2004－SE－145　(10)
2004／8／20

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

加藤木和夫*　　畠山正行**　　竹井健太朗***

*加藤木技術士事務所　　**茨城大学　　***茨城大学大学院

科学技術分野のドメインユーザ(DU)向けにソフトウェア開発支援環境の構築を進めている. その一環として対象世界の分析を記述する分析記述言語 OONJ(オブジェクト指向自然日本語)を開発しているが, OONJ は記述に自然日本語(NJ)を用いており, プログラミング言語(PL)への自動変換は困難である. 我々はこの困難性を回避するため, NJ 記述とプログラム言語(PL)記述の中間に中間言語 OEDJ(オブジェクト指向実行記述言語)を導入し, 解決を図った. OEDJ は実行性の実現可能を主眼に置いた記述言語である. 言語仕様はPLへ変換可能でなければならないという条件の下, OONJからの記述乖離をできるだけ少なくするよう設計し, 限定された NJ 記述パターン, DU にオブジェクト指向を意識させない1クラス1インスタンスなどを採用した. OEDJ の導入によりオブジェクト指向に基づく NJ 記述を可能にした OONJ から PL(具体的には C++)への変換を円滑に出来る見通しを得た.

Object-Oriented Executable Description Japanese and its Translator

Kazuo Katougi*　Masayuki Hatakeyama**　and Kentaro Takei***

*Katougi Technical Office　　**Ibaraki University

We have designed a new description language OONJ(Object-Oriented Natural Japanese) based on the native mother language (natural Japanese; NJ) for the domain users in science and technology field. But, direct translation from OONJ into PL (Programming Language) is very difficult. Therefore, we have introduced the intermediate language OEDJ (Object-Oriented Executable Description Japanese) between OONJ and PL. We have designed the description gap become a minimum between two language on condition that OEDJ is the executable language. We have adopted the constrained NJ description pattern etc. The implementation of the support environment is not yet, the introduction of the OEDJ is confirmed to be valid.

1. はじめに

科学技術計算・研究分野のドメインユーザ(以下, DU)向けのソフトウェア開発環境が強く望まれているが, 専用環境を除けば適切なものがほとんどない. DU の要望は手軽に対象分野の分析ができ, そのまま計算機上で分析結果を実行できる, あるいはシミュレーション実験結果を早く知りたい等である. また, DUはソフトウェアの様々な知識を習得する時間的余裕がなく, このため出来る限り最小の学習で使える環境がほしい, という強い要望を持っている.

我々は要求実現の方法として, 学習時間の短縮とソフトウェア工学の技術要素を取り入れた環境を実現する手法として, DU が馴染みの深い自然日本語(以下, NJ)による分析とその実行方式および支援環境の研究を進めている[1][2].

我々は既にいくつかの環境構築の試行を重ねてきた[3][4][5]. 当初は日本語で記述した文書をDUが注意深く読み解き, クラス名候補, メソッド名候補を文章の中から切り出してオブジェクト指向プログラムへと書き直し, 実行させる日本語一貫プログラミング環境を開発した[3]. 次いで環境を DU に使い易いように再構築した[4]. 更にオブジェクト指向

設計記述言語を導入し，設計段階での日本語記述によるプログラムを記述可能とした[5]．しかし，いずれの方式においても記述する日本語の文は記号を区切りに用いており，プログラミング言語（以下，PL)に近い記述形式であり，依然としてDUにプログラミング負担をかけている．

そこで現在，DU の負担を大幅に軽減できるように自然日本語(Natural Japanese:以下，NJ)記述を大幅に取り入れたDU向け分析記述言語 **OONJ** (Object-Oriented Natural Japanese)の開発を進めている[1][2]．OONJ は記述のフレームを提供し，分析内容の記述には NJ を用いる．NJ は単文を使用するなど文にいくつかの制限を加えているが，表現力は自然言語とほぼ同等である．また，OONJ はオブジェクト指向を表面に出さず，また，プログラミング言語の形式も取り入れない． このような OONJ は分析に向いているが，それを計算機上で実行するのはそのままでは，NJ の解析とプログラムへの変換という困難な課題がある．

この課題解決のために前述の開発経験[3][4][5]を踏まえ，また NJ 記述による DU 向け環境の集大成として，OONJ を実行できる環境を開発することとした．解決方法としてはNJ記述とPLとの間に中間的な言語を導入し，分析記述から PL への変換を図る環境を設計・構築することとした．すなわち，中間言語を設けることで，NJ 表現を生かしつつ，PL への変換を可能にするという方式である．

本稿では変換する手法として導入した中間言語（以下，**OEDJ**（Object-oriented Executable Description Japanese)と呼ぶ)の言語設計，支援環境の概要について述べる．

2．OEDJ 言語仕様の基本方針

2．1 基本方針

OEDJ の位置づけは NJ 記述を中心とする OONJ 文書を PL(具体的には C++)プログラムへ変換するための中間言語である．**図 1** に中間言語の性格から前後の言語仕様からどのような要求を受けるかを示す．

第 1 にはC++へ変換できる仕様で無ければならないことである．これは C++の持つ機能を実現することではなく，C++の仕様を借りて OEDJ 文書をプログラムとして実行できる言語として設計することである．したがって，仕様にはクラス，メッセージ送信，代入文などのプログラミングの概念・構造・文の記述を基本におくこととする．

次に OEDJ 文書は OONJ 文書からの変換が円滑でなければならない．OEDJ の仕様が OONJ から大きく隔たり PL に近すぎると，OONJ の NJ 記述を直接 PL 記述へ変換することになり，自然言語解析の困難さを OONJ から OEDJ に変換する段階に持ち込むことになる．場合によっては変換不能なため DU に OONJ 文書を PL へ全面書き直すことを強いることになる．DU に要求できるのは PL 変換のための補足記述の範囲に留めなければならない．

第 3 にはOONJを計算機の世界で実行できるようにすることが目的であるので，PL の持つ機能で分かりにくい仕様（具体的には C++のポインタ型など）を仕様に盛り込む必要は無い．また，インクルード文のような物理的な情報はトランスレータが自動的に生成することが必要である．

最後に OEDJ は中間的な言語で直接 DU が全てを書き下ろすことは無い．だが，テスト中あるいは記述後に OEDJ や C++を一部手直すことは考えられる．したがって，DU が OEDJ 文書を読めるようにすることは必要である．

言語仕様の基本方針は次のように整理できる．

(1) PL（ここでは C++）へ変換可能である．
(2) 個々の文の記述は NJ 記述に近似させる．
(3) 機能は最小限にし，自動出力する．
(4) OEDJ を単独でも読めるようにする．

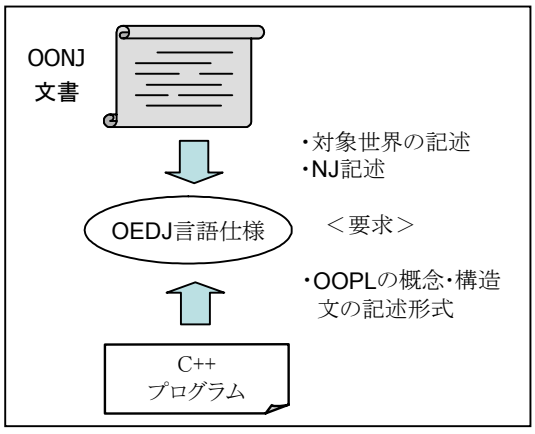

図1 言語仕様への要求

この中で(3)はトランスレータへの要求である. また, (4)は言語仕様を分かりやすくし, 開示することで可能である. したがって, 具体的な言語仕様への要件としては次の2点である.

(1)C++へ移行可能なクラス, 継承, プログラム構造
(2)C++へ変換可能な限定された NJ 記述

(1)については第 3 章の言語仕様の中で述べる. (2)については次節で限定された NJ 記述についてその考えを述べる.

2.2 限定された NJ 記述文の設計

OONJ は制限されている NJ 記述(単文, 平叙文, 5 文型)とはいえ, かなり自由度の高い文章が記述できる. 一方, OEDJ ではトランスレータで PL の文へ変換するために厳しい制限をつけなければならない. 以下, 属性記述, 振舞い記述(動作)の NJ の取り扱いについて考察する.

(1)属性記述の文

属性は OONJ の段階では日本語単文とはなっているものの複雑な記述ではなく, OEDJ では簡潔に記述できれば良く, NJ 記述にする必然性は無い. OONJ では属性名は決まっているが, データ型はまだ記述されていない仕様となっている. このため DU は OEDJ へ移行する過程で不足する情報(データ型等)を追記しなければならない. この過程で属性名に対し必要に応じて属性値, データ型を記述環境のダイアログ等を利用して入力できるようにすれば, 属性記述を実現できる. したがって, OEDJ の仕様としては比較的 PL 表現に近い記述形式とすることが可能である.

(2)振舞い記述の文

OEDJ での振舞い記述は NJ による記述を可能にしなければならない. これは OONJ で記述された振舞いを受け入れるには必要な要件である. しかし, 限定されているとはいえ NJ の記述は PL への自動変換が困難であるのも事実である.

しかし, オブジェクト指向のメッセージ送信は表現としてメッセージ(短い文)を送信する形式を取るという点から, もともと NJ 記述とは親和性がある. そこでまずメッセージ送信文について考えてみる. メッセージ送信は「オブジェクト」と「オブジェクト」の相互作用を記述する文であり, NJ 記述ではA, B, C, D をオブジェクトとすると,「A は B から C へ D を何々する. 」と記述できる. つまり, メッセージ送信はこの書き方に限定すると PL への変換が可能であると推測できる. そこで, 我々は振舞いの記述には限定的な**単文記述パターン**を導入する方法を採ることとした.

記述パターン導入の目的は複雑な自然言語解析を回避し, PL 文へ変換可能である限定された文の導入である. オブジェクト指向での動作はオブジェクト間の相互作用で表現する.

そこで単文記述パターンは

〈句〉 ”(が｜は)” 〈句〉“から” 〈句〉”を”
〈句〉”で” 〈句〉”に” 〈句〉“(に｜へ)” 〈動作〉

とした. ここで, 〈句〉は名詞を複数個の「の」で繋いだものである. 〈句〉の中の名詞はメッセージの「送信オブジェクト／受信オブジェクト／送受信対象オブジェクト」を含む. 文中に現れる名詞のひとつはクラス名を指す名詞(受信オブジェクト)で無ければならないとする. また, 動作はメッセージの一部を構成するもので, パターンにはひとつしか存在しないものとする. また, 全ての〈句〉を記述する必要はない.

ここで, 記述パターンの自然言語の解釈(例えば, 格助詞[6]の役割による解釈等)は不要である. というのは, OONJ ではメッセージ送信先を DU が記述する仕様となっているため, OONJ から OEDJ のこの記述パターンに移行してもその記述は残り, 自然言語解釈により相互作用の相手先クラスやメソッドを探索し, 特定化する必要はない.

なお, 本論文で導入した記述パターンは格助詞により名詞の役割が明確な「表層格」の概念[6]を参考としたが, そこでは名詞(句)の役割を表す格助詞の種類として, 主格(格助詞:が), 対格(を), 与格(に), 道具格(で), 方向格(へ／に), 奪格(から), 共同格(と)の7種類と取り上げている. しかし, この中の共同格は複数個の並列化の表現であるので, 元来一個ずつしか存在しない「送信オブジェクト／受信オブジェクト」では使われない. つまり, 共同格は「送受信対象オブジェクト」の役割限定に使われると考えられる. だが, 送受信対象オブジェクト

はすでに OONJ の段階で文の外に抽出されている．したがって，共同格は不要であるため記述パターンから外すこととした．なお，OEDJ では「送受信対象オブジェクト」は引数として扱うこととした．

3． OEDJ 言語仕様

3．1 構文記述則

本稿で構文の記述に使用する定義記号を**表 1** に示す．

本定義記号は拡張 BNF を参考とした．

文法定義記号

文法は拡張BNFで定義する。
定義は “=“で表す。
　　左辺は右辺によって帰納的に定義される。
< > 非終端記号
《 》 終端記号
a｜b aまたはb
( ) 結合の順序，ひとつの要素
[a] aまたは空
“a” aはOEDJの基本記号
1*a 1個以上のa を繰り返す
*a 0個以上のa を繰り返す
1#a 1個以上のaを “,”と省略可能な空白で区切って繰り返す．
// 行末までコメント

表1 構文記述記号

3．2 OO構造と駆動記述

表 2 にクラス構造と駆動シナリオの構文を示す．

(a)クラス構造

記述は「クラス」を単位とし，「クラス」は相互に関係を持つ．「クラス」の中に属性と振舞い(メソッド)を記述する．また，1 クラス 1 インスタンスという制限をつける．というのも，OONJ の記述には DU の混乱を避けるため，モノを操作する概念はあっても，クラスとクラスから生成するインスタンスを区別する考えが入っていない．そこで，OEDJ では OONJ を受け，「モノ」(クラス)で記述を統一することとした．一方，C++ではメッセージ送信先はインスタンスであるので，これはトランスレータが補足することとした．

(b)駆動シナリオ記述

DU がオブジェクトの駆動を明確に記述できるように駆動記述(他言語の主プログラムに相当する)を「クラス」記述とは別に設けた．<駆動属性記述>の<要素数>は配列の要素数を表す．

1． OO構造と駆動記述
1．1 OEDJ記述
<OEDJ文書>= 1*<対象クラス記述>
　　<駆動シナリオ記述>
1．2 クラス記述
<対象クラス記述>=“クラス”《クラス名》
　　[“%クラス種別” (“抽象”｜“具象”)]
　　*(“%クラス間相互関係” <関連クラス>)
　　[(<クラス制約>｜ <一般制約>)]
　　*(“%属性” (<対象属性記述>｜
　　<対象複合属性記述>))
　　*(“%振舞い” <対象振舞い記述>)
<関連クラス>=<汎化>｜<集約>｜<一般関連>
<汎化>= “is-a”《上位クラス名》
<集約>= “has-a”《被集約クラス名》
<一般関連>= “assoc”《相手先クラス名》
<クラス制約>= “クラス制約:” <制約記述>
<制約記述>= (″許可:″｜″禁止:″)
　　1*(《相手先クラス名》｜《相手先振舞い名》)
<一般制約> = “事前制約:” <事前条件>｜
　　“事後制約:”<事後条件>
<事前条件>=<条件>
<事後条件>=<条件>

1．3 駆動シナリオ記述
<駆動シナリオ記述>=“駆動シナリオ”
　　[“%クラス間相互関係”<関連クラス>]
　　*(“%初期化” <開始状況記述>)
　　“%駆動” <駆動記述>
<開始状況記述>=<駆動属性記述>
　　｜<駆動複合属性記述>
<駆動属性記述>=《属性名》 [“[”<要素数>“]”]
　　“:” <基本データ型>
　　[“＝” 1#《初期値》]
<駆動複合属性記述>=《属性名》 “:” 《クラス名》
　　[“＝”《初期値》]
<要素数>= <整数値>
<駆動記述>= 1* (<振舞い記述文>“.”)

表2 クラス構造と駆動シナリオの構文

3．3 属性とデータ型の記述

表 3 に属性とデータ型の構文を示す．

属性のデータ型は基本データ型とクラスを用いたクラス型からなる．属性には基本データ型に限定して配列が指定できる．配列の要素数は[]で囲む．また，初期値も設定できて初期値が要素数に不足の場合は初期値の最後の値を不足分として用いる．記述例 2 では初期値0が全ての要素に設定される．

(記述例1)　体長 :整数型 ＝5
(記述例2)　粒子[1000] :整数型 ＝0

属性のアクセス制限として私有(C++の public)，限定共有(protected)，共有(public)を設ける．また，属性は1クラス1インスタンスの原則からインスタンスごとに確保される変数は無いので全てが静的(static)扱いとなる．DU が OONJ 文書への追記で属性に与えることのできるデータ型は基本データ型のみとなる．

```
2. 属性と振舞い
2．1　属性
<対象属性記述>=《属性名》 ["["<要素数>"]"]
          ":" <基本データ型>  [<アクセス制限>]
              ["＝" 1#《初期値》]
<対象複合属性記述>=《属性名》
          ":"《クラス名》 [<アクセス制限>]
              ["＝"《初期値》]
<アクセス制限>= "アクセス制限:"《制限記述》
《制限記述》= ("私有" | "限定共有" |  "共有")
   // 省略時は属性では私有，振舞いでは共有．

2．2　データ型
<データ型>=<基本データ型> | 《クラス名》
<基本データ型>=<算術型> | <論理型> |
                    <文字型>
<算術型>= <整数型> | <実数型>
<整数型>= "整数型"
<実数型>= "実数型"
<論理型>= "論理型"
<文字型>= "文字型"
```

表3　属性とデータ型の構文

3．4　振舞い記述

表 4 に振舞い記述の定義(メソッド定義)と**OEDJ** 単文の構文，**表 5** に OEDJ 複文と条件の構文を示す．

(a)振舞い記述

振舞いは<OEDJ 単文>，<OEDJ 複文>(プログラム制御文)，<PL ライク文>で記述する．

(b)OEDJ 単文

メッセージ送信文に変換されるNJ記述(単文記述パターン)である．ここで，動詞の体言止は名詞との区別をつけるため禁止する．

(c)　引数

引数相当部分は OONJ にて NJ 記述の外に取り出されている．OEDJ では引数として括弧()で囲み，文の一部に取り入れる(fn は NJ 文種別番号)．

(OONJ 例)　fn3.2/c　　川に流れる

　　fn2.5.1　　| H_2O 増分

(OEDJ 例)　川に流れる(H_2O 増分)．

```
2．3　振舞い
<対象振舞い記述>= <対象振舞い定義文>
          <振舞い制約>
       1*(<振舞い記述文>". ")    // 本体
<対象振舞い定義文>=<OEDJ単文>
   // メソッドのヘッダ定義に相当
<振舞い制約> =<アクセス制限>
   // <属性制約>に同じ
<振舞い記述文>=<OEDJ単文> | <OEDJ複文> |
                <PLライク文>
<OEDJ単文>= [<主部>] <述部>
          // OEDJ単文は平叙文のみ。
<主部>= [<句> "(が | は | も)"]
<述部>= [<句>"から"] [<句>"を"]  [<句>"で"]
        [<句>"に"] [<句>"(に | へ)"] <動作>
<句>= <名詞句> | <リテラル>
<名詞句>= 《名詞1》 ["の"《名詞2》
            ["の"《名詞3》]]  | <引数>
《名詞i》= (《クラス名》| 《属性名》)
            ["["<要素>"]"]
      // iは1，2，3　要素は配列要素
      //《属性名》が書かれた場合は
      // 次の[の・・]以下は書けない。
<引数>="(　0*《実引数名》 ")"
<動作>::《動詞》
```

表4　振舞い記述の定義と**OEDJ** 単文の構文

(d)OEDJ 複文

プログラム制御文を表す．多方向分岐と2方向分岐を合わせた分岐記述文と繰り返しを表す反復記述文とからなる．

(e)条件

<OEDJ 複文>の<条件>部分は，「N1 が N2 を超え，かつ N1 が N3 を超えない」といったNJの重文が使われることが多い(ここで Ni(i=1～3)は名詞を表す)．論理式は条件を簡潔に正確に記述できる．情報リテラシーの水準から考えるに，DU にとって論理式の記述はさほど困難なことではない．したがって，解析が複雑になる<条件>部分は，<OEDJ 単文>で表現できない場合には論理式で記述することとする．上記の例では 「N1＞N2　and　N1＜N3」 と記述する．

OONJ 段階では自然日本語単文で表現されている可能性が高い．DU は OONJ 文書を OEDJ 文書(プログラム)へ移行する段階で論理式へ書き直すのが望ましい．トランスレータが解釈できない場合はエラーが出力される．

```
<OEDJ複文>= <分岐記述文>｜<反復記述文>
<分岐記述文> = “cond” <分岐条件>  “{”
    1*( ″case“ 《case名》 ″:“
          1* (<振舞い記述文>“. ” )  )
    [ “default” 1* (<振舞い記述文>“. ” ) ]  “}”
<分岐条件>=  “(“<OEDJ単文>｜<論理式>“)”
   // 条件が<論理型>ならば二方向
   // <整数型>ならば値の一致するcase名へ
《case名》= <整数値>
<反復記述文> =“loop” <繰返し条件>
       “{”  1* (<振舞い記述文>“. ” )  “}”
<繰返し条件>=“(“<OEDJ単文>｜<論理式>“)”
 // 条件は<論理型>で真の間{}内を繰返す
```

表５ OEDJ 複文と条件の構文

３．５ PLライク文

表 6 にPLライク文の構文，**表 7** にリテラルの構文，**表 8** に演算子の構文を各々示す．

(a)式と条件

一般的に DU のアルゴリズムには数式が多く用いられるが，式の記述は自然日本語の単文形式では不自然である．単純に属性の値を増減する演算は OEDJ 単文でも記述可能であるが，演算子を含む場合は一般的な式を用いた記述が合理的である．そこで式を含む代入文を<PL ライク文>という形式で書けるようにする．<PL ライク文>は OONJ 段階でも式表現になっている．<式>には算術式，関係式，論理式，文字式の 4 種がある．代入文では算術式と文字式が記述でき，算術式は算術演算子，文字式は文字演算子のみ使用できる．<条件>での式は論理式のみ使える．ただし，論理式の<項>として<関係式>を書くことが出来る．

(b)演算子

<式>で用いる演算子は DU の情報リテラシー能力から自然に使える範囲のものとし，C 言語等におけるインクレメント、デクレメント，ビット，代入，アドレス，間接などの演算子は記述できないこととした．

```
<算術単項演算子>= “－”       // マイナス
<算術二項演算子>= ”*” ｜ “/” ｜ ”+” ｜ ”－”
<関係演算子>=  ”<” ｜ ”<=” ｜ ”>” ｜ ”>=” ｜
                ”= =” ｜ ”!=”
<論理単項演算子>= ”not”
<論理二項演算子>= ”and” ｜ ”or”
<文字演算子> = ”||”          // 文字連結
```

表８ 演算子の構文

```
<PLライク文>=<代入文> ｜<応答文> ｜
                <停止文> ｜<空文>
<代入文>= 《属性名》”:=“ (<式> ｜”結果”)
   // 《属性名》と<式>のデータ型は一致する
<応答文>=“応答” <リテラル>
<停止文>=“停止”
<空文>= “－”
<式>=<算術式> ｜<論理式> ｜<関係式> ｜<文字式>
   // <算術式>のデータ型は<算術型>
   // <論理式>と<関係式>のデータ型は
   // <論理型>
   // <文字式>のデータ型は<文字型>
   // または文字列クラスのインスタンス
<算術式>=<算術二項式> ｜<算術単項式> ｜
          <ライブラリ参照> ｜<基本項>
<ライブラリ参照>=《メソッド名》 [<実引数>]
                “→”《インスタンス名》
   // 明示的なメッセージ伝達なので
   // インスタンス名を用いる
<算術二項式>= <項><算術二項演算子><項>
<算術単項式>= <算術単項演算子> <項>
   // <項>のデータ型は<算術型>
<論理式>=<論理二項式> ｜<論理単項式> ｜
          <ライブラリ参照> ｜<基本項>
<論理二項式>= <項><論理二項演算子><項>
<論理単項式>= <論理単項演算子> <項>
   // <項>のデータ型は<論理型>
<関係式>= <項><関係演算子><項> ｜
            <ライブラリ参照>
   // <項>のデータ型は<算術型>
<文字式>=<文字式> ｜<ライブラリ参照> ｜<基本項>
<文字式>= <項><文字演算子><項>
   // <項>のデータ型は<文字型>
   // または文字列クラスのインスタンス
<項>= <基本項>  ｜  “(“ <式> “)”
<基本項>= 《属性名》｜<リテラル>
<リテラル>=<数値> ｜<論理値> ｜<文字値> ｜
          “null”
```

表６ PLライク文(式)の構文

```
<数値>= <整数値> ｜<実数値>
   // <整数値>と<実数値>は<算術型>
<整数値>= 1*《数字》
<実数値> =1*《数字》)  ”. ” 1*《数字》
<論理値>=“真” ｜“偽”
   // <論理値>は<論理型>
《数字》 =0|1|2|3|4|5|6|7|8|9
<文字値>= “ ’ ” 1*《文字》“ ’ ”
<注釈>= “//” 《任意の文字列》
    // 行末までコメント
```

表７ リテラルの構文

3．6 名詞の種別

〈OEDJ 単文〉に含まれる名詞の種別については，名詞の種別は一般的に，クラス名詞，属性名詞，引数名詞，インスタンス名詞，一時変数名詞が考えられる． しかし，ここでインスタンス名詞を対象としない．

我々のシステムでは分析段階で「モノ」(オブジェクト)を中心にモデル化を行う．このため DU はクラス名とインスタンス名を厳密には区別していないと考えられる．したがって，〈OEDJ 単文〉ではインスタンス名詞を扱わない．これは必然的に 1 クラス 1 インスタンスとなる．DU はモノ名（クラス名）のみを意識して OEDJ の文を記述すればよくなる．C++へ変換する際にはインスタンスを生成する．

4． 開発支援環境の概要

本章では最初に OONJ から C++までの開発支援環境の全体像を示し，その後で OEDJ のトランスレータ設計方針について述べる．詳細は紙面の都合上割愛する．

4．1 開発環境の概要

図 2 に開発支援環境の全体と DU の開発プロセスを示す．支援環境は DU の記述した OONJ 文書を段階的に C++プログラムへ変換する．

本環境は次の機能を提供する．

(1) OONJ 記述エディタと確認用トレーサ

DU の OONJ 記述支援を行なう．また，OONJ 記述を自動トレースすることにより，相互作用相手の存在と記述の一致を確認する合理性検査を行なう．(OONJ では**2．2**の(2)にも述べたが，相互作用の相手先を指定する仕様となっている．)

OONJ は対象世界を網羅なく記述する．が，そのままの記述では計算機世界への移行はできない．例えば，この時点では属性のデータ型が記述されていない．あるいは条件が複雑なNJ記述となっており，移行が不可能なことがある．そこで，この時点で DU は不足データを追記する．

(2) OONJ トランスレータ

OONJ 文書を計算機移行可能な OEDJ 文書へと変換する．

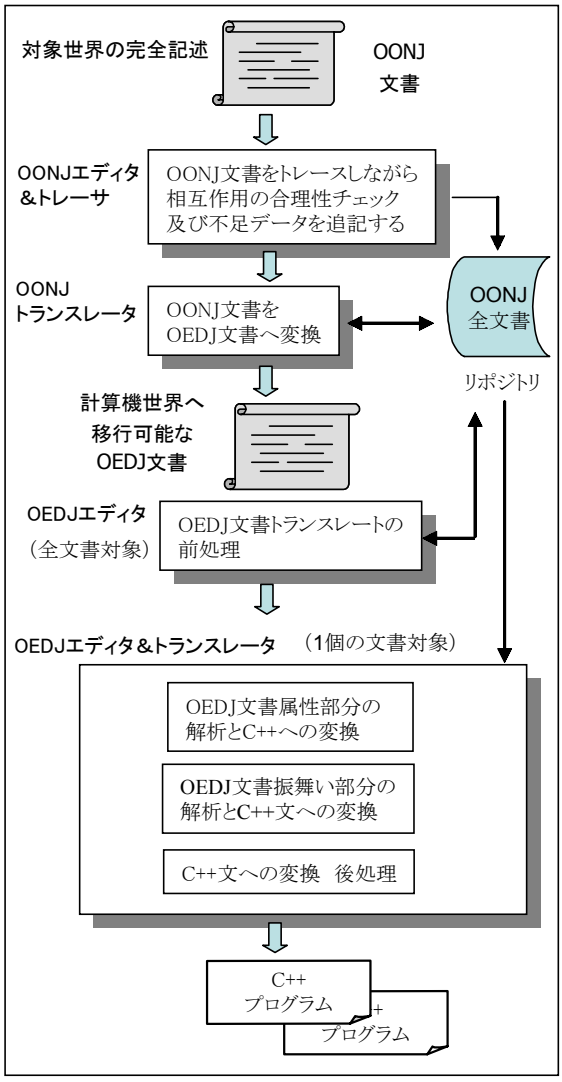

図2 開発支援環境とプロセス

(3) OEDJ エディタ

DU は OEDJ エディタを用いて C++へ変換するための不足情報がまだあるかをチェックする．また，日本語の単語に英字を対応させたい場合などはこの時点で英字を入力する．チェック終了後に OEDJ トランスレータを起動する．

(4) OEDJ トランスレータ

OEDJ トランスレータは OEDJ エディタから起動され，OEDJ 文書から C++プログラムを生成する．変換は次の 3 ステップを経て行なう．

(a) OEDJ 文書の属性部分の C++変数等への変換

(b) 同じく振舞い部分の C++メソッドへの変換

(c) インクルード文などの出力

OEDJ の一個のクラスは C++へ変換されるときには，

一個のヘッダファイルと一個の実装ファイルになる. したがって, 参照するメソッドを含むファイルをインクルード文で指定する必要がある.

(5)リポジトリ

OONJ 文書を解析し, 得られたクラス, 属性, メソッド情報を格納し, 処理の効率化のために用いる.

4. 2 OEDJ トランスレータの設計方針

OEDJ トランスレータの設計方針は次の通りである.

(a)DU の介入がし易いシステム

OEDJ 文書は先にも述べたように C++へ変換するには変換情報が不足しており, このままでは自動変換ができない. DU はデータ型等を記述する必要がある. これらの記述を円滑に行なうため, 例えば不足部分がすぐ分かる等の記述支援・記述誘導が必要である.

(b)C++プログラムの各種情報の自動出力

DU は FORTRAN を理解する情報リテラシーをもっているが C++の分かりにくい文法を理解していない. 逆にそれを不要とするのが本開発環境の狙いでもある. OEDJ トランスレータは例えば, インクルード文, extern 等を DU が記述するのではなく, 自動出力する.

(c)C++単独でも理解できるプログラムの出力

C++が分かる DU はアルゴリズム確認後, 高速化を図る, あるいは C++として一般に流通させたい要望もある. したがって, 日本語に対応する英語の入力を可能にし, 出力したプログラム単独でも理解できるようにする(通常は対応英字を自動出力するため, 理解しにくい C++プログラムとなる).

5. 記述例

図 3 に水の大気循環の記述例を示す. クラスは 1 個で, その中に属性 1 個と振舞い(メソッド)4 個が含まれ, 振舞いに 1,2 行のメッセージ送信を含む簡単な記述例である. DU は OONJ で元の文書を記述する. **図 3** の OEDJ は OONJ トランスレータで変換された結果である. OEDJ トランスレータはこの OEDJ を C++へ変換する.

```
クラス              大地
%クラス間相互関係    assoc 太陽
%属性      熱量 :整数型 =0
%振舞い    熱量(熱量減分)を減らす
              熱量から熱量減分を減らす.
%振舞い    降る(H2O増分)
              川に流れる(H2O増分).
%振舞い    川に流れる(H2O増分)
              川に大地から流れ込む(H2O増分).
%振舞い    熱量(熱量増分)を増やす
              熱量＝熱量＋熱量増分.
              川の熱量を増やす.
              大気の熱量を増やす.
```

図3 OEDJ 記述例

6. まとめ

本稿では OONJ をプログラム言語(PL)へ変換する手段として中間言語 OEDJ を導入する方式について述べた. OEDJ 言語仕様には C++へ変換可能であるようにオブジェクト指向プログラムの基本概念やプログラム構造すなわち, クラス, 相互関係(継承含む), 属性とメソッド等を取り入れた. また, OONJ からの変換が円滑に行われるように日本語単文記述パターンや 1 クラス 1 インスタンスの導入を試みた. 以上から, OEDJ の導入により OONJ の NJ 記述を PL 記述へ変換できる見通しを得た. 現在, 言語仕様およびトランスレータの設計を終了し, 図 2 に示す開発環境の全体を実装中である.

参考文献

[1] 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, シミュレーション学会誌, 掲載予定.

[2] 畠山正行, オブジェクト指向自然日本語記述言語 OONJ の設計とその記述例, 第 145 回ソフトウェア工学研究会研究報告, 情報処理学会, 2004.

[3] 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).

[4] 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).

[5] 加藤木,畠山,上田:オブジェクト指向プログラム設計記述言語 OOPD とその記述環境, 情報処理学会論文誌,Vol.43,No.5,pp.3016-3030(2002).

[6] 戸田他, 認知科学入門, サイエンス社(1986).