社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－SE－150 (10)
2005／11／29

オブジェクト指向実装記述言語 OEDJ の記述環境およびトランスレータの開発

加藤木和夫* 畠山正行**

*加藤木技術士事務所 **茨城大学工学部情報工学科

本研究では，既に発表したオブジェクト指向実装記述言語 OEDJ のための記述環境とトランスレータを開発した報告を行う．OEDJ はオブジェクト指向一貫記述言語 OOJ の実装段階に位置し，前段階の設計記述言語 ODDJ 記述から ODDJ/OEDJ トランスレータを使って自動変換される．DU は記述環境を使って OEDJ 記述を編集し，プログラム風の OEDJ 記述を完成させる．完成した OEDJ 記述は OEDJ/Java トランスレータを用いて Java プログラムに変換される．現在，開発は殆ど出来上がっている．記述環境付きの OEDJ は幾つかの対象分野に適用されつつあり，様々な記述テストが続けられており，最終的な評価も行われる．

A Development of Description Environment and Translator for Object-oriented Executable Description Japanese OEDJ

KAZUO KATOUGI* MASAYUKI HATAKEYAMA**

*Katougi Technical Office **Ibaraki University

We have developed the description environment and the translator of the Object-oriented (hereafter, OO) programming description language OEDJ. The OEDJ is seated at the programming stage of the OO integrated description language system OOJ. The design stage ODDJ descriptions are translated into the OEDJ descriptions by using the ODDJ/OEDJ translators. The Domain User (DU) edits the OEDJ descriptions using the description environment, and completes the program-like image descriptions of the OEDJ. The completed OEDJ descriptions are transformed into the Java program by using the OEDJ/Java translators. At present, the development has almost been completed. The OEDJ with the description environment is now being applied to some target domains, the various description tests are also being continued, and finally being evaluated.

1．はじめに

ドメインユーザ(Domain user:以降，DU と略)向けにオブジェクト指向(以降，OO と略)に基づく分析／設計／実装(プログラム化)技術の開発を進めている．DU は専門領域の研究が主たる対象で，研究支援のためのソフトウェア環境を整備する時間的な余裕がない．筆者らは DU 向けにソフトウェア開発分野で有効性が確認されている OO 技術を役立てるために，自然日本語(以降，NJ)記述を取り入れるなど DU に受け入れられ易い形の支援環境および記述言語の開発を進めてきた．その中でオブジェクト指向一貫記述言語の構想が固まり[1),2),3)]，NJ 記述の基本的な記述構造[2)]，分析／設計／実装[4),5),6),7)]の個別言語仕様の研究がまとまりつつある．

そうした研究開発の一環として，本論文は実装段階に関するもので，分析／設計段階で記述された設計文書 ODDJ (Object-oriented Design Description Japanese)[5)]をどのようにしてプログラムとして実装する

かについて述べる. ODDJ 文書では設計に関する情報は盛り込まれているが, このままの形を計算機上で実行するためには詳細な情報が不足している. 実行するためには ODDJ 文書に情報を追記あるいは修正することが必要であり, また既存のプログラミング言語(以降, PL と略)へ変換する必要がある.

そこで筆者らは先にプログラム記述言語 **OEDJ** (Object-oriented Executable Description Japanese) を導入することで情報を追記し易くし, かつ ODDJ を円滑に既存オブジェクト指向 PL(以降, OOPL と略)へ変換する方式を提案した[7]。本論文では OEDJ 言語仕様とその記述環境およびトランスレータについて述べる.

2. 開発方針の検討

2. 1 開発方式の検討

設計記述文書 ODDJ へ情報を追記し, PL へ変換する場合に設計記述とプログラム記述をどう位置づけるかにより次の 2 つの案が考えられる.

(1)記述兼務案

ODDJ が設計記述とプログラム記述言語を兼ねるとする案である. 本方式での開発はオブジェクトベース Fortran として進められおり[6], 変換先言語として Fortran90 が選択されている.

(2)記述分離案

設計記述言語とプログラム記述言語を明確に区別する案で, 本論文で提案するものである. 本案ではプログラム記述言語 OEDJ を導入する.

各案の長所, 短所については 6 章の評価で述べる.

2. 2 OEDJ 言語仕様の要件

上記(2)の方式で開発するに当たり, OEDJ の言語仕様には次のことが要求される.

(a)学習容易性

新たな言語を導入するということから, 学習の容易性が重要になる. 特に今回の対象となる DU の多くは手続き型 PL には充分な知識はあっても, オブジェクト指向 PL の知識はほとんどないのが現状である. そこで, DU の学習時間を最小限にするためには手続き型 PL 風の記述規則(文法)とする必要がある. Fortran でプログラムが書けるが, OOPL の文法を知らない DU が読めて修正ができるものでなければならない.

(b)OODJ 文書との関連

OEDJ は OODJ から自動的に変換できなければならない. これは DU の学習負担を軽減するためにも必須である.

(c)既存 OOPL との関係

OEDJ は既存 OOPL へ変換可能でなければならない. 一般の OOPL と同様に, あいまい性のない厳密な文法が要求される.

(d)特定 OOPL 非依存

現時点で特定の OOPL に変換するとしても, 将来, DU の既存財産を生かすために変換先の OOPL を変更することが考えられる. その場合, 変換先の OOPL が変わるごとに OEDJ の言語仕様が変わることは避けたい. したがって, OEDJ は特定の OOPL に依存しない言語として設計する必要がある.

2. 3 記述環境とトランスレータ開発の要件

(a)OEDJ の提示と編集

ODDJ 文書から OEDJ 記述への変換は自動的に行い, 結果を DU へ提示する. DU は提示された OEDJ 記述上で追記・修正を行い, 動作の確認もこの上で行う. OOPL へ変換ができない部分は DU が情報を追記する必要があり, その部分については DU に対して入力要求や警告メッセージなどを出す必要がある.

(b)トランスレータ

ODDJ 文書は XML 形式で保存されている[5]. トランスレータは XML 形式の ODDJ 文書を OEDJ 文書へ変換し, DU に提示する. その後で DU によって情報の追記・修正の行われた OEDJ 文書を OOPL へと変換する. したがって, トランスレータはこれら 2 個を設計・製作する必要がある. OEDJ 記述の格納は将来, XML 形式で

ない文書を受け入れる可能性もあるので，この時点で構造化テキスト(OEDJ 記述そのまま)の形式で保存することとする.

(c)変換先の OOPL

本開発ではコンパイラおよび開発環境の入手しやすく，また移植性が高いJavaとする. ただし，2.2節(d)にも挙げたようにDUが変換先OOPLを意識しなくともすむようにする.

3. 支援環境の設計

支援環境の全体はOEDJへの要件から下記とする.

(1)ODDJ文書をPLへ変換する中間で，OEDJを出力し，DUに提示する(トランスレータA:ODDJ/OEDJ).

(2)DUは情報の追記，修正を提示されたOEDJ記述に対して行う(OEDJ記述エディタ).

(3)DU によって修正・追記された OEDJ 記述は既存PLへ変換する(トランスレータB:OEDJ/Java).

図1にOEDJ記述環境に基づき，DUが追加・修正をどのように行うかのプロセスを示す. ODDJ 記述が実行されるプロセスは

① ODDJ記述がOEDJ記述環境に入力される.

②トランスレータ A(ODDJ/OEDJ)により，ODDJ 記述がOEDJ記述に変換される.

③DUは提示されたOEDJ記述に対して追記・修正を行う.

④OEDJ記述が修正される.

⑤追記・修正が行われた OEDJ 記述はトランスレータB(OEDJ/Java)により Java プログラムに変換され，実行される.

4 プログラム記述言語 OEDJ の設計

OEDJ は ODDJ 記述を自動変換し，DU に提示するプログラム記述言語であり，下流の Java へ変換可能な言語でもあるので，両方を考慮しての言語仕様となる.

4.1 OEDJ の設計方針

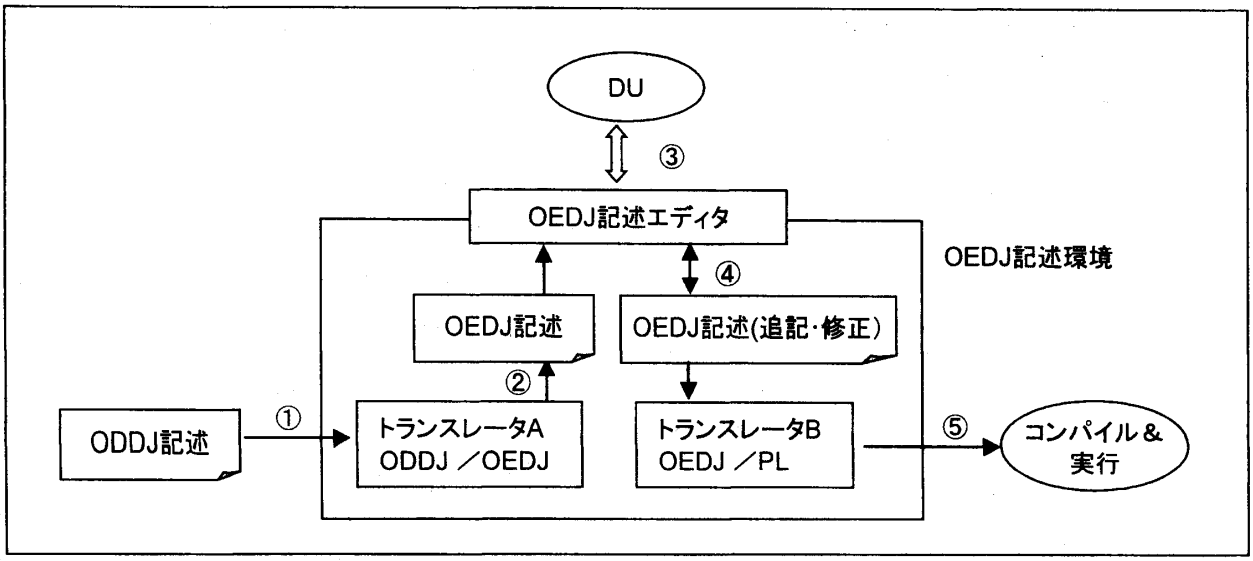
(1)上流言語 ODDJ からの要求

ODDJ 記述文書は以下の特徴を持つ.

・文書の構造は OOSF 構造[2)]

・内容の記述は NJ 記述(複雑な式を含む.)

OEDJ 記述はこのうち NJ 記述部分を引き継ぎ，高い可読性を継続する. 一方，OOSF 構造については OEDJ



→:データの流れ，1回発生する. ◀▶:データの流れ，複数回発生する. ⟺:DUの介入，複数回発生する.

図1 「DUの記述」の追加・修正と変換のプロセス

記述としてプログラムのイメージに手続き型 PL を採用するという観点から，OOSF 構造を引き継がず，簡易なプログラム構造を新たに設計する．

(2) 下流言語からの要求

変換先 OOPL を Java 以外にしたいという DU の要望が出た場合に，その度に OEDJ の言語仕様を変えることがないようにするために，OEDJ は特定 OOPL 非依存とする．言い換えれば，Java でなければ実現できない機能は排除する．

(3) 学習容易性と手続き型 PL 風

DU に学習の負担をかけないように，(1)とも関連するが，手続き型 PL 風の記述形式とする．

手続き型PLのイメージを前面に出すために，プログラムの構造を簡易にし，OOPL の言葉は使わないこととする．

4．2 OEDJ の特徴

記述規則の全体は本稿末の**表 1** に示す．ここではその中から抜粋して主な項目について以下に述べる．記述規則の定義記号については同じく本稿末の**表 2** に示す．

(1) キーワードは手続き型 PL 風

表 3 に OEDJ 記述の全体構造を示す．

表 3　OEDJ 記述の全体構造

```
1.　OO構造と駆動記述
1．1　OEDJ記述
OEDJプログラム::= プログラム +　シナリオプログラム
1．2　プログラム記述
プログラム::= "%プログラム" 《プログラム名》
      ("%プログラム間相互関係"　関連プログラム )*
      ("%変数"　変数記述 |"%定数"　定数記述 )*
      ("%処理"　処理記述 |"%関数"　関数記述　)*
      "%クラス終了"
関連プログラム ::= 汎化 | 集約 | 一般関連
```

キーワードは手続き型 PL 風にするために OOPL で使用するクラス，メソッド等は用いず，<プログラム>，<変数>，<処理>，<関数>等とした．

(2) プログラム構造は簡易

表 3 に示すように OEDJ 記述は複数の<プログラム>(処理を記述)とひとつの<シナリオプログラム>(主プログラムに相当)から構成する．ひとつの<プログラム>は処理を記述する枠組みで，情報隠蔽を実現する．

(3) 継承ではなく処理呼出し

<プログラム間>の継承は許さない．これは OOPL が持つ継承の理解しにくさを排除するためである．OEDJ 記述では処理呼出しという手段のみで処理の共有を実現した．

(4) データ型

表 4 に変数とデータ型の定義を示す．

表 4　変数とデータ型

```
2．1　変数
変数記述::＝基本データ型変数記述　|
          抽象データ型変数記述
基本データ型変数記述::=《変数名》("[" 要素数 "]")?
          ":"　基本データ型
                ("=" 初期値 (", " 初期値 )*)?
抽象データ型変数記述::=《変数名》("[" 要素数 "]")?
          ":" 抽象データ型 ( "=" 初期化引数 )?
定数記述::=《定数名》":"　基本データ型　"=" 定数値
2．2　データ型
データ型 ::= 基本データ型 | 抽象データ型
基本データ型::= 算術型 | 論理型 | 文字型
抽象データ型::=《プログラム名》
```

変数のデータ型は整数型，実数型などの<基本データ型>と共に，<プログラム名>(OO のクラス名に相当)を許す．この点は DU にはなじみにくいが，<プログラム名>は段階的な記述を実現するためのアブストラクタ・データ型(ADT)として導入する．本機能は ODDJ 記述には現れないが，DU が OEDJ で標準ライブラリを使用するときに用いることになる．

(5) call 文

表 5 に<call>文の構文を示す．<call>文はオブジェクト指向の動作の基本であるメッセージ送信を DU に分かりやすく理解してもらうために，概念は異なるが類推し易いということで導入した．

表 5　call 文

```
2．3　文
call文::= "call"　参照
参照::= プログラム参照 | ライブラリ参照
プログラム参照::= 《プログラム名》(配列要素)? " . "
          《処理名》 実引数
ライブラリ参照::= 《ライブラリ名》(配列要素)? " . "
          《処理名》実引数
実引数::= "(" ( 式 ("," 式 )*)? ")"
```

記述形式は

call プログラム名．処理名(実引数)

として Fortran のサブルーチン呼び出しに近い形とする．ただし，<プログラム>に含まれる処理のひとつを指定する必要があるので，<プログラム名>と<処理名>をドット(．)で結ぶ形とした．

(6) マクロ

バッファ処理などはマクロ表現(NJ 表現)を用意する．トランスレータはこの表現を解釈して Java の該当クラスにアクセスする文に変換する．

(7) クラス／インスタンスの取り扱い

DU は対象世界の分析／設計において OOPL におけるクラスやインスタンスを意識しない．ODDJ の OOSF 構造(フレーム構造)はオブジェクトベースの考え方[1]に基づき構成されており，クラス／インスタンスの概念は導入していない．DU は「フレーム」を意識すれば良い．一方，OEDJ は普及した OOPL への変換を考慮してクラスに相当する<プログラム>という記述枠組みを備えている．しかし，この差異は問題ではなくフレーム構造を<プログラム>構造へ写像すれば解決する．

5. OEDJ 記述環境とトランスレータ

5．1 設計方針

OEDJ の記述環境(エディタ)，トランスレータAおよびトランスレータ B(**図 1**)の設計方針を下記のように設定した．

(1)記述環境(エディタ)

・記述ガイドの充実

(<プログラム>の構成や各文の構文を憶えていなくともガイドにより記述できるなど)

・適切な警告・エラーメッセージの出力

・ODDJ と OEDJ の対応の明瞭化

(2)トランスレータA(ODDJ を OEDJ へ変換する)

・ODDJ からの OEDJ 自動出力

(OEDJ はこの時点では不完全でも可とする)

・適切な警告・エラーメッセージの出力

(3)トランスレータB(OEDJ を Java へ変換する)

・ODDJ の正確な実現(例:オブジェクトベース実現)

・Java プログラムとしての完全性

5．2 記述環境の画面例

図 2 に OEDJ 記述のための記述ガイドの例を示す．本例は「新規作成」の場合である。「新規作成」項目が選択されると，<プログラム>の基本構造が表示される。個々の文はポップアップメニューを表示し、各文のメニュー項目を選択することで各文の構文を指定の箇所に埋め込み，DU はそこに各文の情報を追記する．

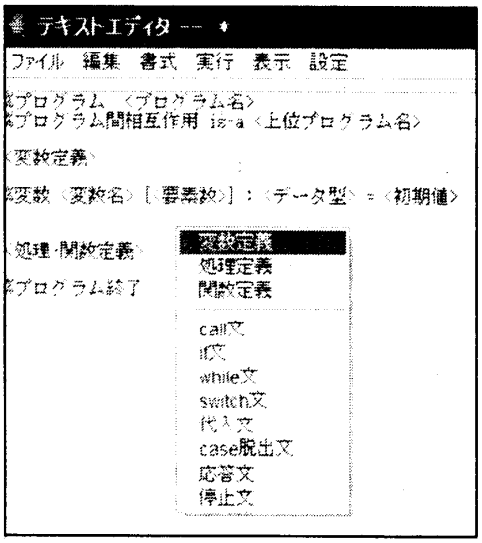


図 2 OEDJ 記述エディタの画面

5．3 記述例

本稿 8 ページの**表 6** に OEDJ 記述および変換後の Java プログラムを示す．また，同一内容の ODDJ 記述は参考文献 3)，オブジェクトベース Fortran の記述は参考文献 4)に掲載してある．

OEDJ 記述は ODDJ 記述の NJ 記述を受け継いでいるため，構造がフレーム構造からプレーンテキストに変換されていても対応が分かり易いことが見て取れる．

6. OEDJ の評価

(1) ODDJ との記述落差の検討

OEDJ 記述で修正・追記された情報は

・if文の判定条件の論理式

・数式ライブラリの呼出し(べき乗, SIN関数)

・入出力ファイル操作の詳細記述

である. いずれもプログラム記述に要求される厳密性の修正である. 設計を行っている段階では後回しにしたい部分であり, 設計記述とプログラム記述とに記述が分離されていることにより, 設計段階では動作の詳細なことまで考えず, 動作の正しいアルゴリズムの記述に専念できる.

(2) 学習容易性の評価

OEDJ は大部分が ODDJ から自動変換で出力されること, 個々の文の構文はガイドで記述を誘導してくれること, 手続き型PLから類推し易いキーワードが用いられていることなどで理解しやすい.

(3) 記述兼務か分離かの評価

設計記述とプログラム記述を兼務する長所は DU が新たにプログラム記述段階の言語を習得しなくともすむ点である. 短所は設計記述言語という位置づけがあいまいになり, 設計記述にプログラム記述要素が混入する点である.

一方, 分離案の長所は段階的記述が可能であること, プログラムとしての実行順序が明確であるので修正時などにアルゴリズムの追跡がし易い点が上げられる. 短所は新たな記述言語の習得が必要になる点である. しかし, 本論文で提示したようにプログラム記述の自動出力と手続き型 PL 風にすることでかなりこの問題を解消することができた. 総合的な評価は今後更に検討を進めたい.

(4) 方式としての評価

今回開発した OEDJ 記述環境は一般化して言えば, 設計記述言語を入力とし, 既存の PL を出力する. 中間段階で汎用的なプログラム記述をユーザに提示し, 情報の追記・修正を行う方式, と捉えることができる. したがって, 将来の拡張としてトランスレータAとBに対して異なった開発することでユーザに慣れ親しんだ開発環境に対応できる記述環境を構築できる.

7. 結論と今後の課題

設計記述言語 ODDJ の次の実装段階としてプログラム記述言語OEDJとその記述環境およびJavaプログラム出力するトランスレータを開発した. DU は設計記述 ODDJ から Java へ手書き変換することなく半自動的に変換・実行を行えるようになった. 今後は更に記述評価を進め, DU が使い易いように記述環境の改良を行っていきたい.

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, 21-4, pp.295-309, Dec., (2003).

2)畠山正行, オブジェクト指向自然日本語構造化フレーム OOSF の設計と表現技法, シミュレーション学会誌, 195/209, Dec., (2004).

3)畠山正行, オブジェクト指向一貫記述言語 OOJ の構成とその概念設計, 第 150 回 SE 研究会報告, 2005-SE-150, pp.23-26, (2005).

4)畠山正行, オブジェクト指向自然日本語記述言語 OONJ の設計とその記述例, 第145回SE研究会報告, 2004-SE-145, pp.53-60, (2004).

5)川澄成章, 畠山正行, 野口和義, オブジェクト指向設計記述言語 ODDJ の設計とその記述環境の開発, 第150回SE研究会報告, 2005-SE-150, pp.23-26, (2005).

6)齋藤正樹, 畠山正行, オブジェクトベース Fortran の設計とそのトランスレータの開発, 第 150 回 SE 研究会報告, 2005-SE-150, pp.23-26,

7)加藤木和夫, 畠山正行, 竹井健太朗, オブジェクト指向実行記述言語 OEDJ とそのﾄﾗﾝｽﾚｰﾀ, 第 145 回 SE 研究会報告,2004-SE-145,pp.65-72, (2005).

表1 OEDJ 記述規則

下記に OEDJ の記述規則を示す. 記述規則の定義記号は**表 2** に示す.

```
1. OO構造と駆動記述
1.1 OEDJ記述
OEDJプログラム::= プログラム + シナリオプログラム
1.2 プログラム記述
プログラム::= "%プログラム"《プログラム名》
    ("%プログラム間相互関係" 関連プログラム )*
    ("%変数" 変数記述 |"%定数" 定数記述 )*
    ("%処理" 処理記述 |"%関数" 関数記述 )*
    "%クラス終了"
関連プログラム ::= 汎化 | 集約 | 一般関連
汎化::= "is-a"《上位プログラム名》
集約::= "has-a"《被集約プログラム名》
一般関連::= "assoc"《相手先プログラム名》
1.3 シナリオプログラム
シナリオプログラム::="%シナリオプログラム"《シナリオ名》
    ("%プログラム間相互関係" 関連プログラム )*
    ("%設定" 開始状況記述 )*
    ("%処理" 処理記述 |"%関数" 関数記述 )*
    "%シナリオ" シナリオ記述 "%シナリオ終了"
開始状況記述 ::= ("%変数" 変数記述 |"%定数" 定数記述 )*
シナリオ記述::= ("%変数" 変数記述 |"%定数" 定数記述 )*
    ( 処理記述文 ";" ) +
```

```
2. 変数と処理
2.1 変数
変数記述::= 基本データ型変数記述 | 抽象データ型変数記述
基本データ型変数記述::=《変数名》("[" 要素数 "]")?
            ":" 基本データ型
                ("=" 初期値 (", " 初期値 )* )?
要素数 ::= 整数値
初期値::= リテラル |《定数名》
抽象データ型変数記述::=《変数名》("[" 要素数 "]")?
            ":" 抽象データ型 ( "=" 初期化引数 )?
定数記述::=《定数名》":" 基本データ型 "=" 定数値
定数値::= リテラル
初期化引数::= 基本項
2.2 データ型
データ型 ::= 基本データ型 |抽象データ型
基本データ型::= 算術型 | 論理型 | 文字型
抽象データ型::=《プログラム名》
2.3 処理
処理記述::=《処理名》 仮引数定義 処理制約?
    ("%変数" 変数記述 |"%定数" 定数記述 )*
    ( 記述文 ";" )+ // 本体
関数記述::=《関数名》 仮引数定義
        ":" 戻り値のデータ型 処理制約?
    ("%変数" 変数記述 |"%定数" 定数記述 )*
    ( 記述文 ";" )+ // 本体
仮引数定義::= "(" ( 仮引数 ("," 仮引数 )*)? ")"
仮引数::=《仮引数名》("[]")? ":" 仮引数データ型
仮引数データ型::= データ型
戻り値のデータ型::= データ型
処理制約 ::= "アクセス制限:" ("私有" | "共有")
```

```
2.3 文
記述文 ::= call文 | 制御文 | 代入文 | case脱出文 | 応答文 |
        停止文 | 空文
call文::= "call" 参照
参照::= プログラム参照 | ライブラリ参照
プログラム参照::= 《プログラム名》(配列要素)? " . "
            《処理名》 実引数
ライブラリ参照::= 《ライブラリ名》(配列要素)? " . "
            《処理名》実引数
実引数::= "(" ( 式 ("," 式 )*)? ")"
制御文 ::= if文 | switch文 | while文 | for文
if文 ::= "if" if条件 "{" ( 処理記述文 ";" )+ "}"
    ("else if" if条件 "{" ( 処理記述文 ";" )+ "}" )*
    ("else" "{" ( 処理記述文 ";" )+ "}" )?
switch文 ::= "switch" switch条件 "{"
    ( "case" case名 ":" ( 処理記述文 ";" )+ )+
    ( "default:" ( 処理記述文 ". " )? "}"
if条件 ::= "(" 式 ")"
switch条件::= "(" 式 ")"
case名::= 整数値 | 文字値
while文::= "while" 繰返し条件式 "{" ( 処理記述文 ";" )+ "}"
繰返し条件式::= "(" 式 ")"
for文::= "for" "(" 初期化式 ";" 繰り返し条件式 ";" 更新式 ")"
        "{" ( 処理記述文 ". " )+ "}"
初期化式 ::= 式
更新式::= 式
代入文::= 変数 "=" 式
case脱出文::= "break"
応答文::="応答:" リテラル | 変数
停止文::="停止:"
空文::= "−"
```

```
式 ::= or式
or式 ::= and式 | or式 "or" and式
and式 ::= 等式 | and式 "and" 等式
等式::= 関係式 | 等式 "=" 関係式 | 等式 "!=" 関係式
関係式 ::= 加減式 | 関係式 "<" 加減式 | 関係式 ">" 加減式 |
        関係式 "<=" 加減式 | 関係式 ">=" 加減式 |
加減式::= 乗除式 | 加減式 "+" 乗除式 | 加減式 "−" 乗除式
乗除式 ::= 単項式 | 乗除式 "*" 単項式 | 乗除式 "/" 単項式 |
        乗除式 "%" 単項式
単項式::= "-" 単項式 | "not" 単項式 | 基本項
基本項::= リテラル | "(" 式 ")" | 参照式 |
        変数名 (配列要素)? |《定数名》
参照式::= プログラム参照式 | ライブラリ参照式
プログラム参照式::= 《プログラム名》(配列要素)? " . "
        《関数名》実引数
ライブラリ参照式::= 《ライブラリ名》(配列要素)? " . "
        《関数名》実引数
変数名::=《変数名》|《仮引数名》
配列要素::= "[" 要素番号 "]"
要素番号 ::= 式
リテラル ::= 数値 | 論理値 | 文字値 | "null"
```

表2 記述規則定義記号

記号と説明を対にして下記の表に示す.

::=	非終端記号	《 》	終端記号 (任意の文字列)	"a"	終端記号 (aはOEDJの基本記号)	a | b	aまたはb
( )	グループ化	a?	aまたは空	a+	1個以上のa を繰り返す	a*	0個以上のa を繰り返す

表6 記述例

OEDJ の記述例とトランスレータによって Java に変換されたプログラムを示す(Java は一部紙面の都合で略す).

OEDJ記述例	変換後のJavaプログラム

%プログラム 海
%定数 σ : 実数型 = 5.67
%定数 ε : 実数型 = 0.95
 	
%変数 面積 : 実数型 = 70
%変数 比熱 : 実数型 = 0.95
%変数 吸収熱量 : 実数型
%変数 表面温度 : 実数型 = 1.2
%変数 表面質量 : 実数型 = 144.2
%変数 放射熱量 : 実数型
%変数 保有熱量 : 実数型
 	
%処理 熱吸収する (吸収熱量 : 実数型)
保有熱量 = 保有熱量 + 吸収熱量 * 面積 ;
call 海 . 温度が上昇する ( ) ;
 	
%処理 温度が上昇する ( )
表面温度 = 表面温度 + 保有熱量 / (表面質量 * 比熱) ;
call 海 . 蒸発する ( ) ;
 	
%処理 蒸発する ( )
%変数 高度0の温度 : 実数型
%変数 蒸発量 : 実数型
%変数 結果 : 実数型
 	
高度0の温度 = 高度0の海上空大気 . 温度を取得する ( ) ;
蒸発量 = 1.555 * (表面温度 - 高度0の温度) ;
call 高度1の海上空雲 . 水を受ける (蒸発量) ;
call 海 . 熱放射する ( ) ;
 	
%処理 熱放射する ( )
%変数 放射熱量 : 実数型
放射熱量 = ε * σ * (関数 . べき乗 (表面温度 + 0.948 , 4)) ;
call 高度0の海上空大気 . 海から熱吸収する (放射熱量) ;
call 海 . 温度が低下する ( ) ;
 	
%処理 温度が低下する ( )
表面温度 = 表面温度 - 放射熱量 / (表面質量 * 比熱) ;
 	
%処理 海へ流出する ( )
応答: ;
 	
%処理 雨／雪を受ける (雲量 : 実数型)
応答: ;
 	
%処理 大気から熱吸収する (吸収熱量 : 実数型)
保有熱量 = 吸収熱量 * 面積 ;
 	
%プログラム終了
	public class Cls14{ // 海
	//1プログラム1インスタンスのために
	//自分のインスタンスを生成する
	public static final Cls14 instance = new Cls14();
	// 定数
	public static final double attr0= 5.67; // σ
	// 定数
	public static final double attr1= 0.95; // ε
	// 変数・設定
	private double attr2 = 70; // 面積
	// 変数・設定
	private double attr3 = 0.95; // 比熱
	
	[中略]
	
	// 処理・関数
	public void mthd86(double kari0){ // 熱吸収する
	attr8 = attr8 + kari0 * attr2;
	mthd87();
	} // メソッド終了・・・・・
	// 処理・関数
	public void mthd87(){ // 温度が上昇する
	attr5 = attr5 + attr8 / (attr6 * attr3);
	mthd88();
	} // メソッド終了・・・・・
	// 処理・関数
	public void mthd88(){ // 蒸発する
	double tempVar0; // 高度0の温度
	double tempVar1; // 蒸発量
	double tempVar2; // 結果
	tempVar0 = Cls15.instance.mthd99();
	tempVar1 = 1.555 * (attr5 - tempVar0);
	Cls16.instance.mthd100(tempVar1);
	mthd89();
	} // メソッド終了・・・・・
	// 処理・関数
	public void mthd89(){ // 熱放射する
	double tempVar3; // 放射熱量
	tempVar3 = attr1 * attr0 * (Math.pow(attr5 + 0.948,4));
	Cls15.instance.mthd94(tempVar3);
	mthd90();
	} // メソッド終了・・・・・
	// 処理・関数
	public void mthd90(){ // 温度が低下する
	attr5 = attr5 - attr7 / (attr6 * attr3);
	} // メソッド終了・・・・・
	
	[中略]
	
	} // クラス終了・・・・・

(補足)

(1) OEDJ 記述例

本記述は「水の大気循環」の一部で<プログラム名>は「海」であり, 定数 2 個, 変数7個, 処理 8 個を含む.

(2) 変換後の Java プログラム

<プログラム名>は Java では Cls*(*は番号)となる. 同様に変数は attr*, 処理名(関数名)は mthd*, 処理内の変数は tempVar*に変換される. 日本語識別名はコメントとして Java プログラムの該当識別子名の後に記される.