびメインプログラムに相当する記述での記述量差はあまりなかった．プログラミング時間の差は日本語SMDL記述の容易さとC++記述の複雑さの差であった．C++に慣れると差はなくなると考えられるが，ドメインユーザはC++で記述するのが苦手でそれを好まず，またC++の学習に多大な時間を割くのを望まないという事情（1章「はじめに」，および同脚注参照）を考慮すると，日本語SMDLによる記述方式はドメインユーザには大きなメリットがあることが分かる．テスト時間にもC++の複雑さによる時間差が現れた．

本実験の結果，事例としては小さいプログラムではあるが，総合的には新規にプログラムを作成する場合の開発効率は約1.5倍という評価を得た．本実験の評価法では開発プログラムの規模が大きくなるにつれて，記述力の比率はさらに大きくなると考えられる．C++記述に不慣れなドメインユーザではその比率はさらに大きく開く．

最後に，教育期間はそれぞれ3日間であったが，それに続く学習時間は日本語SMDLの構文がシンプルであるために本環境の方が格段に短くて済んだ．C++は細部を学ぶには実プログラミングでの試行錯誤が必要で，マスターするには時間がかかる．

表3 Cプログラムの書き換え
Table 3 Rewriting C program to Japanese SMDL program.

	記述行（動作記述）	
既存Cプログラム	C言語記述	1．2k行
書き換えたプログラム	日本語SMDL C言語記述残	0．5k行 0．6k行

以上の実験結果はいずれも，ドメインユーザに大きなメリットをもたらすことを実証したものと評価した．

7.2.2 記述実験2：既存プログラムの書き換え・再利用[12)]

(1) 実験の概要

既存のC言語プログラムを本システムを用いてオブジェクト指向記述に書き換え，記述力の強さを評価する．既存のCプログラムは画像処理を用いて魚群分布を認識するものを取り上げる．比較項目は既存のCプログラムがどのくらい日本語SMDLで書き換えられるかをプログラムのステップ数で比較する．

(2) 実験手順

まず，既存の画像関連のC言語ライブラリ（関数）120個を本システムを用いて日本語ライブラリ化する．ライブラリ化はインタフェースのみを日本語化し，各関数の中身はC言語のままとする日本語ラッピングを行った．関数群はクラスのメソッドになる．なお，単純に関数をクラスへのメッセージ形式に書き換えると，実行効率を下げる問題が発生したが，間にインライン関数を自動的にはさむことで解決した．これにともない本環境の改訂が発生したので，これを行った．

次に上記日本語化ライブラリを用いて既存の約0.4k行のCのアプリケーションプログラム3個をオブジェクト指向プログラムへと書き換えた．

(3) 実験結果と評価

結果を**表3**に示す．平均的には各モジュール記述行の約半分はオブジェクト指向日本語SMDL表現に書き換えられた．書き換えができない部分は既存の共通変数へのアクセス等である．原因は本システムでは属性のデータ型を限定したのと，その属性へのアクセスが他の既存モジュールとの関係で書き換えられなかったためである．日本語SMDLではまだC++との完全互換とはしていないので，部分的な書き換えではかえって属性データ型の限定がネックになる．なお，現状はこのような特別な場合に対しては，トランスレータが英小文字を無変換のまま出力する仕様なのを利用して日本語SMDLの中にC++言語で書くことで対応している．しかしこの場合，トランスレータ出力とユーザ記述のC++記述間での食い違いの調整作業が発生する．

以上の結果として基本機能としては一応の記述力はあると評価できるが，上記のようなプログラミングの要求にはまだ不足機能が見出される．

7.3 実適用での経験評価

本節では本システムを道路交通シミュレーション（一部）と画像認識プログラムの両開発に利用してきた結果をまとめて示す[12)]．

道路交通シミュレーションは新規に設計したシミュレーションシステムで，本環境適用部分は大きさとしては約2k行である．一方，画像認識プログラムは7.2.2項の書き換え・再利用を行った実験である．以下は上記システムの開発経験から得た社内ユーザ評価を抜粋したものである．

(1) 開発プロセスを類似の日本語文の言い換えや詳細化した記述でとらえることができたので，開発各段階の移行にスムーズ感が得られ記述ミスが減少した．

(2) 日本語SMDLでは識別子（日本語）をOOJからそのまま移行できるので，識別子のネーミングが楽であり，対応関係も分かりやすい．

(3) オブジェクト指向に不慣れであったが（道路交通シミュレーションのユーザ），本システムを使用することで，オブジェクト指向による開発の全体の流れを学習するのに大いに役立ち，学習システムとしても有用と考える．開発効率化（1.5倍）よりも心理的な効用（見なれた日本語記述の既視感）が大きいと感じた．

(4) 日本語インタフェース化したライブラリを再利用して新規プログラムを開発した場合（画像認識プログラム）では，その作成時間も数分の1であり，かつ自然語風な記述ができたので，作成者だけでなく他の共同作業メンバの理解や共有化も容易であった．ライブラリを充実させれば，本システムを用いた移行は従来システムよりはるかにスムーズで短時間であると評価できる．

(5) システム開発全体の流れの見通しの良さは評価できる．しかし，分析・設計段階のOOJは自由度が高すぎ，慣れるまではどう記述すれば良いのか迷う．

(6) 日本語表現が主体であるためOODJ文の記述自体が詳細なコメントの役割を果たし，第三者にも読みやすく，開発ミスを減らし，保守性も良い（図12もその例）．

(7) 日本語SMDLによるプログラミングではインクルード文，C関数引用時のextern宣言が自動生成されるため，システム情報をプログラミングする煩わしさから解放されたのは高く評価したい．

(8) 日本語SMDLではポインタ型を使用するような細かい記述を現状では許していないため，既存のCプログラムをうまくオブジェクト指向プログラムには変換できず，書き換える必要が生じた．新規開発ではこのようなプログラミングを避けることができるが，既存プログラムの書き換えには記述力が不足する．この点は改良の余地がまだある．

(9) 日本語SMDLエディタの動作記述はアウトラインと詳細記述の2段階であるが，アウトライン記述に煩わしさを感じた．しかし上位階層でプログラムの要旨を把握し，下位階層でアルゴリズムが把握できるので，双方向での直接比較が可能となり可読性・了解性[2]やドキュメント性（SMDL記述自体がドキュメントとしての役割を果たすこと）は従来のプログラム言語よりもはるかに良好になった．

(10) 本環境を製品としてみるにはまだ完成度があまり高くなく，特に辞書やライブラリの整備が進んでいないために，ドメインユーザの負担はメソッドの実装部分に関してだけは負担軽減にはつながっていない．このへんを解決するには日本語インタフェースによるラッパー[12),13)]を用いた既存メソッドや関数の再利用でこれを補うべきであると考える．

8. 考察と議論

8.1 日本語一貫プログラミングについての考察

Nステップ開発手順の基本はウォーターフォール型のソフトウェア開発手順である[14)]．しかし，従来のウォーターフォール型の開発プロセス実現では各段階の要求記述，仕様書，OMT表記[15)]，C++プログラムと個々に対応する部分の記述系や表現法が異なるため，直接の対応関係が見えにくい記述作業を多く抱えている．このためユーザ自身もそれを読む第3者も大きなギャップを各々自力で埋め，対応関係の連続性や相似性を各個人ごとに改めて確認し，それらを評価する必要があった．

これに比べて本方式では開発の各段階に適応した同系統の限定日本語記述言語を導入して言語面からシームレスな変換を実現している．この試みは7.1節の記述例，7.2節の記述実験および7.3節のアンケート(1)から(4)に見られるように一定の評価を得た．したがって，モデル化とプログラミングの記述系に，ドメインユーザがその専門分野において持つ強い表現力を発揮して記述でき・理解できる自然言語系列を一貫して用いる記述方式を採用した方針およびその実現は，成功したと判断してよかろう．もちろん，一部アンケート(5)，(8)，(10)や7.2.2項の評価にあるように機能の一部や性能についてまだまだとの意見もあり，次段階への課題もある．

8.2 日本語SMDLの考察

日本語SMDLは7.3節のユーザ評価(8)に見られるようにまだ記述力不足があるが，現状，新規で数百行，再利用で数千行の大きさを持ったプログラムを作成できた（7.2節，7.3節）ことで記述力は一応十分な水準に達していると評価できる．しかし，自然言語（日本語）の強い記述力はまだ十分には発揮できる記述言語システムにはなっておらず，まだ潜在している自然言語の力を生かせば記述力を飛躍的に高められる可能性があると考える．

また，記述性は7.3節の(2)，(3)，(4)，(6)，(9)に見るように通常のプログラミング言語より理解が容易，7.2節に見られるようにプログラミング言語直接の記述を少なくできるという評価であり，したがって，ドメインユーザに大きなメリットをもたらす特長を持つといえよう．

8.3 OODJ一貫支援エディタの評価と考察

本プログラミング環境によりドメインユーザはオブジェクトを定義・生成する際にC++等のプログラム言語を直接使わずに，日本語風の記述のまま連携・連続作業を支援するエディタ群（図13）を用いてNステップを経て（図3 → 6 → 7 → 8 → 9の順に），順次日本語SMDLに変換していけば従来システムに比べて違和感少なくオブジェクトが記述できた．一貫してOODJを用いてプログラミングができるような環境になったか否かという意味では，そうなったと判断しており，未完成部分（たとえば，ユーザ評価(5)，(9)，(10)）を認めたうえで本エディタ環境は一応実用化できたといってよいであろう．

8.4 関連研究との比較

ソフトウェアの開発を支援する環境の多くはCASE環境[16)]として実用化が図られている．しかし，ドメインユーザ向けに概念モデルの作成からプログラムの実行までを一貫して日本語風記述系で支援する環境は見当たらない．CASE環境に第4世代言語を組み込んだ例[17)]では処理の記述を日本語で表現し，COBOLプ

ログラム等を出力するが，適用分野がビジネス分野に限られている．市販ソフトのオブジェクト指向CASEツールのROSE[18]等は表現言語としてC++を用いており振舞いの抽象表現等は実現していない．

専用ソフトウェア環境，言語によるアプローチ法[19),20)]も適用分野をより限定する方法であり，一部は成功を収めているが，分野が狭く特定されるために専門領域が合わないと使えない欠点がある．これに対し，本研究における環境はモデルを試行錯誤的に作成するドメインユーザにとって専用環境に比較し記述に汎用性を持つとともに，分野ごとの辞書充実により特定分野への環境へも移行が十分可能であるというほかにはない特長を持つ．

また，プログラムの製作を容易にする視覚的プログラミング[21)]は魅力的ではあるが，用途に合わせた手作りの多い応用工学分野の複雑多岐にわたる要求には視覚化の煩わしさゆえにとても向かない．たとえば，数値風洞システムの境界条件は数十個の類似のクラス[22)]の視覚化が必要となる．これに比して本環境では部品名称が日本語であるため，分かりやすい部品名称や部品リストを設定できる．

9. 結論と今後の課題・展望

本論文においては，オブジェクト指向に基づき一貫して日本語系記述を用いることができるプログラミング方式および個人規模用のプログラミング環境を提案・開発した．モデル化・プログラミング過程は複数の開発手順に区分され，各ステップに対応する日本語系記述言語（OODJ）とGUIガイダンスおよび一貫支援エディタ，リポジトリ環境等が考案・実現された．これらにより，本論文の目標であった日本語一貫プログラミングの実現とその有効性が検証された．

日本語プログラミング記述言語である日本語SMDLの記述力およびトランスレータも評価され，一応十分な有用性・実用性を持っていることが記述実験と長期の社内評価から明らかにされた．OODJ一貫支援エディタについても実用には耐えるものと判定されている．関連研究の調査から本研究の方式と直接競合するものは見つからなかった．

以上から結論として，オブジェクト指向日本語系記述言語で一貫して記述し，最後にトランスレータによりオブジェクト指向言語プログラムへと自動変換するという方式の合理性，およびそれを実現したプログラミング環境の有用性・実用性が実証された．その過程で以下の今後の課題が見出された．

- OODJの言語仕様の充実
- 各分野別辞書や日本語ラッピングした再利用ライブラリの充実
- 製品としての使い勝手や完成度の向上

今後の課題としては現状のOODJの仕様および実現では，自然言語日本語の持つ強い記述力や豊富な記述性は十分には発揮されてはいない．そこでOODJの仕様と設計に自然言語の記述力・記述性をさらに強化し，一方で日本語SMDLに至る全過程においてドメインユーザに十分な知識記述をさせる（これは十分期待できる．1章「はじめに」参照）．それらをOODJで十分取り込んで記述できるようなプログラミング環境を構築することで，より完成度の高いC++プログラムの自動生成が期待できる．これは自然言語系記述をプログラムに自動変換する方法の1つとして，あるいは，ドメインユーザのオブジェクト指向言語による直接記述力の弱さを飛躍的に強化する方法の1つとして，展望を持てる方法論ではないかと考える．

参考文献

1) 米沢明憲：オブジェクト指向計算の現状と展望，情報処理，Vol.29, No.4, pp.290–294 (1988).
2) 米沢明憲，柴山悦哉：モデルと表現，岩波書店 (1992).
3) 落水浩一郎：ソフトウェア・リポジトリ，情報処理，Vol.35, No.2, pp.140–149 (1994).
4) 加藤木，畠山：オブジェクト指向シミュレーション・モデル記述の開発環境，情報処理学会第98回ソフトウェア工学研究会，Vol.94, No.43, pp.9–16 (1994).
5) 本位田真一，山城明宏：オブジェクト指向分析・設計，情報処理，Vol.35, No.5, pp.392–401 (1994).
6) Meyer, B.: *Object-Oriented Software Construction*, Prentice-Hall (1989). 二木（監訳），酒匂（訳），オブジェクト指向入門，アスキー．
7) 小林，畠山：オブジェクトベース・リポジトリの概念設計と実装，オブジェクト指向'95シンポジウム，情報処理学会シンポジウム論文集，pp.309–316 (1995).
8) 加藤木，畠山：オブジェクト指向日本語一貫プログラミング環境，情報処理学会第118回ソフトウェア工学研究会，Vol.98, No.20, pp.15–22 (1998).
9) 加藤木，畠山，小林：オブジェクトベースト・リポジトリを用いたオブジェクト生成支援環境，情報処理学会第101回ソフトウェア工学研究会，Vol.94, No.99, pp.57–64 (1994).
10) 加藤木，畠山ほか：シミュレーション向けのオブジェクト生成支援環境の設計と実装，オブジェクト指向'95シンポジウム，情報処理学会シンポジウム論文集，Vol.95, No.3, pp.205–212 (1996).
11) 加藤木，畠山：オブジェクト指向日本語プログラミング環境「JOMON」，HIPRO技報，創刊号，

pp.53–58 (1996).
12) 加藤木，畠山ほか：統合日本語プログラミング環境「JOMON+」，HIPRO 技報，第 2 号，pp.35–40 (1997).
13) Booch, G.: *Object-Oriented Analysis and Design with Application*, 2nd ed. (1993).
14) 松本吉弘：ソフトウェア工学，丸善 (1992).
15) Rumbaugh, J., et al.: *Object-Oriented Modeling and Design*, Prentice-Hall (1991). 羽生田（監訳），オブジェクト指向方法論 OMT，トッパン.
16) 大筆　豊，川越恭二（編）：特集 CASE 環境，情報処理，Vol.31, No.8, pp.1012–1094 (1990).
17) 吉野松樹，田村和敏，稲益良夫：ソフトウェア開発支援ツール SEWB3，EAGLE/4GL の機能と特長，日立評論，Vol.75, No.11, pp.21–28 (1988).
18) Rose/C++ 4.0 マニュアル： RATIONAL SOFTWARE CORPORATION (1997).
19) 佐川，金野，梅谷：数値シミュレーション言語 DEQSOL，情報処理学会論文誌，Vol.30, No.1 (1989).
20) 廣田豊彦ほか：応用ドメインに特化した概念モデル記述言語に関する一考察，情報処理学会論文誌，Vol.36, No.5, pp.1151–1161 (1995).
21) 西川博昭，寺田浩詔：視覚的プログラミング環境，情報処理，Vol.30, No.4, pp.354–362 (1989).
22) Hatakeyama, M., Katougi, K., et al.: Object-based Simulation World Development System, *3rd Asian-Pacific Conference on Computational Mechanics (APCOM)*, pp.2485–2490 (1996).

(平成 10 年 4 月 8 日受付)
(平成 11 年 4 月 1 日採録)

加藤木和夫（正会員）

1970 年北海道大学理学部物理学科卒業．現在，日立プロセスコンピュータエンジニアリング（株）勤務．制御用プログラミング言語，ソフトウェア開発環境等の研究・開発に従事．著書「ソフトウェア開発環境」（共著），「PAD によるプログラム技法」（共著）ほか．技術士（情報工学部門）．電子情報通信学会会員．

畠山　正行（正会員）

1976 年東京大学大学院博士課程（航空学専攻）修了．工学博士．東京都立航空高専を経て，1990 年 10 月より，茨城大学工学部情報工学科専任講師，現在に至る．この間，超音速凝縮流れ等の非連続気体流れの数値的な解析の研究に従事．次いで，数値シミュレーションユーザ用の計算機環境，オブジェクト指向シミュレーション，エージェントを用いた数値シミュレーション手法の研究開発に従事．日本機械学会，日本航空宇宙学会，日本数値流体力学会，日本計算工学会，日本シミュレーション学会，電子情報通信学会，ACM 各会員．