日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

オブジェクト指向を実現する日本語プログラミング言語の試作

岡田 健　杉浦 学　大岩 元

我々が開発した「ことだま on Squeak」は，プログラミング入門用の環境として実績をあげている．開発意図である「学習者が自分で書いたプログラムを日本語として理解できる」ことが有効であったものと推測され，これはエキスパートによるプログラム作成の場合についても同様の効果が得られると考えている．そこで，エキスパートによる利用を考慮し，オブジェクト指向プログラミングが可能な日本語プログラミング言語「言霊」の文法を考案し，処理系を試作した．オブジェクト指向型プログラミング言語としての文法と，正しい日本語で読むことを両立させるため，(1) 手続き呼び出しを 1 文で表現する，(2) 動詞・動詞修飾節・助詞付き引数・助数詞付き引数をシグニチャとする，(3) レシーバを先頭に配置する，という文法規則を採用した．

1　はじめに

我々が行っているプログラミング教育では，サンプルプログラムを書き写すだけでなく，学習者がプログラムの意味内容を理解した上でプログラムを記述できるようにすることを目標としている．

我々は日本語を使ってプログラミングを行うために,「Squeak」を改造して「ことだま onSqueak」を開発した．ことだま onSqueak はテキストベースのプログラミング環境ではなく，マウスによるタイルスクリプティングを使ってプログラミングを行う．ソートアルゴリズムを教えるためにことだま onSqueak を用いることで，受講者全員がソートアルゴリズムをプログラミングすることが出来た [2].

エキスパートによるプログラミングにおいても創作過程で日本語が思考に用いられるため，プログラムが日本語で表現されることが有効であると予想される．

我々は日本語でオブジェクト指向プログラミングが出来るプログラミング言語「言霊」を開発している．日本語で記述されたプログラムはコンパイラにより Java のソースコードに変換され，実行される．コンパイラ付属の専用エディタを開発し，試験的に慶應義塾大学の授業において学生に利用してもらっている．

本論文では言霊の文法の中でオブジェクト指向プログラミングに関係する，関数定義と関数呼び出し，戻り値，レシーバ，クラス定義の文法について述べる．

2　文法

我々は，オブジェクト指向型プログラミング言語として成立させつつ，正しい日本語として読めることを目指して，日本語プログラミング言語「言霊」の文法を定義し実装した．本論文では関数とクラス定義に絞って論じる．

2.1　概要

言霊は略記を避け，初学者でもコードを読むだけでプログラムの動作をイメージできるような表現を目指した．

以下は，インスタンスを生成して 1 つのメソッド呼び出しを行うプログラムを，言霊，Java の 2 つの言語で比較したものである．

Making of Object-oriented Programming Language in Japanese for trial purposes

Ken Okada　Manabu Sugiura　Hajime Ohiwa, 慶應義塾大学環境情報学部, Environmental Information, Keio University.

コード 1 言霊プログラム例

```
亀を新規作成して、それを亀太郎と名付ける。
亀太郎を右に３０度回す。
```

コード 2 Java プログラム例

```
Turtle 亀太郎 = new Turtle();
亀太郎.rt(30);
```

２つの言語を比較すると，Java 言語は略記しているのに対して，言霊はプログラムの意味内容が理解しやすい表現を目指している．インスタンス作成を記述する際に，Java は「new ＜クラス名＞ ()」と少ない記述で表現している．この記述法はプログラムを理解している人には少ないタイプ数で記述できるメリットがあるが，プログラムを理解していない人にはコードが持っている意味がイメージがしにくい．言霊は「＜クラス名＞を新規作成する。」と記述する.「新規作成する」という動詞を使うことで，何かが生まれていることがイメージできる．またクラス名に助詞の「を」が付くことで，クラス名と「新規作成する」の関係が明らかになり,「あ，＜クラス名＞が作られるんだな」と簡単にイメージできる．

変数初期化の記述も同様に，言霊はプログラム動作がイメージできる表現を目指している．言霊は「＜初期値＞を＜変数名＞と名付ける。」と記述する．変数定義を「名付ける」という動詞を使い，また初期値と変数名の関係を助詞「を」「と」で明示することで，変数定義という処理をイメージしやすくしている．

2.2 関数定義と関数呼び出し

言霊は関数定義と関数呼び出しを，プログラミング言語として成立させつつ，日本語として読めてプログラムの動作がイメージできるような表現になるように，文法を定義した．

本章では，言霊における関数定義と関数呼び出しの文法を扱う．レシーバを含む関数呼び出しの議論は 2.4 章に回し，本章ではレシーバを含まない関数呼び出しの議論を行う．コード１の２行目を例にすれば，本章で議論するのは「３０度右に回す。」の関数呼び出しの記述である．

言霊における関数のシグニチャには，以下の要素が含まれる．

- 関数名
  - 動詞
  - 動詞修飾節
- 引数
  - 助詞付き引数
  - 助数詞付き引数

コード１の２行目を例にすれば,「右に」が動詞修飾節であり「３０度」が助数詞付き引数であり,「回す」が動詞である．

以下の章では，動詞，助詞付き引数，動詞修飾節，助数詞付き引数の順序で，文法とその目的を述べる．

2.2.1 動詞

言霊における動詞は，既存のプログラミング言語における関数名に相当する．動詞だけを使って定義された関数は，言霊における最もシンプルな関数になる．

単純な関数定義例を挙げる．コード 3 は，文字列の出力に関する関数である．コード 4 のように呼び出すことが出来る．

コード 3 動詞のみの関数定義例

```
改行するとは {
  （省略）
} と定義する。
フラッシュするとは {
  （省略）
} と定義する。
```

コード 4 動詞のみの関数呼び出し例

```
改行する。
フラッシュする。
```

言霊では動詞活用形をサポートしており，以下の４種類の活用形をサポートしている．この４種類は，三上文法[1] を参考にした．

- 強変化形
- 弱変化形
- サ行変格活用形（以下，サ変と略記する）
- カ行変格活用形（以下，カ変と略記する）

コード３で宣言した動詞は「〜する」という表記なので自動的にサ変と解釈されるが，その他の活用形の動詞を使って関数定義する場合は，以下のように活

用形を明記する．

コード 5　様々な動詞活用形を使った関数定義例

```
閉じる（弱変化形）とは {
  （省略）
} と定義する。
開く（強変化形）とは {
  （省略）
} と定義する。
```

動詞の活用形を区別することは，日本語を理解している人なら可能である．動詞の表現が「〜する」ならサ変,「来る」ならカ変である．それ以外の動詞なら，動詞の末尾に「ない」を付けて否定表現を作り，その直前の子音がア行なら強変化形，そうでないなら弱変化形になる．例えば「開く」という動詞の否定形は「開**か**ない」なので，強変化形である．

動詞活用形をサポートする目的は３つある．一つ目は，戻り値を記述する際に重文と動詞活用が必要だった．詳しくは 2.3 章で述べる．

二つ目は，複数の命令文を繋げて文章を作ることで，コードを分割することが出来る．コード 6 はいずれもプログラミング言語的には同じ意味だが，一番下の例では前処理の「開く。」，出力処理の「改行して、フラッシュする。」，後処理「閉じる。」の３つに分割されている．出力処理のように複数の命令文を繋げて１つの文章にすると，プログラムの構造を分かりやすく出来る．

コード 6　動詞活用形を使った関数呼び出し例

```
開く。改行する。フラッシュする。閉じる。
開いて,改行して,フラッシュして,閉じる。
開く。改行して,フラッシュする。閉じる。
```

動詞活用が必要な理由の三つ目は，将来複雑な文構造を表現する時に動詞活用が必要になるからである．現在はまだ実装されていないが「５から１００までの範囲で乱数を発生させたものをｘに入れる。」のような複文を表現する時，従属節の「〜を発生させたもの」という記述を扱うために動詞活用を解析する必要がある．

2.2.2　助詞付き引数

言霊で引数を扱う手段の１つが助詞付き引数である．以下に引数を取る関数の定義例を挙げる．

コード 7　助詞付き引数を持つ関数の定義例

```
文字列（文字列型）を出力するとは {
  （省略）
} と定義する。
```

この関数シグニチャは，以下の通りになる．

- 文字列型の値を実引数として取り，その実引数には助詞「を」が付く．
- 動詞は「出力する」であり，活用形はサ変である．

助詞付き引数が複数になる場合は，以下のように関数を定義する．

コード 8　複数の助詞付き引数を持つ関数の定義例

```
出力先（ストリーム型）に
文字列（文字列型）を出力するとは {
  （省略）
} と定義する。
```

この関数シグニチャは以下のようになる．

- ストリーム型の値を実引数として取り，それには助詞「に」が付く．
- 文字列型の値を実引数として取り，それには助詞「を」が付く．
- 動詞は「出力する」であり，活用形はサ変である．

言霊の関数シグニチャには，引数順序は含まれない．そのため引数はどんな順序で記述しても構わない．だからコード 8 の関数を呼び出すときは，コード 9 のどちらの書き方でも，同じ関数呼び出しとして解釈される．なおコード 9 では，ストリーム型の変数「標準出力」が定義されているとする．

コード 9　複数の引数を持つ関数の呼び出し例

```
標準出力に「こんにちは」を出力する。
「こんにちは」を標準出力に出力する。
```

言霊の関数シグニチャに引数順序を含まないのは，記述のし易さを考慮したためである．言霊はプログラムが日本語文に近いため，ユーザはもちろん筆者でさえ関数呼び出しを記述するときに引数の順番を入れ替えて記述するのが普通である．そのため引数順序を入れ替えても適切に解釈してくれる言語文法にする必要があった．

2.2.3 動詞修飾節

言霊では動詞と動詞修飾節が関数名になる．既存のプログラミング言語では関数名に動詞以外の情報が含まれることが多いが，同じ事を日本語で表現するために生まれた文法要素が動詞修飾節である．

Java で定義された以下の関数定義を例に挙げる．

コード 10 関数に動詞以外の要素が含まれる例

```
void printLine(){ .. }
```

コード 10 では関数名に動詞「print」と名詞「Line」が含まれる．同じ関数を言霊で表現するには，以下のように宣言する．

コード 11 動詞修飾節が入る関数宣言

```
改行を出力するとは {
  （省略）
} と定義する。
```

コード 11 の「改行を」は動詞修飾節であり，関数名の一部である．また「改行を」は，コード 10 における「Line」と同じであり，助詞付き引数ではないことに注意する必要がある．

もちろん動詞修飾節を使わず，強引に動詞に色んな情報を付加して定義することも可能である．例えばコード 11 は「改行出力する」という動詞として宣言することも可能であり，動詞修飾節を使うかどうかを選択するかはプログラマ次第である．

動詞修飾節と助詞付き引数が混在する関数もある．以下に例を挙げる．

コード 12 動詞修飾節と助詞付き引数の混在例

```
改行付きで文字列（文字列型）を
出力するとは {
  （省略）
} と定義する。
```

コード 12 と同内容の関数を Java で定義すると，以下のようになる．

コード 13 コード 12 を Java で表現

```
void printLine( String str ){ .. }
```

Java では関数名に含まれていた「print」と「Line」が，言霊では動詞修飾節「改行付きで」と動詞「出力する」に分割されている．このように分割すると，助詞付き引数も混ざったときに，動詞以外の語順を自由に変えることも出来るようになる．コード 12 で定義された関数は，以下のどちらのように書いても呼び出すことが出来る．

コード 14 呼出例

```
改行付きで「こんにちは」を出力する。
「こんにちは」を改行付きで出力する。
```

動詞修飾節は，既存のプログラミング言語で使っていた関数名に動詞以外の情報を含める習慣を，日本語表現を損なわずに利用するために生まれた文法要素である．現在は動詞修飾節は名詞と助詞の組み合わせしか対応していないが，将来的には副詞や形容詞もサポートする予定である．

2.2.4 助数詞付き引数

言霊において，引数を扱う手段の 2 つ目が助数詞付き引数である．助数詞付き引数は,「〜度」「〜倍」「〜ドット」のように助数詞を付けて表現する．

2.1 章のコード 1 で紹介した「右に３０度回す。」という関数は，助数詞付き引数を使っている．その関数定義を以下に挙げる．

コード 15 助数詞付き関数の定義例 basicstyle

```
右にx（整数型）度
回す（強変化形）とは {
  （省略）
} と定義する。
```

コード 15 における「右に」は動詞修飾節,「x 度」は助数詞付き引数である．動詞修飾節と助数詞付き引数は，呼び出し時には順序を変えて記述することも可能である．以下の例は，どちらも同じ関数呼び出しとして解釈される．

コード 16 助数詞付き関数の呼出例

```
右に３０度回す。
３０度右に回す。
```

2.3 戻り値

言霊における戻り値の受け渡しは，重文と名詞「それ」を使って表現する．具体例を以下に示す．なお整数型の変数「長さ」は既に定義済であるとする．

コード 17 戻り値を使ったプログラム例

```
コンソールから整数を受け取って、
それを長さに入れる。
```

言霊では戻り値の受け渡しは，一時変数「それ」を用いて行われる．コード 17 の１文目「コンソールから整数を受け取って」は，戻り値を返す関数である．これを実行すると標準入力から入力を受け取り，その整数値を返す．関数呼び出し後，戻り値は一時変数「それ」に格納される．２文目「それを長さに入れる。」は代入文であり，変数「長さ」に変数「それ」の中身を代入する．

一時変数「それ」の値は，句点が現れた時点で消滅する．例えば「コンソールから整数を受け取る。それを長さに入れる。」というプログラムはコンパイルエラーになる．これは戻り値の利用が関数呼び出しの直後に行われないと潜在的なエラーの原因になると考えたからである．関数を呼び出してから１０行後に「それ」を記述しても，コードを読む方はどの時点の値が入っているか判別できない．

以上の理由から言霊における戻り値表現は「〜して、それを〜する。」という表現になり,「〜して」という表現をサポートするために 2.2.1 章で述べた動詞活用をサポートする必要があった．

2.4 レシーバ

本章ではレシーバを扱う．2.2 章の例は全てレシーバを省略していたが，実際の関数呼び出しではレシーバを指定する方が多い．

言霊におけるレシーバ記述のルールは以下の通りである．

- 文の先頭に記述する．
- 参照には任意の助詞を付ける

以下にその例を挙げる．

コード 18 レシーバを含む言霊の関数呼び出し

```
配列に 3 を追加する。
```

コード 18 において，レシーバは「配列」であり，任意の助詞「に」が付いている．関数名は「追加する」であり，助詞付き引数「３を」が与えられている．

レシーバに付く助詞は任意なので，以下もコード 18 と同じ関数呼び出しとして解釈される．

コード 19 レシーバに付く助詞が任意である例

```
配列に 3 を追加する。
配列へ 3 を追加する。
配列が 3 を追加する。
配列で 3 を追加する。
配列は 3 を追加する。
配列を 3 を追加する。
```

レシーバ記述において任意の助詞を付けるのは，日本語では主語を区別できないためである．例えば以下に，配列が持つ様々な関数呼び出しを，Java と言霊で記述したものを例に挙げる．

コード 20 レシーバを含む関数呼び出しの Java 表現

```
array.add(3);
array.remove(3);
array.clear();
```

コード 21 レシーバを含む関数呼び出しの言霊表現

```
配列に 3 を追加する。
配列から 3 を削除する。
配列を初期化する。
```

コード 21 を見ると，レシーバである「配列」に付く助詞が変わることが分かる．英語は主語と動詞とそれ以外を区別する語順になっているが，日本語の場合は主語を明確に取り出す文法が無い．日本語にはそもそも主語が存在しないとする主張も多い．

日本語でレシーバを特定するため，言霊は現実的な方策としてレシーバ記述のルールを作った．2.2 章では「引数や動詞修飾節の語順を変えても良い」というルールがある一方で，本章の「レシーバは先頭に記述しなければならない」というルールは矛盾しているが，今のところこの矛盾の解決には至っていない．

2.5 クラス

言霊はクラスを定義することができる．以下に例を挙げる．

コード 22 言霊のクラス宣言例（座標クラス）

```
座標とは {
  x（整数型、非公開）を持つ。
  y（整数型、非公開）を持つ。

  xを値（整数型）に設定する(公開)とは {
    xに値を入れる。
  } と定義する。

  yを値（整数型）に設定する(公開)とは {
    yに値を入れる。
  } と定義する。

  xを取得する（整数型、公開）とは {
    xを返す。
  } と定義する。

  yを取得する（整数型、公開）とは {
    yを返す。
  } と定義する。

} と定義する。
```

コード 22 は座標クラスの定義例である．２つの属性（ｘとｙ）と，４つの関数を定義している．４つの関数はアクセッサであり,「〜に設定する」がセッターで「〜を取得する」がゲッターである．

属性や関数定義にカッコ付きで「公開」「非公開」と記述されているのは，その属性や関数がクラス外部に公開するか非公開にするかを定義している．Java 言語におけるアクセス修飾子と同じ機能を持つ．

継承を使って，Java クラスをラップするクラスを作った例を以下に挙げる．

コード 23 言霊のクラス宣言例（タートル）

```
亀とは {
  Turtleを継承する。
  距離（整数型）ドット進める
  （公開、弱変化形）とは {
    「this.fd(距離);」をJava実行する。
  } と定義する。

  距離（整数型）ドット戻す
  （公開、強変化形）とは {
    「this.bk(距離);」をJava実行する。
  } と定義する。

  ※以下の定義を略す。
} と定義する。
```

コード 23 は，Java で定義された Turtle クラスを継承している.「＜クラス名＞を継承する。」と記述することでクラス継承を実現する．

またコード 23 は，Java で実装したメソッドを言霊の関数表現でラッピングをしている.「〜ドット進める」という関数は，内部で Turtle クラスが持つ fd メソッドを呼び出している．

このように，コード 22 のように言霊で独自にクラスを定義したり，コード 23 のように Java クラスをラップすることで機能を実現することができる．

3 まとめ

本論文ではオブジェクト指向を表現できる日本語プログラミング言語の文法を提案した．関数呼び出しにおいて動詞，動詞修飾節，助詞付き引数，助数詞付き引数を使うことで，プログラミング言語として成立させつつ，日本語として理解できる表現になった．日本語を使ってクラスが定義できることを示した．またレシーバを指定するために現実的な文法を採用していることを示した．

今後の課題は，この文法を使って中規模以上のソフトウェアを作成して，文法が一般のソフトウェアを作るに足る能力を持っているかを検証することである．現在は大学の授業においてプログラミング実習に使っている段階であり，そこで書かれるプログラムの量は充分に長いとは言えない．今後は言霊のコンパイラを言霊で書くことを通じて，文法を検討していきたい．

参考文献

[1] 三上章：日本語の構文，くろしお出版，1963.
[2] 杉浦学, 松澤芳昭, 岡田健, 大岩元：アルゴリズム構築能力育成の導入教育：実作業による概念理解に基づくアルゴリズム構築体験とその効果, 情報処理学会論文誌 Vol. 49 No. 10 (2008), pp. 3409–3427.