ソフトウェア工学 130－22
（２００１．３．２３）

オブジェクト指向プログラム設計記述言語ＯＯＰＤの設計

加藤木和夫*　　畠山正行**　　上田賀一**

*日立プロセスコンピュータエンジニアリング（株）　　**茨城大学工学部情報工学科

科学技術分野のドメインユーザ向けにオブジェクト指向設計記述言語OOPDを設計した．本言語は筆者らが従来から追及している日本語一貫プログラミングの一部で，設計段階の文書を記述する．ここで日本語一貫プログラミングは要求記述段階からプログラム製作段階までを，各段階に対応した複数の相似系シリーズ言語でプログラミングを行う方法である．OOPDは分析設計段階の記述言語OODJの次に位置し，プログラム設計段階に対応した言語である．設計に当たってはすでに実装段階向けに設計し，稼動中の日本語プログラム言語JSMDLを設計記述の観点から見直し，設計記述モデル，構文規則，変換手順等を新たに設計し直した．OOPDは分析記述言語からのスムーズな移行による高可読性とプログラム言語C++へのトランスレータ開発による実行可能性を両立できる言語であり，ドメインユーザのプログラム開発力を大幅に向上させることが可能である．

Design of Object Oriented Program Design description language OOPD

Kazuo Katougi*　　Masayuki Hatakeyama**　　and　　Yoshikazu Ueda**

*Hitachi Process Computer Engineering, Inc.　　** Ibaraki University

To improve the program development ability of the domain users in science and technology fields, we have designed a new description language called OOPD (Object-oriented Program Design description language) based on the native mother language (Japanese) in the program design processes. We have introduced and designed a new description model in the program design process, the grammar of OOPD, and the rules of the translating from the analysis process description language to OOPD.

１．はじめに

科学技術計算・研究系分野のドメインユーザ向けに自然語に近い日本語でプログラムを記述する研究を進めている．研究の動機は本ドメインユーザがプログラム言語の持つ記号性と厳密さになかなか馴染めないで苦労していることと，これと同義になるがプログラミングの負担を軽減してほしいという要求である．解決策として，ドメインユーザの記述する日本語の文章をできるだけ自動的にプログラムに変換できないか，を中心的な課題として研究を進めている．しかし，日本語の文章からのプログラム自動生成はコンパイラが自然言語の持つ多様性・曖昧性・非構造性に対応できないことから明白である．

そこで，ドメインユーザのプログラム開発環境を実現すために，我々は開発過程の各ステップに対応し，自然日本語，オブジェクト指向記述日本語 OODJ(Object-Oriented Description Language)，日本語プログラム記述言語 JSMDL(Japanese Software Modeling Description Language)の複数言語をシリーズ化し，日本語一貫プログラミン

グ[1]として提案・設計・構築した．また，この中で分析段階に対応するオブジェクト指向記述日本語 OODJ は構成や機能に不十分な面があったため，自然日本語からの変換規則，OODJ 文法の大幅な強化・改良を行った[2) 3)]．

本論文では，OODJ の改訂に合わせて設計段階の記述言語 **OOPD**(Object Oriented Program Design description language)を新たに設計したので，報告する．OOPD は先の論文[1)]で開発した日本語プログラム記述言語 JSMDL を次のポイントから全体的に見直し，オブジェクト指向プログラム設計言語として設計した．見直しのポイントは下記である．

(1)言語の位置付け

分析記述段階から実装記述段階までの過程の中で日本語プログラム記述言語 JSMDL をプログラム設計記述言語として再定義する．JSMDL から OOPD への改訂に当たっては，プログラム言語の実行可能性を保持しつつプログラム設計記述という上流工程記述へ言語仕様をシフトする．

(2)設計記述モデルの提示と OOPD への反映

JSMDL はプログラム言語として開発したが，対象世界をどのように記述するかのモデルは特には想定していなかった．本論文では対象世界を設計記述するときの記述モデルを設定，すなわち設計仕様に要求される事項を明確にした上で，それを反映する形で文法を再定義する．

(3)変換手順の見直し

OODJ では対象世界の構造が明確に定義できるようになったが，振る舞いの部分はまだ日本語単文の列である．この日本語単文列をプログラム設計記述言語 OOPD へ変換する手順を見直し，ドメインユーザがスムーズにプログラム化が図れるようにする．

実現目標

上記の見直しにより，可読性の高い日本語文章を実行可能プログラムへ変換する方式を定式化する．また，この定式化を支援環境として実現することで，ドメインユーザが新たにプログラム言語の習得をせずに現在持っている Fortran 程度のプログラム言語知識で，対象世界を日本語でプログラム化できるようにする．

2．日本語一貫プログラミングの概要

日本語一貫プログラミングの記述プロセスと環境[1)]は図１のようになる．

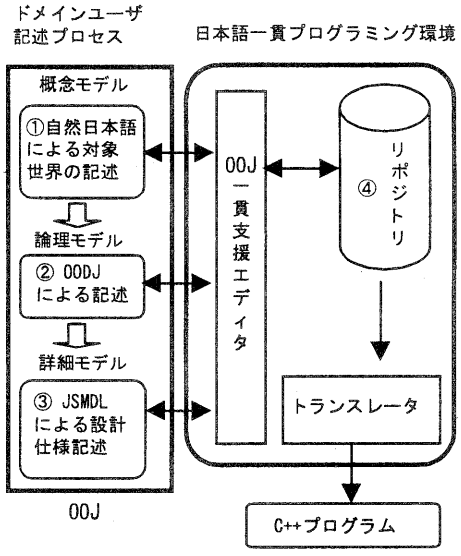

図１　一貫プログラミング

図１を番号に沿って説明する．

①概念モデルでは要求，具体的なシナリオを自然日本語で記述する．

②次に OODJ を用いて論理モデルを作成する．OODJ は自然日本語にオブジェクト指向の構造記述を追加し，自然日本語からの変換規則を定めた言語である．各シング（Thing：クラス相当）の振る舞い（メソッド相当）記述は自然日本語に緩い制約（単文のみ許す．複文，疑問文は排除等）を加えたもので記述する．OODJ で論理モデルを記述終了した時点で，ソフトウェア全体の構造および個々のシングの振る舞い記述を日本語単文の列挙段階まで終えたことになる．

③OODJ で記述した論理モデルは JSMDL を用いてプログラム化する．JSMDL は C++ライクの日本語プログラム記述言語である．この記述結果はトランスレータにより C++プログラムに変換され，

実行される．

④各モデル間の情報はリポジトリに蓄積・利用される．

3．日本語一貫プログラミング環境の課題

3.1　問題点と課題

OODJ文書をJSMDLプログラムへ自動的に変換できる部分はクラス構造，クラス間関係であり，自動的にできない部分は振る舞い記述のところである．現状ではユーザ自身が OODJ の振る舞い記述を見ながら，プログラムへの変換を頭の中で考え JSMDL プログラムを記述するという従来からの方法を取っている．このため，現状の日本語一貫プログラミングでは振る舞い記述に関しては OODJ と JSMDL の間に大きなギャップが残っており，ドメインユーザのプログラミング負担は従来と変わっていないという大きな問題が残っている．これは日本語を用いての一貫プログラミングを実現するために克服しなければならない大きい課題である．

そこで，我々はこの課題を解決するために，JSMDL をプログラム言語より上流のプログラム設計記述言語へシフトし，プログラムへの変換ギャップを少なくし，さらに OODJ からの変換規則を明確にすることで，可能な限りドメインユーザのプログラミング・フリーを実現する方法を取ることとした．ここで導入するプログラム設計記述言語を OOPD と呼ぶ．

OOPD は変換規則により C++へトランスレートできるように JSMDL の実行可能性を残すようにすることで，OODJ 文書の高可読性を維持しつつ実行可能性を実現することを狙いとする．

また，ドメインユーザは従来の一貫プログラミングのプロセスにおいて，論理モデル（シング単位）を JSMDL（クラス単位）へ単純に移行し，設計方法や手順を用いない傾向があり，このため設計の全体見通しを悪くしている．そこで設計仕様の記述モデルを明確にして，この中に設計手順を取り込むことでドメインユーザが設計全体の見通しを良くなるようにする．

設計記述言語 OOPD を導入した後の OODJ から C++に至る変換システムは図２のようになる．変換規則１は OODJ から OOPD への変換，変換規則２は OOPD から C++への変換規則とする．

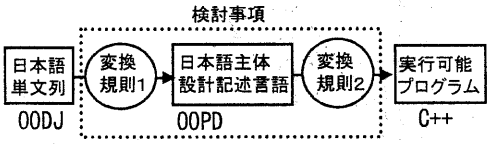

図２　変換システム

3.2　設計課題

設計課題を整理すると，表１のようになる．OODJ のように厳密な振る舞い記述構文を持たない日本語記述を変換規則１でできる限り設計記述言語 OOPD に近いところまで変換する．設計記述言語 OOPD はプログラム設計の記述実現が重要なポイントとなる．最後の変換規則２は既存の JSMDL トランスレータ（C++コード生成規則）の再設計となる．

表１　設計課題

分類	変換規則１	設計記述言語	変換規則２
設計項目	変換規則，手順	OOPD文法	C++コード生成規則
内容	OODJで振る舞いはどこまで記述可能かを検証し，OODJ記述をOOPDに変換する規則を確立する．	JSMDLの文法を（実行可能な）設計記述言語 OOPDの文法として見直す．	JSMDLの文法見直しに伴い既存のJSMDLトランスレータを再設計する．

4．プログラム設計記述言語 OOPD の設計

4.1　OOPD の位置付けと特長

プログラム設計記述言語として JSMDL の改訂版 OOPD を定義する．OOPD はプログラム設計記述を目的とし，変換規則２により実行可能なプログラムへ変換できる記述言語とする．本言語は図３に示すように日本語一貫プログラミング言語 OOJ 系列のひとつで，OODJ の次に位置付けることとする．ここで，OOJ（Object-oriented Japanese）は言

語シリーズの総称である．

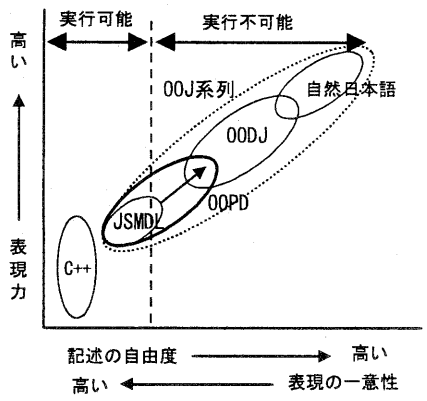

図３　OOPD の位置付け

図３からも分かるように OODJ の単文は自由度が高い代わりに一意性はなく，実行可能プログラムとはならない．OOPD 文法は OODJ 記述を OOPD 記述の第１ステップとし，変換規則１により順次書き直していくものと位置付ける．書き換えていくうちに，一意性の壁を乗り越えて実行可能な OOPD 記述となるように設計する．そして最終的に記述されたステップでの文法は変換規則２により実行可能なプログラムへ（自動的に）変換可能なものとする．この最終ステップの記述を OOPD の文法とする．したがって，OOPD の第１ステップの記述と最終記述ステップの間には中間ステップの記述が存在することを許した文法とする．

４．２　プログラム設計記述言語の要件

プログラム設計記述言語の要件を下記に整理する．

ドメインユーザからの要求

(a) ドメインユーザの主たる関心事は実験結果にあり，新しいプログラム言語を習得する時間はなく，また言語をおぼえる気がしない．したがって，ドメインユーザの情報リテラシー（Fortran 程度は理解し，記述できる）を前提とするが，厳密な文法は避けた記述体系とする．

設計記述に要求される項目

(b) ドメインの対象世界の記述は対象全体の振る舞い記述部分と個々のオブジェクト定義部分の２階層に明確に分かれるケースが多い．したがって，オブジェクトを操作しながら実験の仮説等の記述を行う部分（トップダウン記述）とオブジェクト定義の部分（ボトムアップ記述）とが明確に分かれる体系とする．

(c) 日本語一貫プログラミングのバックボーンであるオブジェクト指向設計（クラス，クラス継承，メッセージ送信等）を実現できるものにする．

(d) プログラム設計を進める場合，外部仕様から入り内部仕様へと進む．したがって，外部仕様記述と内部記述とを分離記述できることは重要である．また，外部仕様を他クラスへの公開情報，内部仕様を非公開情報と位置付けることで情報隠蔽の実現にもなる．

(e) 外部仕様の記述には属性名，振る舞いの外部インタフェースを記述できるようにする．

(f) 内部仕様の記述では実行可能な詳細なアルゴリズム記述ができるようにする．

(g) 内部仕様記述の途中段階にはいくつかの記述ステップを許す．

(h) 設計記述言語であることからプログラム言語に特有な機能は外し，ユーザに詳細文法にとらわれない様にする必要がある．以下の機能は外す．

・実行効率を上げる為の機能：フレンド宣言
・物理的な情報の記述：include，extended
・メモリ構造を意識させる機能：ポインタ型

JSMDL の文法で残すべき項目

(i) プログラム言語へ変換でき文とする（一意性保持）．

５．　記述モデルとＯＯＰＤ文法

５．１　記述モデル：何を記述するか

４．２の要件を実現する記述モデルを提示する．要求を要約すると，

・対象世界の振る舞いと個々の定義の分離記述
・オブジェクト指向設計の記述
・外部仕様と内部仕様の分離記述

・既存のプログラム言語への変換

ができることである．

したがって，記述モデルには要求を満たす下記を組み込む．

記述モデルの基本項目

①クラス定義系と操作系との分離

・クラス定義系はクラスを記述単位とする．

・操作系は実験単位を記述単位とする．

②オブジェクト指向設計

・クラス，クラス継承，メッセージ送信が記述できる．

③外部仕様と内部仕様との分離

・外部仕様は属性と外部に開放する相互作用のインタフェースを記述する．

・内部仕様はアルゴリズムを記述する．

図４はクラス定義系の記述モデルであり，図５は操作系の記述モデルである．記述構成を（ ）で囲み示す．図４の2.2属性におけるアクセス制約は，共有（クラス外へ公開），限定共有（特定のクラスへのみ公開）および私有（自クラス内のみ）の３種類を設ける．同様に2.3動作のアクセス制約としては，クラス外公開，限定公開およびクラス内部のみ有効の３種類を設ける．

図４の3.2(3)動作本体と図５の2.2操作本体の文はメッセージ送信文等である．

```
1．クラス名
2．外部仕様
2.1 クラス間相互関係（クラス継承指定）
      記述構成（継承クラス名リスト）
2.2 属性
      記述構成（属性名，アクセス制約，型）
      のリスト
2.3 動作
      記述構成（アクセス制約，動作名，引数
            リスト，戻り値の型）のリスト
3．内部仕様
3.1 属性の詳細定義
      記述構成（属性名，アクセス制約，型，
            属性値）
3.2 動作の詳細定義
   (1)動作インタフェース定義
      記述構成（アクセス制約，動作名，引数
            リスト，戻り値の型）
   (2)一時変数：この動作内のみ有効
      記述構成　（変数名，データ型）
   (3)動作本体
      記述構成　（文）のリスト
```

図４　記述モデル(クラス定義系)

```
1．操作名
2．操作の手順
2.1　一時変数
      記述構成　（変数名，型）
      この操作系内のみ有効
2.2　操作本体
      記述構成　　（文）のリスト
```

図５　記述モデル(操作系)

５．２　OOPDの文法：どう記述するか

図６，図７，図８に５.１の記述モデルに基づいた OOPD の構文を示す．構文の意味するところを以下に纏める．なお，図６，図７，図８の中で定義していない非終端記号は一般の常識に従うものとする．

①情報隠蔽と外部仕様：

モデル記述ではひとつのクラスの中を外部仕様と内部仕様に分離することで，情報隠蔽を実現している．

②属性：

属性の記述はモデル記述，OOPD 共に外部仕様では型のみを記述し，属性値は内部仕様に記述することとした．型は基本的なデータ型に限定する．

③動作の仕様：

動作のインタフェースは外部仕様に記述し，内部のアルゴリズムは内部仕様に記述する．

④メッセージ送信

動作の記述はメッセージ送信を基本とする．OOPD 文法ではメッセージ式

＜項＞ ← ＜メッセージ＞

で実現する．メッセージ送信のイメージを明確にするために，記号”←”を用いる．

⑤制御文

制御文は繰り返しのための loop 文，選択の if 文，多分岐の cond 文を設けた．

⑥アクセス制約

属性名や動作名を他クラスへ公開するか否かのアクセス制約についてはドメインユーザのプログラム言語知識の前提から考えて，言語仕様には入れないこととした。

```
<外部仕様>          ::= "%外部仕様"<クラス外部仕様>
<クラス外部仕様>    ::= "%クラス"<クラス名>
                        <クラス間相互関係>
                      [{<属性外部仕様>}][{<動作外部仕様>}]
<クラス間相互関係>::= "%クラス間相互関係"
                        [<クラス特性>][<関連クラス>]
<クラス特性>        ::= <任意の文字列>
<関連クラス>        ::= <汎化クラス> |
                        <集約クラス> | {<関係クラス>}
<汎化クラス>        ::= "is_a"<上位クラス名>
<集約クラス>        ::= "have"<集約クラス名>
<関係クラス>        ::= "assoc"<関係クラス名>
                                <関係の名前>
<属性外部仕様>      ::= "%属性"<属性名><属性の型>
<動作外部仕様>      ::= "%動作"<動作定義>
<動作定義>          ::= <動作名>{<引数>}<戻り値の型>
<引数>              ::= <引数名><引数の型>
<戻り値の型>        ::= <属性の型>
```

図６　OOPD 構文（定義系：外部仕様）

```
<内部仕様>          ::= "%内部仕様"<クラス内部仕様>
<クラス内部仕様> ::= "%クラス"<クラス名>
                    [{<属性内部仕様>}][{<動作内部仕様>}]
<属性内部仕様>      ::= "%属性"<属性名>
                        <アクセス制限><属性の型><属性値>

<動作内部仕様>      ::= "%動作"<動作名>{<メソッド>}
<メソッド>          ::= <文の並び>
<文の並び>          ::= {<文>". "} "return"<文>". "
<文>                ::= <メッセージ文> | <loop文> |
                        <代入文> | <if文> | <cond文> |
<メッセージ文>      ::= <メッセージ式>
<loop文>            ::= "loop"<条件>
                        "{" <文の並び> "}"
<if文>              ::= "if"<条件>"then"
                              "{"<文の並び>"}"
                          "else"  "{"<文の並び>"}"
                        "endif"
<cond文>            ::= "cond"<条件 > "{"
                          {"case"<定数式>
                              "{"<文の並び>"}"}
                            "default"
                              "{"<文の並び>"}"
                        "}"
<条件>              ::= "("<項> | <式>")"
<代入文>            ::= <変数>": ="<式>
<式>                ::= <メッセージ式> |
                        <２項式> | <単項式>
<メッセージ式>      ::= <項>"←"<メッセージ>
<２項式>            ::= <項><２項演算子><項>
<単項式>            ::= <単項演算子><項>
<メッセージ>        ::= <メッセージ名>["("{<項>", "}")"]
<項>                ::= <基本項> | "("<式>")"
<基本項>            ::= <属性名> | <リテラル>
```

図７　OOPD 構文（定義系：内部仕様）

```
<操作>      ::= "%操作"<操作名>
                  [<一時変数>]<操作仕様>
<一時変数>::= "%一時変数"<一時変数名>
                        <属性の型><属性値>
<操作仕様>::= "%動作"{<メソッド>}
<メソッド>::= <文の並び>
<文の並び>::= {<文>". "}
              "stop. "
```

図８　OOPD 構文（操作系）

５．３　JSMDL 文法からの改訂点

OOPD 文法の JSMDL 文法からの改訂点は，プログラム設計記述言語としての位置付けから，設計仕様の要件であるクラス定義と操作系記述の分離と外部仕様と内部仕様記述の分離を取り込んだ点である．

一方，オブジェクト指向設計記述と動作記述（文）については一部細かい点の変更はあるが，大筋は JSMDL を継承した．これはこの部分に関しては JSMDL が既に日本語プログラミングとして実現しているためである．

６．　OODJ から OOPD への変換の課題と解決

本章では OOPD の動作記述（文）に焦点を当て，OODJ から OOPD への変換規則１（３．２および表１）の課題とその解決策について述べる．

なお，変換規則２は OOPD から C++への変換，すなわち C++コード生成規則である．これは従来の JSMDL トランスレータ[1)]を改造することで実現するので，詳細検討は省略する．

６．１　変換のプロセス分析と課題

OODJ 文法の日本語単文はプログラム化するための情報が不足しており，またアルゴリズムとしても未完の文である．この日本語単文から OOPD 表記への変換プロセスは次の２ステップに要約される．

(a)**単文を品詞分解し，メッセージ形式へ書き直す．**

(b)**クラス名等の特定を行い，ほぼ OOPD 表記に近い形にする．**

しかし，この状態ではデータ型等の詳細はまだ設定されておらず，トランスレータへの入力ができない．

そこで，

(c)**データ型等の詳細を補って正確な OOPD 表記とする．**

ステップの追加が必要である．

さて，この変換ステップは日本語単文の品詞分解からスタートするが，ドメインユーザにとって

実際には一般的な文章をオブジェクト指向のメッセージを意識して分解し，メッセージ形式に書き直すのは簡単でない．そこで我々は，上記(a)，(b)，(c)のステップの前にあらかじめ日本語単文の段階でオブジェクト指向を意識した文章へまず変換し(記述し直す)，その上でメッセージ形式へ書き直す方法を検討することとした．

以下，OODJ記述にさかのぼって日本語単文段階でのオブジェクト指向記述への書き直しについて詳細に検討する．

6．2　OODJ記述とその記述限界

（1）記述限界

OODJ記述では文献[2]から論理的には矛盾がなく，構文としても平述文と簡易制御文からなり，記述の自由度に制限をあまり加えることなくシステムの分析記述が行える．しかし，自由度はまだ大きく，オブジェクト指向の基本文型であるメッセージ送信形式への変換はユーザの記述能力に左右される．特にプログラムロジックに不慣れなユーザにとっては単文記述からメッセージ送信形式へ書き直すのはなかなか面倒な作業である．

（2）パターン化の導入

そこでOODJの自由度の高い記述にオブジェクト指向に基づいた記述制限を加えることとした．このことにより，オブジェクト指向に不慣れなユーザを特定の記述の型に誘導し，ユーザが一定水準の記述ができるようにする．方法としてはOODJ単文に記述パターンを導入し，この様に書けばメッセージ送信が容易に書けるというガイドを行う．このことでOOPDへの変換を容易にすることを狙う．

OODJの段階でのオブジェクト相互関係を日本語で表現する典型は次の5パターンが考えられる．5パターンの内(A)と(B)はオブジェクト間の静的な構造関係を表しており，相互関係で既に表現されている．一方，(C)，(D)，(E)は動的な相互関係を表しており，メッセージ送信で記述すべきものである．

(A) **Haveパターン：**
　オブジェクト間の全体・部分の関係を表す．
　××は○○の部分である．
(B) **Is-aパターン：**
　オブジェクト間の上位・下位の関係を表す．
　××は○○である．
(C) **Giveパターン：**
　対象のオブジェクトに何かを与える．
　一般にオブジェクトからの戻り値はない．
　××へ△△を○○する．
(D) **Takeパターン：**
　対象のオブジェクトから何かを取り出す．
　メッセージに対しオブジェクトからの戻り値が存在する．
　××から△△を○○する．
(E) **状態変化パターン：**
　対象となるオブジェクトの内部状態を変化させる．
　××を○○する．

図9記述のパターン化

6．3　変換規則1の定式化

OODJ記述からOOPD記述への変換規則1（表1）は6.1，6.2から図10のようにまとめられる．

（1）パターン化　(6.2)
（2）メッセージ化 (6.1の(a))
（3）プレステートメント化 (6.1の(b))

図10　変換規則1

（1）は次のメッセージ化への橋渡しとして日本語単文上でオブジェクト指向記述を意識した文章に書きなおすステップである．

（2）はパターン化した文章をメッセージ送信形式に書きなおすステップである．記述形式としては　　オブジェクト　←　メッセージ

とする．引数無しの表現か引数付きの表現かは出現箇所，内容によって決まる．

（3）はOOPDの文法に正確に従う記述へ移る前に，ユーザの自己流でOOPDに近い文（プレステートメントと呼ぶ）を記述するステップで，新たに設ける．このステップはOOPDの正確な文法にとらわれずに，論理的に手順をまず明確にすることを優先させるために設けた．論理的手順が明確になればプレステートメント化は終了する．

変換規則1，OOPD記述，変換規則2を用いてOODJ文書をC++プログラムへ変換する作業手順を図11に示す．

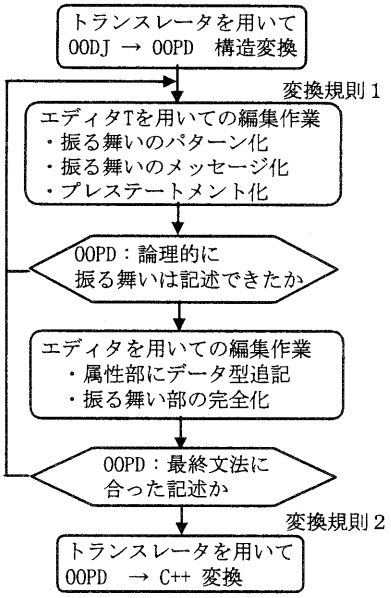

図１１　変換手順

7．記述サンプル

図１２にOOPDによる記述サンプルを示す．熱帯魚飼育のシミュレーションプログラムで，熱帯魚クラスの振る舞い（内部仕様）について記述した部分である．

```
%内部仕様
%クラス　　熱帯魚
%属性　　色　私有　カラー型　＝赤
%属性　　体長　私有　長さ　＝５ｃｍ
%属性　　生存状態　　私有　　状態型　＝０
                    （０：生存、１：死亡）
%属性　　年令　　私有　　整数型　　＝０
%属性　　餌をもらった時刻　　私有　時刻型　＝０
%動作　　表示する　　公開
    熱帯魚を表示する．
%動作　　生きている　　公開
    餌をもらった時刻を記憶し、２日経過しても
        餌がもらえなかったら死ぬ．
    温度が標準より一定時間はずれると死ぬ．
                    （条件を考える必要あり）．
%動作　　加齢する　　公開
    熱帯魚は一定時刻毎に加齢する．
    ２００日経過すると死ぬ．
%動作　　餌を与える　　公開
    餌をもらった時刻を記憶する．
%動作　　移動する　　公開
    蛍光灯が点いているときは動き消えると止まる．
    水槽にぶつかる寸前に反転する．
%動作　　表示しない　　公開
    形状を消す．
```

図１２　OOPD記述サンプル

8．結論と今後の課題

本論文ではプログラム設計記述言語 OOPD の設計について述べた．OOPDは自然言語の能力を保持する高可読性の記述言語であると同時に，実行可能な記述を実現した記述言語である．この実現のために，従来の日本語一貫プログラミングで用いたプログラム言語 JSMDL の改訂，分析設計を行うための記述言語 OODJ からこの OOPD への変換規則の定式化を行った．

JSMDLからOOPDへの改訂では記述モデルの明確化とモデルに基づく文法定義を行った．文法をプログラム記述言語からプログラム設計言語へと上流工程へシフトし，実行効率等を一部犠牲にし，可読性の向上と学習容易性を中心にする文法とした．また，変換規則は変換プロセスを解析し，そこにパターン化，メッセージ化，プレステートメント化の手順を見いだし定式化することができた．

現在，図１１を実現する OOPD の支援環境を実装中である．

参考文献

１）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，情報処理学会論文誌，Vol.40，No.7，pp.3016-3030(1999)．

２）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，1998年3月10日，98－SE－118，pp.15-22

３）畠山，加藤木，石井：オブジェクト指向記述日本語 OODJ とその記述環境，情報処理学会論文誌，Vol.41，No.9，pp.2567-2582(2000)．

４）Bartrand Meyer，"Object-Oriented Software Construction"，Prentice-Hall (1989)（邦訳「オブジェクト指向入門」二木監訳，酒匂訳，アスキー）．