(14)

EC83−55

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

神田陽治・田中英彦・元岡　達
（東　　大）

1984年3月29日

社団法人　電子通信学会

EC83－55

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

Name-based Computer Architecture for Software Design

神田　陽始　　田中　英彦　　元岡　達

Youzi Kohda　Hidehiko Tanaka　Tohru Moto-oka

東京大学　工学部

Faculty of Engineering
University of Tokyo

1　はじめに

概要

コンピュータを積極的に利用するためには，プログラムを開発しなければならないが，プログラムの作成には多くの困難があることは，ソフトウェア工学が提唱され発展してきたことでも知れる。

我々の研究は，このソフトウェア作成という難しい作業をコンピュータシステムによって助けることができると考え，作成作業支援用のコンピュータの新しい構成を求めることを目的にする。

具体的には，ソフトウェア工学の結果（オブジェクト指向）と新しく提案する概念（ネーム指向）を使い，プログラミング作業を助けるユーザインターフェスを与え，ハードウェアの進歩（マイクロプロセッサ，ビットマップディスプレイ，マウス）により可能となったスーパーパーソナルコンピュータの形をとる。

このシステムORAGAでは，『名前』が重要な位置を占めている。名前はシステム内で，厳重に保護すべき情報と定義する。名前の考え方自体は決して新規性のあるものではないが，ハードウェアからソフトウェアまで一貫して名前の概念を中心に設計することに独自性がある。

名前はいろいろな用途で用いられる。オブジェクト（プログラムで扱いうる計算資源の単位）を指す名前がある。それ以外の名前は全て何らかの概念を指すものである。例えば，例外処理要求を意味する名前や，手続きに必要な引数対応を意味する名前を考えることもできる。これらの名前は，ソフトウェアのレベルでは目に見える文字列（プログラムの識別子）に，ハードウェアのレベルでは組込みの命令セットが扱う情報の最小単位として具体化される。

名前が重要なのは，それがコンピュータにとっても，人間にとっても，事物の同一性，類似性の判断基準となるからである。コンピュータは名前が同じものを同じ概念，もしくはオブジェクトと判断する。人間はよく似た名前が指すものは同種と判断し，多くのものから欲しいオブジェクトを選びだす。数多くある講演から自分の聞きたい講演を選択する際の判断基準はそれに付いた名前，即ち講演題目からである。

ORAGAプロジェクトはプロトタイプORAGA－I，プロダクトORAGA－HALを目指して計画段階にある。本論文はその目標，設計思想，重要な着想を述べたものである。

2　研究の背景

課題

コンピュータは種々のプログラムによって多くの仕事を行いうる汎用の機械である。このためのプログラムを作る困難さは，ソフトウェア工学が提唱されたことからも知れるように非常に大きい。

なぜ，プログラミングが難しいか？

優秀なプログラマは与えられた問題を理解し，自分のものとしたうえで，（直感に従い）扱い易いまとまりのよい単位へと問題を分割している。これより（一般プログラマにとって）プログラミングが難しいのは，以下の理由によると考える。

> 与えられた問題を素早く自分の言葉で理解することは難しい。

3　研究の目的

第１の目的

・ソフトウェア作成支援コンピュータアーキテクチャを求める。□

コンピュータの適用範囲が急速に広まるなかで，ソフトウェアの生産現場はその需要を満たすためにますます混乱の度合を深めている。（マーフィ的逆説。）これをコンピュータアーキテクチャから根本的に解決したい。

第２の目的

・（こうして得られたコンピュータシステムで）真にソフトウェアの生産性を向上させるためのユーザインターフェスを求める。□

Batch job からTSS へ移行し会話処理となり，ソフトウェアの大幅な生産性向上が期待されたにもかかわらず数倍に止まった。ハードウェアの能力向上は目覚しいが，（個人的な経験から考えると）ソフトウェアの生産性は期待されるほどには改善されていないと思われる。その主な原因は誤りの本質を理解することなく試行錯誤でデバッグする態度にある。（新しい着想は試行錯誤から生れてくることからすれば決して悪い方法ではないが。）このことは単に第一の目的を満たすハードウェアを作っても，その効果が現れないかもしれないことを示唆している。無駄な試行錯誤を避けるためにはプログラムの理解を助けるソフトウェアが重要であり，コンピュータアーキテクチャはその支援をすべきである。

4　目的実現のためのアプローチ

『EPROM』アーキテクチャ

・概念を，扱う情報の基本単位とし，ある場面・場所で通用した概念はいついかなる場面・場所でも通用するアーキテクチャ。□

プログラムの生産性を向上させるためには，解くべき問題を自分の言葉で説明できる程，十分に素早く理解することが重要であると既に述べた。これはプログラマに，作りつつあるプログラムを自由に調べる能力を与えることで解決できるだろう。従来の計算機システムではデバッガを通してみ可能であった。しかし，デバッガはステップ実行・ブレイクポイント実行と変数の値の読取り・変更操作を基本としており，プログラムのイメージを描くことは依然としてプログラマの頭の中で行う作業であった。しかも，デバッグを考慮したアーキテクチャは希で，デバッガを動かす環境はソフトウェアで作り出すのが普通である。これは従来のアーキテクチャでは実行環境が速い実行を目的として設計されていて，実行が終れば捨て去られていたことに因る。

『EPROM』アーキテクチャは『ある局面で意味ある概念は別の所でも同じ意味を持つ』べきと主張する。実行環境も一つの概念であり，名前を付けて保存でき，これを調べたいとき呼び出して特別な工夫なしに内容の調査ができる。実行環境といえども一種のデータ構造であり特別扱いはしない。重要なことは，ある所で得た知識がいつでもどこでも使えることで，計算機の行動を予測できることになる。つまり，ユーザに一つの計算機の行動モデルを与えることに等しい。自分の頭の中に計算機の行動モデルを一旦組み立ててしまえば，それに従ってプログラムの内容を（ドキュメントなどを参照せずとも）予測を行いつつ素早く理解することが可能になる訳である。万人に使い易い計算機の行動モデルを作ることは難しいから個人の好みを入れて変っていく柔軟性が望ましい。

『EPROM』の名前はEPROM が一旦覚えたことは容易には忘れないところから採用した。これに対し従来のアーキテクチャは実行が終るとメモリを一掃して何もかも忘れてしまうので『RAM』アーキテクチャと呼ぶ。

『EPROM』アーキテクチャの中心概念

・オブジェクトとネーム。□

『EPROM』アーキテクチャは，第一に概念を塊りとして扱うことを主張する。オブジェクトの存在はこれを可能にする。第二にプログラマに時間・空間に渡る一つの大きな世界を提供する必要があり，いつでも好きなときに好きなものへのアクセスを許さなければならない。ネームの存在はこれを可能にする。

・オブジェクトは機能と実行の単位であり，アーキテクチャ上の単位としても有効。□

オブジェクトを意識して設計されたソフトウェアは，クラス（他のオブジェクトを生成する能力を備えたオブジェクトで，生成されたオブジェクトを扱う手続きを定義する。）をインヘリタンス（クラス間でオブジェクトを扱う手続きを融通しあう操作）で体系化された機能単位と捉えることにより，プログラムの内容の素早い理解を可能にする。オブジェクトはプログラムの文面上にある単位というばかりでなく，計算機の資源を消費して計算を実行する実体でもある。したがってオブジェクトの考えかたは抽象データ型のそれを含み，よりダイナミックな概念である。例えばオブジェクトの活動を（外から指令を送って）止めて，（やはり指令を送って）その状態を調べることも自然な形で含められ，プログラムの調査に役立つ。〔文献 Goldberg83 〕

すなわち，オブジェクトはソフトウェアでもハードウェアでもうまく扱える自然な計算単位であり，実行したり記憶しておくこともできるので，『EPROM』アーキテクチャの情報単位にふさわしい。

・ネームは概念を一意に指す。□

概念を的確に他人またはコンピュータに教えるためには，ある単語がその概念を指すことを双方が知っていて，単語を提示することで伝えあう。この単語をネームと呼ぶ。ネームが一意であれば，同じネームは常に同じ概念を指示することになり，『EPROM』アーキテクチャの情報単位にふさわしい。

ネームにはオブジェクトを指さないものも考えられることに注意する。たとえばキューオーバーフローと言うネームは実体のない例外状態を表していて，いつでもどこでもキューへの書込みが失敗したことを意味している。

人間にとって把握しやすい単位はコンピュータにとっても扱い易い単位だろう。このように考えて作られるアーキテクチャを指向アーキテクチャという。オブジェクトを単位にしたときがオブジェクト指向アーキテクチャである。ネームを単位にしたときがネーム指向アーキテクチャである。『EPROM』アーキテクチャの一つの実現法としてオブジェクト指向かつネーム指向アーキテクチャを用いる。

本研究のアーキテクチャを『ORAGA』，プロトタイプを『ORAGA－I』，プロダクトを『ORAGA－HAL』と呼ぶ。ORAGAは，ORiented Architecture to Govern Abstractions の省略形である。

5　ORAGAにおける『EPROM』アーキテクチャ

ORAGAでのオブジェクト指向アーキテクチャ

・オブジェクトの世界の計算は互いにメッセージをかわし，処理できるメッセージを受信したオブジェクトが自分の手持の資源へ変更を与えることで進む。□

オブジェクトはプログラム上の機能単位（抽象データ型）というばかりでなく現実に進む計算の単位でもあることは既に述べた。オブジェクトをプロセッサ対応に割付け，プロセッサ間通信でメッセージを実装して並列動作するマルチプロセッサを考えることは自然な発想であろう。

第一の利点はコンテキストスイッチの頻度を減らせることにある。仮想記憶の場合と似て近い未来に必要になるコンテキストの数は余り多くなく，その集合構成が段々と変化していく参照の局所性があると期待できそうだからである。

第二の利点は物理プロセッサにキャッシュを設けたとき，コンテキストスイッチが起るまで割当てられたオブジェクトにローカル

な資源はキャッシュに置け，共有メモリでのアクセス競合が避けられる点にある。オブジェクトは機能単位なので使用資源はローカルなものが大部分である。

・オブジェクトのハードウェア上の実装はコンテキストである。□

ＯＲＡＧＡでのオブジェクトの実現，つまり計算を実行する論理プロセッサをコンテキストと呼ぶ。コンテキストは物理プロセッサの割当てを受け計算を進める。

・メッセージのハードウェア上の実装は２つのネームによる。□

メッセージは処理内容を指示するセレクタと引数のオブジェクトから成る。一つ目のネームはセレクタと同様に処理を指示する名前であり，もう一つのネームはオブジェクトを指す名前である。ネームについては以下で詳しく説明する。

ＯＲＡＧＡでのネーム指向アーキテクチャ

・すべての事物・概念は一意の名前を持ち，名前で同一性が判断される。□

計算機内のすべての実体には，それを指すアドレスがあるはずである。それが無いものはいわゆるゴミであってガーベッジコレクタなどにより回収されることになる。アドレスが指す実体にアクセスできるのは，そのアドレスを所有している計算主体だけに限るように，さらにアドレスの偽造・改変を完全に禁止するようにアーキテクチャが保障するとき，アドレスは特にケーパビリティといわれる。この方式を採るアーキテクチャをケーパビリティベースアーキテクチャという。一度使用したケーパビリティは再使用せず残す方法が一つの有効な実装法である。すなわちケーパビリティは実体を指す一意に厳重に保護された名前である。〔文献 Fabry74〕

しかしながら，計算機内で厳重に保護すべき情報はこれ以外にもある。例えば，例外要求はプロセッサに取り込まれ割込みベクタを引くために使われ，求められた割込みベクタは要求された処理を実行する命令列を指している。割込みベクタはケーパビリティであるが，例外要求は対応する実体がなく，なんらかの概念を意味している点に注意。例外要求もまた誤って書換えられたとしたら正常な動作は望めない。このように厳重に保護すべき情報をＯＲＡＧＡではネームと呼ぶこととし，ネームを扱う情報の基本単位とするのでネームベースアーキテクチャと称する。ケーパビリティもネームの１つであることに注意。ここでもまた一度使用したネームの再使用を許さないことが一つの実装法である。すなわちネームは一意に厳重に保護される名前であるとする。

ネームはまた同一性の基準を提供する。同じ名前を持つ２つの情報は同じものを意味している。このことはネームの一意性から保障される。

ハードウェアが扱うNameとソフトウェアが扱うSymbolicNameの２つに大別される。

・NameにはCapability,Event,Keyの３つがある。□

Capabilityは，Objectを指すNameである。ただしObjectとは計算を進めるために用いる資源のことである。上の例で説明すれば，割込みベクタがCapabilityであり，起動される割込み処理命令列がObjectである。

Event は，Object内の手続きを間接的に指すNameである。上の例で説明すれば，例外処理要求がEvent であり間接的に割込みベクタが指す手続きの起動を行う。

Key は，Object内のnameSlotを指定するNameである。nameSlotとはObjectが外の世界から受け取る情報（つまり，Name）を格納する場所である。Ada のキーワードパラメタあるいはSmalltalk のセレクタと働きが似ているが，それらが手続き起動要求（call命令）と共にまとめて送る必要があるのに対し，ＯＲＡＧＡ－Ｉではこれらを分離し，引数が準備できるたびに，送り先nameSlotを指定するKey を付け，相手Objectに転送してしまう。手続きの起動は，相手のObjectが十分に引数が揃ったと判断したとき自動的に，あるいは陽にEvent を送ることによる。この分離はアーキテクチャの問題であり，必ずしも一般ユーザから見える訳ではない。

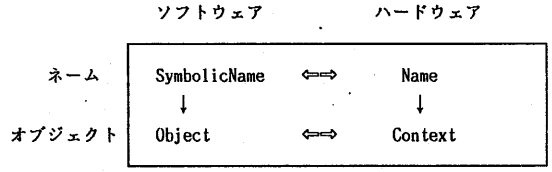
・SymbolicNameは識別子の拡張である。□

ユーザがObjectを指す手段は識別子である。プログラムの識別子やファイル名にあたるが，単に文字列というばかりでなくp-list付きのLisp変数のような構造を持ったものでもよい。

SymbolicNameは，Nameの機能に加えてそれが指すものの意味を暗示させる能力を持つ。これまではSymbolicNameの持つこの自由度はユーザの側に任せられていた。その結果，場合により（気分により）同じものに種々の名が，別のものに同一の名が付けられたりして混乱を招いていた。ＯＲＡＧＡではSymbolicNameの与え方をコントロールし，ユーザに矛盾のない識別子の世界を与える。完全な意味世界を作ることは望むべくもないが一人一人に充分なものは作りうると考えている。ＯＲＡＧＡを個人用に限った理由の一つはここにある。

SymbolicNameを重要視するのはプログラムの品質に強く関係すると考えるからである。名前の一斉の変更はプログラムの正しさとは無関係であるが，プログラムを理解する際に，適切な名前は名付けられた対象のみならずモジュールの機能をも予想させる働きを持つので，付けられた名前の適切さはプログラムの品質の一つの指標となる。SymbolicNameの配分の管理をすることで体系的な識別子の世界をつくれば，プログラムを機能を予想しながら読めることになる。『ＥＰＲＯＭ』アーキテクチャはユーザに一つ計算機の行動モデルを与えることを目標としており，SymbolicNameの配分管理は重要な位置を占めている。

まとめれば次の様な図式で示す関係が成立する。



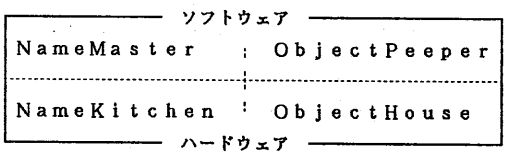
ユーザに見えるのはSmbolicName とObjectであり，それらのハードウェア上の実装がそれぞれNameとContext である。

6　ＯＲＡＧＡ－Ｉの全体構成

ＯＲＡＧＡ－Ｉはソフトウェア指向アーキテクチャＯＲＡＧＡのプロトタイプである。本論文はこの章以降このプロトタイプの

構成要素とその実現手法を述べる。

・2階層＊2部分　＝　4構成要素。□



NameMasterとObjectPeeperはソフトウェアで構成され，ハードウェアで実現されるNameKitchen とObjectHouse が提供する『EPROM』アーキテクチャを助けて，ソフトウェアの生産作業を簡単にする。ObjectPeeperがある指定されたオブジェクトを表示面積の限られたウィンドウへ効果的に表示し，素早い内容調査を可能にするのに対し，NameMasterはSymbolicNameの管理により数多くのオブジェクトから望む機能を持つオブジェクトの素早い選択を可能とする。

NameKitchen はNameを扱う命令セットを実装し，ObjectHouseはObjectを管理する命令セットを実装するハードウェアであり，『EPROM』アーキテクチャの実現を受け持つ。NameKitchen はマルチプロセッサであり，構成プロセッサをKitchenMaid と言う。ObjectHouse も複数のHouseServantからなる。

7　NameMaster

説明

・NameMasterはSymbolicNameの配布をするソフトウェアである。□

NameMasterはSymbolicNameの命名に制限を課し，SymbolicNameとそれが指すObjectの対応を予想できるものとし，Objectを詳しく見ることなしに素早くユーザに内容を理解させうることを狙うものである。この対応関係を完全に保つことはできないだろうが，不完全ながらも使用するに充分なものは作れるのではないのだろうか。この種の制限を課すことには心理的に抵抗が予想されるから，良いユーザインターフェスが必要とされる。モジュールライブラリと一般ライブラリに分け前者への格納はNameMasterの修正を受け承認をもらったモジュールにのみ許し，モジュールライブラリにはコントロールされた名前付けを利用した数々の支援を提供することにより間接的な強制を行う。もともと（名前付けに関し）良いプログラムを書くプログラマにとってはNameMasterは制約にならない。プログラムの正しさには関係ないが良さに大きな影響を持つ制限で成功している例としては，Structured Programmingにおけるコントロール構造の制限，Smalltalk-80システムにおけるクラス分割を方法論とするモジュール分割法がある。特に後者はSystemBrowser と呼ばれるソフトウェアによりその制限をユーザに意識させず課している点に注意する。机上の方法論ばかりでなくこのような優良な支援ソフトウェアが不可欠である。

さらにSymbolicNameは互いの関連をも示すことができる。似たObjectにはその関連を強く示唆する名前をそれぞれに付ける。名前の関連度を定義してモジュールライブラリ内のモジュールをクラスタ分析し機能の大分類をおこない，キーネームを持ちいて詳細検索することなどを期待できる。

・SymbolicNameは構造を持っていてもよい。□

SymbolicNameは従来の例から拾うとプログラムでの識別子やファイル名にあたる。これらの実現は普通は文字列によるが，Lispのp-list付き変数のように構造を持っていてもよい。例えばdateというプロパティの値はSymbolicNameの作成期日を表したり，short，longはそれぞれ省略名，正式名を返したりする。

SymbolicNameの2つの原理

原則1．名は体を表し，かつ，関連も表す。□

原則2．世界は一つ，その内で名は一致の基準を与える。□

原則1はSymbolicNameはそれが指すObjectの機能を予想させ，複数のObjectの互いの関連を示唆すべきと主張し，原則2は同じSymbolicNameを持つObjectは同一であることを主張する。

これら2原則を満たす名前付け法則をネーミングの法則と呼ぶことにする。つぎの2法則が原則から直ちに導かれる。

ネーミングの法則

法則1．SymbolicNameはそれが指すObjectの機能を『必要かつ充分』に予想させる。つまり，SymbolicNameから連想される範囲と表したい概念・機能がほぼ一致すべし。□

法則2．SymbolicNameはそれ全体で『無矛盾』な集まりである。つまり，自由に名前を発生してこれを集まりに加えることはできず，すでにある他のSymbolicNameとの間で調整を必要とする。□

さらに補助則として，

補則3．SymbolicNameは同じ能力を持つ異なった表現を取ってもよい。例えば，省略名と正式名の2表現があってもよい。□

法則1は一般的な概念ほど，ありふれた短い単語で表し，特殊な概念ほど修飾が付いた長い単語を組み合わせて使うことを主張する。基本語として例えばBasic English の単語とコンピュータ関係の特殊な用語，問題の分野の特殊な用語が使える。後者は辞典や実際のプログラムから収集される。

法則2はそれ以前の名付けを破壊するほどの非常に新規な概念が導入されねばならぬときにも耐えられるネーミング規則が必要なことを主張する。木構造のデータベースよりRelationalデータベースの柔軟性が必要である。一斉に基本語を取り替える操作などを必要充分なだけ欲しい。

補則3は場合によって便利である表現が異なることがあるからである。PopUp Menuでは意味の良く判る長い名前が好ましいし，プログラム修正のときは短い名前でも良い。時たま使う名前はわかり易く，何回も現れる名前は一目でわかるようにする。

以上のSymbolicNameはハードウェア上ではCapabilityに写像されるものであり，この他にEvent，Key に写像されるSymbolicNameもある。これらついても上の『Objectの機能』の箇所を『概念』と読み変えれば上と同じ議論が成立する。

理論

ネーミングの法則に従うネーミング規則の実装はこれからであるが，法則1に関し一つの着想を述べる。

あるSymbolicNameがどの位の広くいろいろなものを連想させえるかをある程度定量的に計りたい。この度合をSymbolicNameの連想度ということにする。あいまいさを示しているのではなくありふれ度を意味している。

・基本語のSymbolicNameの連想範囲　＝

（ある実際の一群のプログラムでの）出現確率。□

ただし，同一のプログラム内に現れるSymbolicNameは同じものを指しているので，一回の出現と数える。異なるプログラムに現れ違うライブラリからのSymbolicNameはそれぞれ違うものを指定しているので，それぞれ一回と数える。この出現確率が高いものほどSymbolicNameを見たとき異なるもののどれを指しているのか判断に迷うことになる。つまり，ありふれているのでどれを意味しているのかわからなくなる。

実際のプログラムで実測できない場合は辞書の記述の長さ，項目数で代用することも考えられよう。記述が長いほど重要な単語であり，項目数が多いほどいろいろな局面で使われうる単語である。

・複合語のSymbolicNameの連想範囲　＝

基本語の連想範囲の積　＊　複合の型で決まる値。□

複合語は基本語より的確にあるものを指示できることを意味する。その際，複合の型（動詞＋名詞など）も影響するだろう。

あるSymbolicNameがどのような意味（概念・機能）を持つているかを定めたい。この集合をSymbolicNameの意味範囲と呼ぶ。

・基本語のSymbolicNameの意味範囲＝あらかじめ決められる値。□

・SymbolicNameの意味範囲＝基本語の意味範囲の和集合。□

SymbolicNameの意味範囲を基本語の持つ意味を全部合せたものとする。並置はもとの意味をアンドで結ぶのが自然な解釈である。

以上の連想範囲と意味範囲で与えられた２つのSymbolicNameの類似度を計りうる。

これらがどの程度意味のある指標なのかは不明だが，NameMasterの構築に役に立つと期待している。

8　ObjectPeeper

説明

・ObjectPeeperはオブジェクトを早く理解するためのソフトウェアである。□

現状の問題点

・ウィンドウサイズは限られている。□

高機能ワークステーションの条件の一つはビットマップディスプレイと（多くは）マウスとウィンドウ技術によるより良いユーザインターフェスである。これらの基礎的な研究は商用のよいワークステーションが市場にでていることからも分かるように完成をみている。しかし，ウィンドウにオブジェクトを分かり易く表示する能力についてはもう少し改善の余地があると思われる。普通はウィンドウにそのテキスト表現を単に写しだすだけであり，納まり切れない場合にウィンドウを自動的に広げるか，ウィンドウはそのままでスクロールを行えるかのいずれかであった。ウィンドウを広げる方法は他のウィンドウの情報が，スクロール可能とした方式は自身の情報が損なわれてしまう。この問題は特にウィンドウサイズが限られるとき深刻となる。例えば，デッバッギングのときに関わるすべてのオブジェクトを別々のウィンドウに割り当てて，１ステップ１ステップの変化を刻々見たいときである。プログラムによっては関係するオブジェクトの数が多く一度に全てを眺められるように配置できないこともありうる。

問題の解決

・クラスごとに指示を与えて，適切な表示をする。□

どのようなオブジェクトでも与えられたウィンドウの表示面積のなかにできる限りユーザが見たい情報を優先して写し出すことを考える。上の例では各オブジェクトの注目するフィールドを優先して写し出せば，より多くの情報を一度に見られる。

・指示はパレットでする。□

オブジェクトのフィールド構成は，オブジェクトが属すクラスで決まる。オブジェクトを効果的に表示する指示をクラスにごとにパレットと呼ぶテンプレイトで行う。パレットはいくつかのロットという間仕切りからなる。オブジェクトのフィールドとロットの対応は自由であり，フィールド指定アイコンをペイントボックスから，アイコン引数としてフィールド名をオブジェクトのテキスト表示から選択して，パレットのロット上に配置する。リドローアイコンを選択して実行すればパレットはロットごとに表示アイコンが奇麗に並べられて再表示される。ウィンドウでのフィールドの表示面積はパレット上のロットの大きさに無関係であり，各フィールドの必要面積とウィンドウの面積の兼ね合いできまる。その他圧縮指定のアイコンもロットに載せられる。これは表示面積が足りないとき削ってもよいフィールドを決める。圧縮には，文字のポイントを小さくする，後ろを取り去る，省略形に換える，ハッチにするなどがある。

・アイコンに図ばかりでなく漢字も使う。□

日本語の漢字はそれ自体で多様な意味を包含するのでアイコンにぴったりである。

・ロット間には（表示に関しての）インヘリタンスがあり，指示を省略し，修正しやすくする。□

パレット内のロットには静的に決まる上から下，左から右へのインヘリタンスがあり，親（上，左）のロットのアイコンは子（下，右）のロットにもあると解釈される。特にパレットの最上位置にフィールド指定がないロットを設ければ，オブジェクト全体の大域指示となる。

・オブジェクトの表示はパレットを単に参考にして行う。□

パレット上の指示は不完全であったり矛盾しているかも知れない。指示が全く無いかもしれない。表示プログラムはいついかなるときも過不足を補いつつ表示する。単なるパレットの忠実なインタプリタではない。

・ズームアップはオブジェクトごとの適切な表示を可能にする。□

オブジェクトに属すクラスのパレットを原本として，一時パレットを作り，残したいロットを指定した後ズームアップを行うと，消え去るロットのアイコンがインヘリタンスチェイン上の子のロットへ移り，元のパレットのロットの区切方に残すように，一時パレットが書換わる。その結果，対象のオブジェクトの表示が新しくなる。パレットの書換えが上記のように行われるので表示は大きく変化しない。

・圧縮されたフィールドを『読み取る』こともできる。□

圧縮を受けたフィールドはズームアップにより拡大する他にボイスジェネレータを使い，まさに読み取ることができる。テキスト表現，例えば数字はそのまま音声に変換できる。困難な場合や長過ぎる場合はその旨を報告すればよい。

以上のようなインヘリタンス付きのテンプレイトを用いることにより，数多くのオブジェクトを数多くのウィンドウ上に，ユーザが見たいものを主に残りは必要なら圧縮して効果的に表示できる。

例えば，デバッグ時に必要な部分をステップ動作で監視でき，プログラムを理解するうえで大きな役目を果たすと期待される。

9 NameKithcen

説明

・NameKitchen はNameを扱うアーキテクチャである。□

構成

・NameKitchen はKitchenMaid 単位のマルチプロセッサである。□

なぜ，マルチプロセッサにするのか？

第一に，オブジェクトはハードウェアにとって自然な計算単位である。物理プロセッサであるKitchenMaid を複数置き，オブジェクトを割当てることをすぐに思いつく。（とはいえ実現例が多いというわけでもない。iAPX432 が有名。）近い未来に参照されるオブジェクトは余り多くなく入れ代りも小さい，すなわち参照の局所性が期待できよう。メッセージを処理した後でも，できる限りコンテキスト（ＯＲＡＧＡでのオブジェクトのハードウェア上の実装）をKitchenMaid に残せば，近い２度目のメッセージの処理はコンテキストの無駄なスイッチなしに実行を再開できる。（iAPX432 はこれとは異なる。プロセッサへの割当て単位はタスクであり，メッセージの伝達はキューに依る。）待ち時間の内にコンテキストを退避しておけば，KitchenMaid を他のコンテキストに譲り渡すとき，退避を省略できる。

第二に，オブジェクト間のメッセージ交換をコンテキスト間の通信で行うことも自然な発想である。コンテキストが活動中もメッセージを受け取ることができるとすると，プログラマブルな割込みをメッセージ機構で実現できる。例えば，『catchMeIfYouCan 』を宛名とするメッセージはアイドルしているKitchenMaid なら誰でも受け取れるようにできる。

RISCのようなレジスタバンクを切替えるシングルプロセッサでもコンテキストスイッチは少なくなるが，待ち時間中の退避や，プログラマブルな割込みを自然な形で採り込めない。

構成例

・KitchenMaid をリング状に接続する。□〔図9.1，図9.2〕

リング内をパケットが流れる。パケットの本体はリングアドレスとリングデータである。リングアドレスは送り先コンテキストのCapability（とロックを外すKey 次章参照）であり，リングアドレスはメッセージを構成するKey とCapabilityである。KitchenMaid はリングアドレスを常に監視していて自分宛のパケットを取込み処理する。リング管理を実行しているKitchenMaid はリングを監視し，受取り手のいない周回パケットを取込み次の処置をとる。アドレス部が『catchMeIfYouCan 』でデータ部が周回パケットのアドレス部のパケットを流し，（少し後に）先の周回パケットを送り出す。一方忙しくないKitchenMaid の一つが最初のパケットを取込み，必要なら実行中のコンテキストを自ら退避して空きとなり，周回パケットの宛先のコンテキストを（ObjectHouse ）から呼び出して実行を再開する。遅れて来る周回パケットには今度は受け取り手がいる。

メッセージ機構

・KitchenMaid はNameを扱う命令セットを定義する。□

KitchenMaid は物理プロセッサであり，割付られたコンテキストを実行する。コンテキストは実行準備段階あるいは実行中のオブジェクトのアーキテクチャ上の実装に相当する。ＯＲＡＧＡではオブジェクト内の手続きの実行は以下のように進む。

1．コンテキストの生成

（最初にこのオブジェクトが存在しないとき）所属のクラスにあたるオブジェクトをコピーしてインスタンスを生成する。生成オブジェクトのCapabilityを配る。

2．引数の結合

送り手（受け手と同じでもよい）はCapabilityを用いて相手コンテキストへKey と共に引数のNameをリング経由で送る。Key は結合すべき位置を示す。単に数による相対位置でなく，Key の一致による絶対結合である。

3．手続きの発火

コンテキストは要求された引数の組が揃ったとき呼び手側とは無関係に自動的に発火起動する。あるいは呼び手側がEvent をリング経由で送り，陽に発火を要求してもよい。『catchMeIfYouCan 』がEvent の例である。

4．手続き実行中

コンテキストは実行中においてもリングを監視しており自分宛のパケットを見つけしだい取り込む。即時の手続き起動を要求するものであれば現在実行中の手続きを止め（call命令かgoto命令で）要求された手続きの実行を開始する。

5．手続きの終了

最終命令で自らに『IwIlBeBack』Event を直接起こす。コンテキストは自らを（ObjectHouse へ）メモリの空き時間を利用して退避し，アイドル状態となり，メッセージの到着を待つ。（2か3へ）

6．コンテキストの消滅

最終命令で自らに『IamTooTired 』Event を直接起こす。コンテキストは自らを消去し，KitchenMaid を空とする。

7．ＯＳ機能

受取り手のいない周回パケットを見つけ，宛先のコンテキストにKitchenMaid 割当てを要求するパケットを作り，送出する。これが失敗したときは他の適当なコンテキストからKitchenMaid を取り去ることを要求するパケットを送出する。

8．ＯＳ機能

コンテキストの割当てを要求するパケットを受けたとき，コンテキストは（必要なら自らを退避した後，）そのコンテキストを（ObjectHouse から）復帰する。ただしコンテキストが既に消滅しているときは，Capabilityの使用者に『TooLate 』Event を送る。

メッセージ機能

・上記のメッセージ機構で４つの要求が処理できる。□

コンテキストという明確な単位があり手続き実行中もメッセージを受け取ること，メッセージをネーム単位で授受することで，次の４つのコントロールトランスファが一機構で実現される。

1．手続き呼び出し（外部から，引数付き）。
2．割込み要求　（外部から，引数無し）。
3．分岐　（内部から，引数付き）。
4．例外要求　（内部から，引数無し）。

機能分割

・KitchenMaid は４つの構成部分に分かれ並列に動作する。□

1．Ins.プロセッサ・Ins.メモリ

命令プロセッサとそのローカルメモリで他の３つのプロセッサのシークエンサである。

2．Atr.プロセッサ・Atr.メモリ

属性プロセッサとそのローカルメモリで属性の計算を行う。

3．Val.プロセッサ・Val.メモリ

値プロセッサとそのローカルメモリで値の計算を行う。

4．Aux.プロセッサ・Aux.メモリ

補助値プロセッサとそのローカルメモリで補助値の計算を行う。

属性は主にEvent とKey であり，値は主にCapabilityである。属性の計算を値の計算と並列に行うことがＯＲＡＧＡのネームベースアーキテクチャの特長である。補助値はプログラマブル割込みの計算に使うことを考えているがHouseServantを助ける計算にも使える。〔図9.3〕

１０　ＯｂｊｅｃｔＨｏｕｓｅ

説明

・ObjectHouse はObjectを管理するアーキテクチャである。□

構成

・２つの管理表と記憶域がある。□

ObjectHouse はObjectを格納する記憶域と，その所在を管理する２つの表sysDirectoryとsysDictionary からなる。Objectは２つのName, Key とCapabilityによってアドレスされる。Capabilityは Objectを指すがKey は一種のパスワードである。

・sysDirectoryは，エントリ

Lock	Self	物理アドレス

がCapabilityでアドレスされた表である。

・SelfはCapabilityでハッシュしたときに用いるエントリである

・LockはKey とその一致が要求されるフィールドであり，一致してはじめて物理アドレスが指す記憶域のオブジェクトを扱える。

・sysDirectoryはKey とCapabilityをアドレスとして検索でき，オブジェクトへの物理アドレスが得られる。

・sysDirectoryにはKey,Capability，物理アドレスを一エントリとして登録できる。□

・sysDictionary は，エントリ

Name1　Name2	Name3　Name4

が最初のネームペアName1,Name2 でアドレスされた表である。

Name1,Name2,Name3,Name4 はNameであり，２つのネームペアとして扱かわれる。

・sysDictionary はネームペアName1,Name2 で検索でき，値ネームペアName3,Name4 が得られる。

・sysDictionary にはネームペアName1,Name2 とName3,Name4 を一エントリとして登録できる。

・sysDictionary は常に既約された状態になっているとする。既約状態とはもはや既約操作ができない状態である。既約操作は，値ネームペアをそれをアドレスとしてsysDictionary を検索して求めて得られた値ネームペアに置き変える操作である。□

アドレス機構

・sysDictionary とsysDirectoryの並列引き。□

アドレッシングをコンテキストContext 下でアドレスとしてKeyとCapabilityで行うとき，

DO（sysDictionary をネームペアContext,Capabilityで検索。）

IF（値ネームペアKey+ ,Capability+ が求められたら，）

THEN（sysDirectoryをネームペアkey+,Capability+で検索。）

ELSE（sysDirectoryをネームペアKey, Capability で検索。）

DO部とELSE部は並列に動作できる。

意味

・sysDirectoryはロック付のオブジェクトディレクトリである。□

sysDirectoryはパスワード付のファイルディレクトリに当る。違いはパスワードもアドレスもネームでファイルがオブジェクトへ一般化されていることである。パスワードを付けたことでアドレスのネームを広く配ってもパスワードの所有者のみにオブジェクトへのアクセスが許される。

・sysDictionary はアドレスチェインを記憶する表である。□

sysDictionary は，sysDirectoryを検索するためのアドレスチェイン即ち，アドレスのアドレス，アドレスのアドレスのアドレス，…を記録する。既約機能があるので実際にはアドレスのアドレスのみが登録されている。しかし，既約機能を利用する他の情報も置ける。

sysDictionary は共有され自由に読み書きできる黒板に当る。KitchenMaid はアドレスチェインを登録できる。これらのアドレスが既約機能により短絡されてアクセスパスが繋がる。アクセスパスを得るKitchenMaid のアドレスチェイン登録は参照宣言（import），提供したKitchenMaid のアドレスチェイン登録は公開宣言（export）とみなせる。

分散

・共有メモリへのアクセス競合を押える。□

1．機能分散は可能。

属性に分類できるのはsysDirectoryのLock，sysDictionary のName1,Name3 のエントリフィールドであり，値に分類できるのはsysDirectoryのSelf，sysDictionary のName2,Name4 である。KitchenMaid は命令，属性，値，補助値の４部分にわけられるが，sysDictionary ，sysDirectoryの属性，値部分はそれぞれ属性，値プロセッサからのみアクセスされる。表を構成するメモリを完全に２つの部分へ分けられることを意味する。

2．負荷分散は難しい。

sysDictionary の分割。

アクセス機構からわかるように，リードアクセスからみるとsysDictionary はName1 がコンテキスト名と一致するエントリを各KitchenMaid へ分散することができる。sysDictionary へのライトアクセスはアクセスチェインをsysDictionary へ登録するときにおこり，このとき既約機能もふくめてsysDictionary の更新をする必要がある。ライトアクセスはリードアクセスより，頻度が小さいと期待できる。

sysDirectoryの分割。

困難。sysDictionary のようにリードアクセスに限っても分散できる局所性はみつからない。当面一つにする。その理由は，

・ＯＲＡＧＡ－Ｉは個人用のソフトウェア開発マシンであるから，個人が使用するに十分な速さがあればよい。

・ワーキングセットより多いKitchenMaid を持てばコンテキストスイッチの回数は小さく，各KitchenMaid はローカル資源をキャッシュしているのでアクセス競合が押えられる。

１１　ＮａｍｅＭａｓｔｅｒの現状と議論

実験

・現実のプログラムからデータを集める。□

Ｃ言語，Smalltalk のネーム，コメント，ストリングを抽出するレキシカルアナライザを作り，名前の統計を解析中である。各言語のパーザも作った。パーザにより関数定義，変数型ごとの構文要素を考慮した解析が可能になるからである。ネームの長さの分布，出現度の分布などを調べてみた。調査中であり，予備的な結果を１つ述べるに留める。

・（予約語も含め）多くのプログラムでおおよそ上位１０％のネームで５０％の出現回数を，説明する。上位５０％のネームで９０％の出現回数となる。□

この数字はプログラムによらずネームの出現回数に関してなんらかの法則があることを予見させる。実際，Zipfの法則の一つの例になっているようである。多くの分野で観察される経験則であり小説などの言語統計に見られる。ソフトウェアという狭い範囲でも成り立つ点が興味深い。〔文献 Ziph49 〕

図11.1はSmalltalk Virtual Image Version.2 を解析した例で，

ネーム（セレクタも含む）の種類		5182種
同じく	総出現回数	82653回
同じく	最大長	38.0文字
同じく	平均長	10.7文字　であった。

Smalltalk のネームの付け方は独特で語を大文字で始めてつなぐ。長いが意味はわかり易い付け方である。平均10文字長で，平均16回出現する。最初の数語を除けばZiphの法則にあう。

Smalltalk の世界は数多くのクラスをユーザに公開している。多くの名前が機能や互いの関連を示すように系統的に付けられているはずである。その方法を調べることは有意義であろう。

１２　ＯｂｊｅｃｔＰｅｅｐｅｒの実験

実験

・１９８３年度の卒業論文として実装を行った。□

（東大電気工学科４年　阿部雅彦君による）

１．装置

BBN Computer BitGraph terminalを Vax 730／UNIX 4.1bsd に9600bps で結んでTSS で使用している。1024×768 の縦型のモノクロビットマップディスプレイと機械式マウスを装備している。

２．この研究の目的

与えられた大きさのウィンドウに不必要なフィールドを省略するなどして，オブジェクトのわかり易い効果的な表示をする。パレットは省略可能なフィールドをユーザが記述するスプレッドシートである。これらの概念は，8.ObjectPeeperの章で既に説明した。

３．操作の方法

パレットを作る。

マウスで２点を指定して作る。

パレットをロットに分割する。

２分割したいロット上に縦用あるいは横用の区切りアイコンを載せることによる。（図12.1）

ロット上にアイコンを載せる。

アイコンはフィールド指定，ズームアップ選択，圧縮法指定など種々の用途の指定に用いる。アイコンはロットのどこに置いてもよい。粉らわしいときは，リドローを行えばパレット全体が書き直される。このときインヘリタンスも矢印で示される。（図12.2）

作成したパレットを用いたオブジェクトの表示。

望みの大きさのウィンドウを開き，オブジェクト名を指定する。ウィンドウが必要表示面積より小さいときは，圧縮アイコンが指定されたロットから圧縮を適用していき，必要面積を調節した後，表示する。

オブジェクトのズームアップ表示。

ズームアップはロットを選択しズームアップを指示することによる。インヘリタンスがきちんと処理される。（図12.3）

１３　ＮａｍｅＫｉｔｃｈｅｎと ＯｂｊｅｃｔＨｏｕｓｅに関する議論

意義

・オブジェクト指向アーキテクチャの一つの実現を示した。□

オブジェクト指向アーキテクチャは抽象データ型言語，オブジェクト指向言語で記述されたプログラムの持つ特質，特に保護・並列を活かすアーキテクチャだといわれてきた。インテルのiAPX432，IBM のSystem／38（２つとも商用機！）はオブジェクト指向とはいってもケーパビリティベースであり，ネームベースのＯＲＡＧＡのほうがよりオブジェクト指向を強めたアーキテクチャである。

検討

・データ駆動アーキテクチャとの比較。□

第一の違いは，データ駆動方式がデータをコピーしてネットワークを流し，コントロール駆動の同期を排除する結果として副作用がない計算方式なのに対し，オブジェクト指向方式はデータつまりオブジェクトを共有し，その状態をメッセージで変えていく副作用を積極的に使う計算方式だというところにある。データ駆動方式ではコピーは時間がかかり領域も必要なので，副作用を表にみせないようにしながら共有をいかに取り込むかが課題になる。オブジェクト指向方式では共有のためアクセス競合が無駄な待ちを生むので，一致を保ったままいかに分散を進めるかが課題となる。

第二の違いは，データ駆動方式の計算単位は命令であり，計算は命令が次々と次の命令を陽にチェインしていくことで進むが，オブジェクト指向方式では計算単位はオブジェクトであり，計算はメッセージ交換で進み，オブジェクトは必要がなくなるまで残っている点にある。ＯＲＡＧＡのオブジェクトの実現はコンテキストであり，コンテキストはいつでもメッセージを受け取ることができる。

14 まとめ

ソフトウェアの作成作業を助けるコンピュータアーキテクチャを求めることを目標として設定した。

プログラムで解くべき問題と，作りつつあるプログラムを自分のものとして理解することが重要であるとし，プログラマに計算機の行動モデルを与える，『EPROM』アーキテクチャ（ある時ある場所で通用したことはいつでもどこでも使える）を作れば良いと考えた。

さらに『EPROM』アーキテクチャをオブジェクト指向アーキテクチャとネーム指向アーキテクチャに分解した。このアーキテクチャをORAGAと呼ぶ。ORAGAを4つの部分，

NameMaster　ObjectPeeper
NameKitchen　ObjectHouse

に分けた。

NameMasterは，モジュールの品質を名前付けの観点から判定し，良い名前を推薦し，品質を改善することを狙うソフトウェアである。名前付けがコントロールされたソフトウェア群には統計の手法を適用して多様なサービスが提供できるだろう。

ObjectPeeperは，オブジェクトを大きさが限られたウィンドウに効果的に表示するソフトウェアである。クラス対応にパレットを用いて望ましい表示効果を指定する。パレットにはインヘリタンスがあり記述が楽にできる。ズームアップによりオブジェクトごとに最適な表示にもできる。

NameKitchenはネームを扱うハードウェアである。オブジェクトがハードウェアにとって自然な計算単位であるとし，各オブジェクトを物理プロセッサに割当てるマルチプロセッサとする。メッセージ通信はプロセッサ通信で実装される。

ObjectHouseはオブジェクトを管理するハードウェアである。オブジェクトの所在表とオブジェクトへのアドレスチェインの表の2つを持っている。オブジェクトへのアクセスにはキーを必要とし，キーの配布を管理することによりアクセス権のコントロールができる。

以上4つの構成要素の実現に際しての着想を述べた。ObjectPeeperを除き設計の段階にあるが，本論文で提案したアーキテクチャは従来になかった独自性の高いものであり，紹介するに十分であると信じる。

文献

Fabry74 "Capability-Based Addressing"
CACM Vol.17 No.7 pp403-412 (July 1974).

Goldberg83 "Smalltalk-80 : the Language and its Implementation
Addison-Wesley Press. (1983).

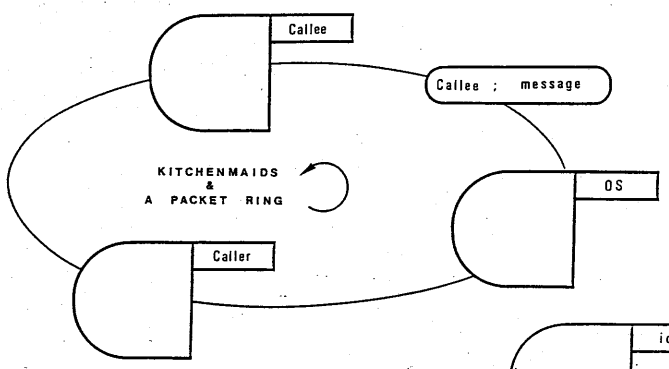
Ziph49 "Human behavior and the principle of leasr effort"
Addison-Wesley Press. (1949).

神田他　抽象データ型支援アーキテクチャの一考察
情報処理学会全国大会　(昭和57年後期) 1F－7.

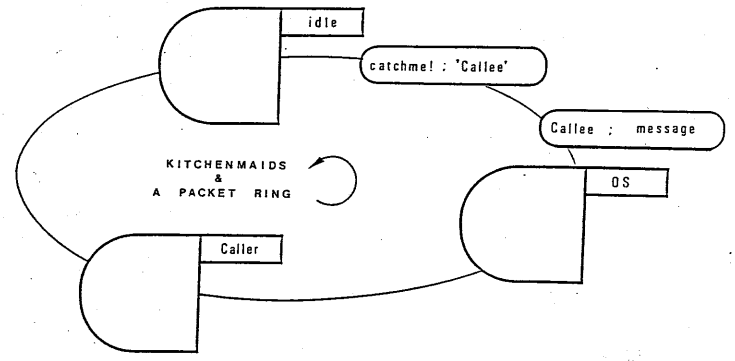
神田他　抽象データ型支援アーキテクチャの一提案
情報処理学会全国大会　(昭和58年前期) 4N－8.

神田他　抽象データ型支援計算機におけるネーミング機構
情報処理学会全国大会　(昭和58年後期) 3P－1.

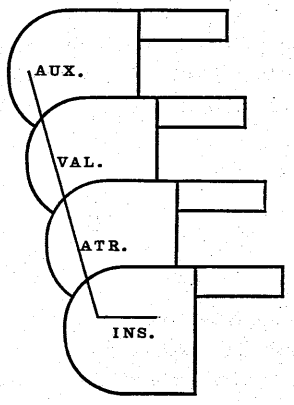
神田他　ソフトウェア作成支援のための
ネームベースアーキテクチャとその実現法
情報処理学会全国大会　(昭和59年前期) 1F－6.



⇐ 図9.1　パケットに受け取り手がいるとき



⇒ 図9.2　パケットに受け取り手がいない，
コンテキストスイッチが起る



⇑ 図9.3 4つの部分からなるKitchenMaid

NameMaster

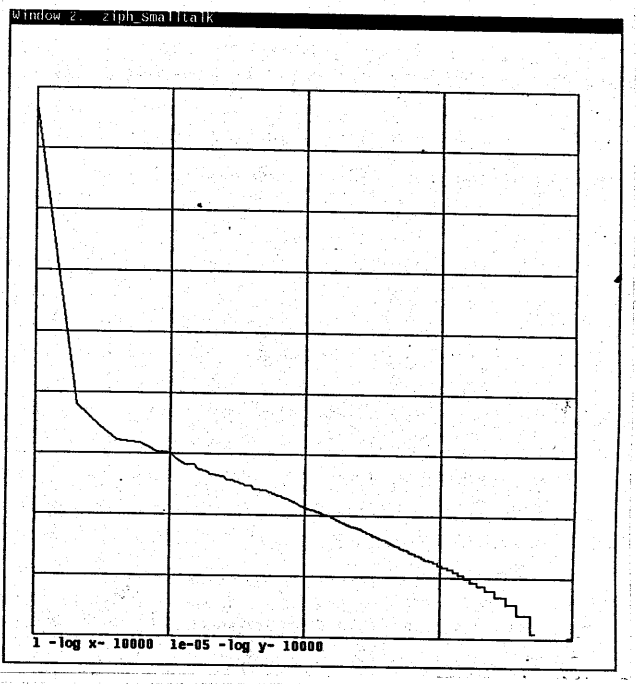
Smalltalk Virtual Image Version 2. Source

5182 word
82653 occurrence

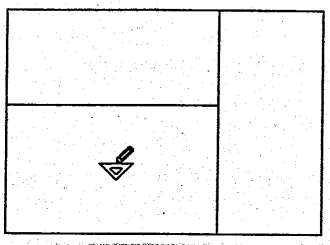
・Ziphの法則

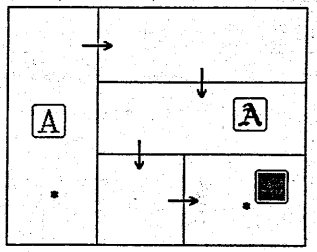
Z (Ri) = ネームRiの出現回数／総出現回数．ただし
Riは出現回数が多い順から i番目のネーム
の log (i) 対 log (Z (Ri)) のグラフの傾きは－1□

No.1 5287 self
No.2 2299 ifTrue:
No.3 1389 methodsFor:
No.4 1281 at:
No.5 1253 ifFalse:
No.7 882 aStream



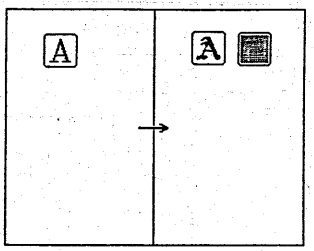
⇒ 図11.1 Ｚｉｐｈの法則



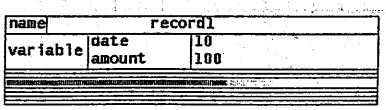


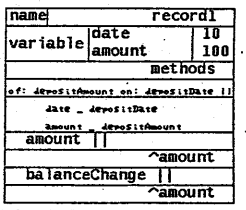
⇐⇐ 図12.1 パレットの分割

⇐ 図12.2 アイコンと
インヘリタンス



⇑ 図12.3 ズームアップ





⇐ 図12.4 オブジェクト
の表示２例