1E-7

統合プログラミング環境ＯＲＡＧＡ（Ⅱ）
名前統合管理システムＮａｍｅＭａｓｔｅｒ

森　厚　神田陽治　田中英彦　元岡　達
(富士通)　(東京大学工学部)

1. はじめに

ソフトウェアの生産性を高めるため，多くの研究者エンジニアは多種多様なアイデアをもって，この研究に打ち込んできた。そのなかで近年，オブジェクト指向型言語が注目をあびている。オブジェクト指向型言語は「オブジェクトを現実の事物としてプログラミングするので構造が理解しやすい。」という特徴をもっている。しかし，分かり易いプログラムとは構造のよさだけで実現できるわけではない。

というのは，プログラムが作成者の考え方を表す創造物であるいじょう，そこから逆に作成者の考え方が読み取れなくてはならないからである。それらを実現する鍵となるものに変数名，モジュール名等の名前付けがある。

我々はこのプログラムの名前付けにスポットをあて，良い名前付けを検査する名前統合システム—ＮａｍｅＭａｓｔｅｒ—について論ずる。

2. 良い名前とは何か

従来から，名前付けに注意しプログラミングすることは各開発プロジェクトで行われてきた。我々は良い名前とは何かをみつけるため，実際どのように名前をつけているか調査し，検討してみた。その結果以降のようなことが目についた。

図１の(1)のように言語の制限あるいは入力の煩わしさから省略している場合が多く，省略方法は人様々で理解しずらい。また，大規模なプロジェクトでは，(2)のように名前をコード化し，短い名前でより詳細な意味を表す工夫をしている。これらに共通していえることは，「名前を省略すると入力は効率的に行えるが，広範囲において記号性—名前とその内容が強く結びついている状態—を維持することは難しくなる。」ということである。これは，作成者，プロジェクトメンバには理解できるが，他の人には理解できないプログラムができあがってしまうことを意味している。(3)は機能が理解できない場合の例である。一般的に「リテラルからは値や型しか理解できず，プログラム上での役割をとらえることは難しい」ということがいえる。変数名も同様で，(4)のような名前では，ファイルが使われている事実とその使用個数しか読み取れない。我々が名前を読む時に欲しいのは機能と役割の情報である。Ｆｉｌｅ１をＩｎｐｕｔＦｉｌｅとすれば，入力するという役割とＦｉｌｅという機能が読み取ることができる。

(5)は名前間で関係がある場合の例である。”ｉ”という単独では意味をもたない名前もパタン化されて理解できる場合がある。よい例が，ｆｏｒ文である。ｆｏｒ文とｉは多くの人が，いわば一般常識的に使用している。しかし，いったんｆｏｒ文を離れてしまうと，単に”ｉ”では関係が理解しずらくなる。

まとめると，良い名前の条件とは

(1) 個人の理解できる名前であり，
(2) その名前の役割と機能が明確であり，かつ，名前間の差異から役割と機能の違いが理解できる。

ことだと思われる。

```
(1) Ａｒｇｃ                          —> ＡｒｇｕｍｅｎｔＣｏｕｎｔ
(2) ＦＩＬＥＯＩ１                    —> ＩｎｐｕｔＯｎｌｉｎｅＭａｓｔｅｒＦｉｌｅ
          |  |  └———————> 用途
          |  └———————> 入出力区分
          └———————> Ｏｎｌｉｎｅ，Ｂａｔｃｈ区分
(3) ｉｆ  (GetCharacter == −１)       —> ｉｆ  (GetCharacter == ＥｎｄＯｆＦｉｌｅ)
(4) Ｆｉｌｅ１ (入力に使う)           —> ＩｎｐｕｔＦｉｌｅ
    Ｆｉｌｅ２ (出力に使う)           —> ＯｕｔｐｕｔＦｉｌｅ
(5) ｉ ＝ .....                       ＩｎｄｅｘＯｆＳｔａｃｋ ＝ .....
      .                               —>                .
      .                                                 .
    Ｓｔａｃｋ〔ｉ〕＝ ...                 Ｓｔａｃｋ〔ＩｎｄｅｘＯｆＳｔａｃｋ〕 ＝ ...
```

図１．名前付けの悪例，改善例

3. 再利用の方法

数年前からプログラム部品の再利用が，ソフトウェアの生産性を向上させるための技術として注目されている。部品化は新規プログラムの開発量の削減にはおおいに役立っているが，反面，再利用する際には，まだいくつかの問題点が残されている。

それは，

(1) 部品名が必ずしも内容を反映しているとは限らないから，名前だけの検索では不十分。

(2) 部品間の機能差が明示されていないので，最適な部品を選択しにくい。

(3) 部品の実装部分が読みやすく書かれているとは言えないので，改造や考え方の再利用が困難でありうる。

ことである。我々は，これらの問題点を解決するための解決案を提案する。

(2)の問題点を解決するには，知識ベースを用意しオブジェクト指向型言語特有のインヘリタンス機構を利用して，モジュール間の類似性をそのまま格納する。これをモジュールベース化と呼ぶ。

(1)，(3)の問題点を解決するには，部品に名前式の記述を追加し，部品自身が名前を決定するようにすればよい。名前式は複合語であり，複合語は単語あるいは，他のモジュールの名前への参照の並びである。格納対象が任意に定まっていないスタックを例にとると

Std-prefix? Stack Of $contents Std-suffix?

等と記述する。$は引用を，？は省略可能を示している。この候補となる名前にはHerStackOfChar，等がある。

オブジェクト指向の考え方は「現事物をそのままプログラムの世界へシュミレートする」ということであるから，モジュール名もしっかりと現実の名前にシュミレートすべきである。その意味からも，名前式は重要であると思われる。これら解決の鍵となるモジュールベースや名前式を活用することで指定した要求に合うモジュールがないときや要求があいまいなときにでも検索を行うことができる。また，名前式では，他のモジュールの名前を引用できるのでモジュールベースの拡張が容易になる。

4. NameMasterの役割

ここでは，ORAGAシステム中のNameMasterの役割について述べる。NameMasterの役割は，

(1) 入力時の不完全さを補い，名前の記号性が保たれた良い名前であることを検査し，さらに，

(2) 出力時の分量を記号性を保った圧縮技法により削減する。

ことである。これらの役割をインプリメントするため，並列オブジェクト型指向言語—DinnerBell—〔1〕，ユーザインタフェース—ObjectPeeper—〔2〕と密接に情報を受け渡しその情報は，単語を処理するfrontend部と複合語を処理するmodulebase部で利用する。その動作を図2を使って説明する。

受理では，個人語彙を用いて，①で受けた文字列から候補となる単語を選びだす。そして，その単語からモジュールの候補を②で抽出し，③で文脈を用いて整合するモジュールを絞りこむ。ここまでの処理で，まだ候補が多数ある場合は，名前の記号性がないものとし，セマンティクス・エラーとして処理する。

生成では，まず④で整合モジュールから名前式を生成し，⑤では名前式から単語列に，記号性を保つよう1：1で変換する。さらに，⑥でもとの文字列と圧縮形が対応できるように圧縮し，理解を助けるために1ヶ所はもとの形で写す。これらを行うことで記号性を保った圧縮変換が実現する。

5. おわりに

本論文では，生産性を向上させるための名前付けの方法及び，それをインプリメントするためのNameMasterの役割，構造について論じた。現在，NameMasterは，設計，試作の段階にある。

＜参考文献＞

〔1〕本大会1E－6　〔2〕本大会1E－8

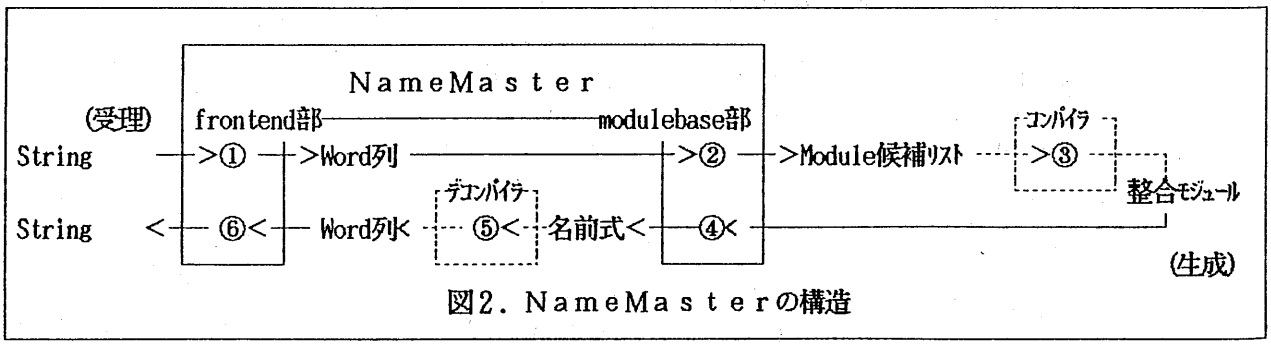

図2．NameMasterの構造