名前付けの統括管理システム，ＮａｍｅＭａｓｔｅｒの構成

神田陽治（こうだ）・　田中英彦　・　元岡　達
東京大学　工学部

論文を読むかどうかは，題名を見て決める。語・図の表出と指示する意味の対応は恣意的な約束事に過ぎないが，結び付きは強力なものであり，プログラミングの能率向上に利用できると考えたところから，この研究は出発する。

始めに，NameMasterを含むプロジェクトであるＯＲＡＧＡ計画の説明をする。次にNameMasterの概念を述べ，事例・研究をまとめ，機能を考察する。最後に，NameMasterを使う利点を調べる。

Ⅰ．　ＯＲＡＧＡ計画

ＯＲＡＧＡは object-ORiented Architecture to Govern Abstractionsの略であり，ＯＲＡＧＡ計画の目的は，人間の持つ抽象化能力を十分生かすことのできる計算機を構成することにある。これは，次の４つのサブシステムからなる。

□　名前の意味を知識として管理する　NameMaster
□　オブジェクトの表示・修正・管理をする　ObjectPeeper
□　並列・オブジェクト指向・システム記述言語である　DinnerBell
□　ネーム・オブジェクト指向のデータ駆動型の実行機構である　Oragaltself

名前管理のNameMaster(3)，オブジェクト管理のObjectPeeper(4)，言語処理系のDinnerBell(5)の３者で統合プログラミング環境を提供する。ネーム・オブジェクト指向アーキテクチャの Oragaltself(4)はセマンティクギャップを縮め，データ駆動による高速な実行環境を提供する。

Ⅱ．　ＮａｍｅＭａｓｔｅｒの提案

NameMasterの研究は，次の研究分野を含む。

□　意味を扱う知識処理の研究
□　ソフトウェアの再利用を進める研究
□　より良いユーザインタフェースを求める研究

１．　NameMasterの発想

プログラム言語では，識別子は互いを区別することが第一義で，適切なわかり易い名前付けは，プログラミング作法として奨励されるに過ぎなかった。ところで，日常言語では単語を別のものに置き変えてしまうと，会話が成立しなくなる。プログラムの世界でも再利用を進めるためには事情は同じで，名前付けを管理して，名前が常に同じ意味を持つことを保証することが重要となる。この管理をコンピュータに載せて，管理する範囲をできる限り広く取ることで，多くのプログラムでの語釈が統一される利点が多いことを示すことが，NameMaster計画の目標である。もちろん，NameMaster自身の実現方法を求めること自体も計画の一部で，意味を扱うところから，知識処理の問題としても興味を持てる。

２．　過去の事例・研究

NameMasterが実現しようとしている機能は，始めは唐突に聞えるかも知れない。しかし，関係ある事例・研究をいくつも見付けることができる。

Ａ．　適切な名前付けの必要性を説く事例

良いプログラムの条件の一つが，良い名前の選択であるという指摘は数多い。また，Unixのコマンド名は，省略法が一定でないとか，適切な名称でないものが多い。たとえば，writeを許可するのはmesgであるが，biffでmailを許可する。一方で，msgsはmailとともに使う。

たくさんのネットワークがぶら下がるインターネットワークでは，メイルの宛先の解決など，資源の名前からネットワーク上での位置を見付けることができなくてはならない。これを受け持つのがNameServerであり，解決が容易な名前付けの方法が必要である(9)(10)。

B．適切な名前付けを強制する事例

Spell checker と Spell correctorは，綴りの誤りを見付けたり修正してくれる。言い変えれば，語彙から逸脱した単語を排除する。Style と Dictionは，Unix上の文書系ツールで,Styleは英文の善し悪しを表面的な特徴から計算し，Diction は英文中の冗長な表現を抽出する。意味まで考慮しないものの，筆運びの悪い英文を指摘し，排除する効果を持つ。

文献検索などのための検索キーは，文献の内容を的確に表現する一方，文献全体のなかで十分な分解能を持つ必要がある。検索キー全体は検索操作における語彙といえ，検索者はキーを並べて検索文をつくる。

C．適切な名前付けの必要性を示す研究

プログラムの言語学側面の研究は学際領域にあるためか，ほとんど見られない。プログラムには文法書に規定されない人的な側面があり，研究の余地があることが指摘されている(2)。また，エディタのコマンド体系の作り方，分り易いエラーメッセージとは何か，名前の良い省略の仕方などの決定を，人間を対象にした実験で明らかにしようとする研究も行われている(6)。頭で考えた方法が必ずしも良くないことが示され，現実のデータに基づく研究が大切であることがわかる(8)。

一方で，プログラムの表面的な意味を考慮しない計量的な研究は多くなされ，ソフトウェア工学の一分野でもある。Unixのコマンド使用の多変量解析などの例もあり(7)，実験的に良いコマンド体系を作る方法としても注目できる。一群のプログラムの語彙についてZiphの法則が成り立つという観察もある(3)。

プログラムを作る際の語彙を統制する研究もある。プログラムは最終的な結果でしかないが，作成過程を残すことでプログラムの読み易さを増し，保守性を高めることができる。一つの方法は，プログラムとドキュメントを合体し，制限した自然言語で書かれた仕様を思考に従って順に具体化していく過程を記録しておくことである(1)。得られたプログラム中の名前は自然に内容を良く表し，十分な解説が付いている。

3．NameMasterの機能

NameMasterの目標は，プログラムに使う名前の意味を統制することにより，プログラムの理解を容易にし，プログラムの再利用を進め，また，コマンド名にも適用範囲を拡げることで，使い易いユーザインタフェースを作ることにある。以下では，NameMasterとDinnerBellは密接な関係にあることを示す。従来の方法との対比をしながら説明する。

A．プログラムでの名前

プログラムに現われる名前から考える。ただし，ここではレキシカルアナライザを通り，外界の制約（たとえば，文字種が少ない）を解決した，トークンと呼ばれる単位で考える。この粗さで名前を分類すると，

・予約語　　(<if>, <while >,...)
・特殊記号　(+ , ; , : =,...　　)
・リテラル　(360, " 3B20 ",...　　)
・変数名　　(myBaby, sweetHome,...)

の４つになる。これらは，プログラムの構文を BNFで定義したときの終端記号にあたる。前の２つは固定の，後の２つは可変長の情報である。

プログラムテキストでの変数名の BNFは，ふつう，

名前　::=　英字 { 英字 | 数字 }

なる文字列である。構造を持つ例もあるにはあり，たとえば，LISPのアトムシンボルのプロパティリストがある。リテラルはそれ自身の表現がそのまま自分の名前になっている。

NameMasterでの名前はどうあるべきか。名前の意味の記述のために，構造体を付与することを思い付く。ここで，発想を転換して構造体を主とし，従来の文字列の名前（以後，表現子と呼ぶ）は，構造体（以後，記述子と呼ぶ）から計算して作ることにする。すなわち，記述子は目に見えない構造データであり，表現子は記述子を人間向けに表示したものである。

NameMasterでの名前　＝　記述子 ＋ 表現子.

記述子から表現子への変換を固定せず，わかり易い表現子に変換する。いくつもの記述子をまとめて表示するとき，同じ表現子への変換を禁止するとともに，表示する状況を考慮して，無駄な重複を避け，互いの関連が分るように変換する。表現も文字列・音声・図形とし，見易く読み易くわかり良い表示法を選択する。表現子から記述子への変換も状況を設定し，表現子と記述子の対応を明確にすることで行う。

{記述子,...} ◁ 状況を指定して，相互変換 ▷ {表現子,...}

この対応を人間の直感に近付けるために，記述子から算出される表現子を見たときに連想する意味概念を，記述子に蓄えることにする。意味とは互いの相違をはっきり認識することであるとの立場にたてば，記述子全体は，組織化された意味ネットワーク，あるいは，階層化された知識ベースとなる。これは，自然言語の語彙にあたるもので，プログラムを作るための語彙となる。これを以後，名前ベースと呼ぶ。

名前ベース　＝　表現子の意味を蓄える記述子全体を組織化したもの．

基礎的な記述子を集めて組織化し語彙を構築することは，知識表現の問題である。基本的な表現子を単語，複合された表現子を文とみると，複合表現子の解析は自然言語文の解析と同じになる。状況を考えた上での記述子と表現子の相互変換は，文脈を考えた文章理解や談話理解の問題である。ただ，プログラム言語特有の性質がある可能性があるので，プログラムの語彙や表現子の構造についての研究が望まれる。

B．コンパイラの記号表

次に，名前のコンパイルを調べる。ふつう，コンパイラは記号表を自分で管理し，名前対応にエントリを作り，コンパイルするのに必要な情報を書き込んでいる。コンパイル後には，リンカーやデバッガーに必要な情報のみ残し，記号表を捨て去っていた。

DinnerBellでの記号表はどうあるべきか。記号表には名前が記録される。名前を記述子に拡張したのにともない，記述子の情報を複数のプログラムを通じて役立てることとし，記号表を使い捨てにせず，全てのプログラムに対しただ一つの記号表を使う。こう考えると，記号表は名前ベースに他ならない。

DinnerBellの記号表　＝　名前ベース．

C．コンパイラの中間木

コンパイラは２つの部分，パーザとコードジェネレータからなる。パーザはプログラムを中間木に変換し，コードジェネレータは中間木をコードに落し，ハードウェアがコードを解釈・実行する。中間木を直接，解釈・実行するのがインタプリタである。 BNFで記述されたプログラムをパーズして得られる中間木のノードは非終端記号，リーフは終端記号に対応する。終端記号は，既に述べたように，予約語・特殊記号・リテラル・変数名に区分できる。予約語・特殊記号は不変なので，リーフには固定情報が書き込まれる。リテラル・変数名は可変な情報なので記号表にしまわれ，リーフにはポインタが置かれる。そして，それらの意味を決める解釈ルーチンはコンパイラのアルゴリズムのなかに埋めこまれていた。

DinnerBellでの中間木はどうあるべきか。DinnerBellの記号表は名前ベースに融合され，プログラム中の名前（終端記号）は記述子として，名前ベースに格納される。ここで，統一性・柔軟性を考えて，非終端記号も名前ベースにしまうことにする。この結果，中間木は記述子からなる関係木となる。

DinnerBellの中間木　＝　名前ベース上の関係木．

D．オブジェクトを知識表現に使う

オブジェクトとは，組織化可能な抽象データ型である。抽象データ型とは，強く関連したデータと操作を集めてパックしたものである。そして，互いの共通点を抜き出してまとめるなどして，全体を体系化できる抽象データ型がオブジェクトである。オブジェクトは自律して動き，その起動はメッセージを送ることで行う。NameMasterでは，知識表現としてオブジェクトを採用する。名前ベースはオブジェクトのデータベースとなり，コンパイラの中間木はオブジェクトをノード・リーフとする木となる。

名前ベース　＝　組織化された多数のオブジェクト．

中間木　＝　複数のオブジェクトの関係木．

E．全体の構成

以上の議論をまとめる。従来のコンパイラはプログラムを中間木に変換するが，実行に必要な情報しか残さない。一方，自然言語の解析プログラムは，自然言語文を構文木やより深い解釈の意味表現に変換する。NameMasterと共同作業するDinnerBellコンパイラは，プログラムの構文を解析するばかりでなく，名前の構造・意味をも解析して，いままでよりも，より精密な構文木を作る能力を持つ。NameMasterは名前の意味を解析するための語彙，すなわち，名前ベースを管理するデータベースシステムである。〔次ページ図参照〕

この中間木のノード・リーフはオブジェクトであり，メッセージを送って仕事をさせることができる。たとえば，この中間木のルートに”eval”なるメッセージを送って起動すると，メッセージが木の上を流れて，各オブジェクトが次々と発火する。nameノード”get-a-Key ”は，”eval”を受けても，それ以上，下のノードにメッセージを伝えないで，キー値を返す。ルートに”print ”を送ると，このときはnameの下のノードまで伝わり，プリント値”get a Key ”を戻す。

4.　いくつかの利点

最後に，これまでに述べた機能を持つNameMasterを用いる利得について説明する。

□ 思考が滑らかに進む

名前は見た通りの意味を持つので，マニュアルなどを参照することが減り，母国語の文を読むときと同じように，思考が滑らかに進む。プログラムの可読性を高め，再利用を進める。

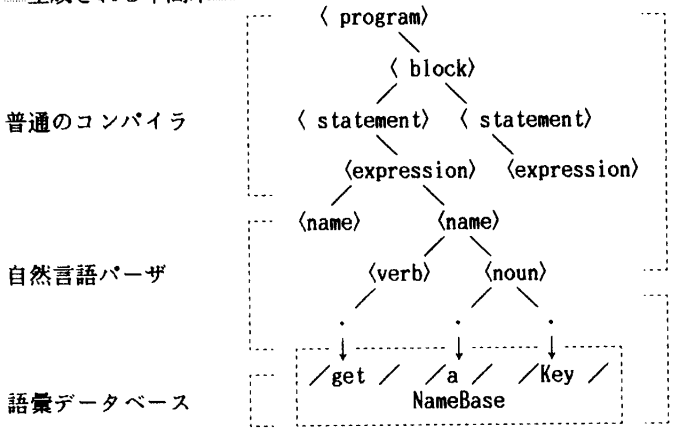
□ 見たい観点からの表示が得られる

NameMasterは一群の記述子を表示子に変換するが，状況をいろいろに設定することにより，見たい点を強調した表示を得ることができる。この機能は他のサブシステムに利用される。たとえば，ObjectPeeperはオブジェクトを大きさの限られたウィンドウに圧縮表示する機能を持つが，その圧縮操作に応用される(4)。また，DinnerBellのプログラムをObjectPeeperによって表示するとき，名前の表示の細かさをこの機能を使って制御し，必要に応じていくらでも詳しく説明できるので，DinnerBell言語にはコメント文が無い(5)。

□ 表現子の統計分析よる分類

NameMasterは一群の記述子を，指定した状況の下での関連・相違がはっきり分るように変換する。従って，求めた表現子を見比べれば，指定した観点からの記述子の分類ができるはずである。表示子間に距離をいれて，クラスタ分析などの統計分類を施すことで，これを実現できよう。プログラムの機能分類や参照関係の分析などの応用が考えられる。

付図．生成される中間木



■

参考文献．

(1). 榎本　肇．他．自然言語に基づくソフトウェア開発システムTELLの概要．他．
情報処理第28回全国大会,1984,2J-1,2,3,4,5,6,7,8

(2). 金田　泰．　プログラミング言語学をめざして．
東京大学情報工学修士論文,1981.

(3). 神田陽治　他．プログラムに出現する名前の性質について―名前管理システムの実現に向けて．
情報処理第29回全国大会,1984,2R-6.

(4). 神田陽治　他．ソフトウェア作成支援のためのネームベース・アーキテクチャとその実現法．
電子通信学会電子計算機研究会,1984,EC83-55.

(5). 神田陽治　他．並列オブジェクト指向言語 DinnerBell の概要．
情報処理学会ソフトウェア基礎論研究会,1984,（12月14日発表予定）．

(6). Badre A.,and B.Shneiderman (ed.) .
Directions in Human ／Computer Interaction.
Ablex,1982.

(7). Hanson S.J.,E.T.Robert,and J.M.Farber.
Interface Design and Multivariable Analysis of UNIX Command Use .
ACM Trans.Office Inf.Syst.,Vol.2,No.1,1984,pp.42-57.

(8). Landauer T.K.,K.M.Galotti,and S.Hartwell.
Natural Command Names and Initial Learning : A Study of Text-Editing Terms .
Comm.ACM,Vol.26,No.7,pp.495-503.

(9). Lyngbaek P.and D.McLeod.
Object Management in Distributed Information Systems.
ACM Trans.Office Inf.Syst.,Vol.2,No.2,1984,pp.96-122.

(10). Oppen D.C.,and Y.K.Dalal.
The Clearinghouse :
A Decentralized Agent for Locating Named Objects in a Distributed Environment .
ACM Trans.Office.Inf.Syst.,Vol.1,No.3,1983,pp.230-253.