2° Name 概念の導入 —— Capability の自然な拡張として

Object とはデータとそれを操作する手続きを不可分と考えカプセル化したもので自由に生成消滅できる。Object の外の手続きはデータを直接操作できずデータの保全性が高まる。Capability とは Object を一意に指す保護されたアドレスである。偽造改造が禁止されているから Capability の所有者以外は Object を操作できず，一意的であるので所有者は誰でも Object を（決められた操作を使って）扱える。しかし、一意的に保護すべき情報は Object のアドレスだけではない。このような情報を抽出してアーキテクチャで一意的な保護を実現することが、アーキテクチャ設計の一つの方向であろう。この考え方は Capability の自然な拡張といえ，Name と総称する。

3° 抽象データ型計算機における Name —— 3種の Name

現在設計中の計算機においては、Name は Capability，Event，Key の3種である。Key は性質と使用時条件を一意に決める Name である。Object が実引数として Context（Module のインスタンス）内の仮引数（実はこれも Key）に結合される際に，Object の性質と使用時条件を示すためにいっしょに提示される。結合は実引数と仮引数の Key 同志の一致による。Event はプログラムのコントロールフロウの遷移時要件を一意に指定する。Context 内ではエントリ名（実は Event）と手続きが対応づけられている。ソフト，ハード的に発生された Event は合成されていき，遷移が必要になった時点での合成 Event と Context 内の Event との照合が行なわれ，一致したエントリの手続きへ実行が移る。実は実引数と仮引数の結合成功は一つの Event 発生であり，Context へ充分な数の引数の結合が生じた時点で遷移が要求され，照合が成功した手続きへ実行が移る [2]。

4° Name を広く使うことの意義

情報を一意に指定する Name を用いることにより，最終目標である「プログラミングの生産性を向上すること」が可能となる。Capability の利用はプログラムの資源（Object）の調査変更を可能にする。Event の使用はプロセスの進行状況の把握を可能にする。Key の使用は Module のインターフェスの細かい指定を可能にする。

なお、Context を前もって生成しそれに引数を結合する方式では、これから発火する Module の手続きに必要な引数の準備状況を陽に表わせるので、従来のようにプログラマが頭の中で引数の揃い具合を考える必要がなくなる。（Object の状況を小さな Window に視覚的に表示する ObjectPeeper を考慮中。）アーキテクチャの設計時において早い時期に Context の発火が予想できる性質を利用して，計算資源（Processor）を効果的に割り付けられる利点も考えられる [1]。

5° Nameの管理 —— 一般論

これまで述べてきたNameはシステム内で使われる呼称であり、ユーザがプログラムの中で使うSymbolicNameとは異なる。SymbolicNameとNameは必ずしも1:1に対応しない。同じ(違う)SymbolicNameが異なる(同じ)Nameへマッピングされるかもしれない。プログラミング支援環境はこれらの対応を管理する"知識"ベースを持ち適切な変換を保障しなくてはならない。† この機構を外部Naming機構と呼ぶ。

また、Nameを解釈する機構が必要である。Capabilityを Objectに結び付け, Eventに従いコントロールを移し, Keyの照合をする機構である。これを内部Naming機構と呼ぶ。

6° 抽象データ型計算機におけるName管理

外部Naming機構の実現をNameMasterと呼ぶ。Capability, Event, Keyの内部Naming機構をそれぞれNameMapper, NameRegister, NameCouplerと呼ぶ。これらの総称はNameManagerである。以下個別に説明を加える。

7° NameCoupler —— Keyの管理者

現在想定している照合ルールは値の一致である。適切にKey値を決め配置する作業はコンパイラー(と知識ベースへの問合せ)が果たす。もっと動的なKeyの解釈が可能なようにダイナミック[アンコリンキング機構が望まれる。

8° NameRegister —— Eventの管理者

NameCouplerと同様なことがいえる。

9° NameMapper —— Capabilityの管理者

NameMapperの機能は, オブジェクト指向仮想記憶とかオブジェクトファイリングと呼ばれるものに他ならない。既にいくつかの設計例がある[3][4][5][6][7]。

設計の際考慮すべき点は,

□ 必要なマッピングテーブルをコンパクトに実装すること, 高速に引けること
□ Nameの表現はコンパクトにし, 内容の変更は極力要しないこと
□ Objectの表現はコンパクトにし, スワップはObject又はObjectGroupで可能なこと
□ 仮想記憶内の置き換え則, ガーベッジ回収方式, クラッシュ対策との連係法

設計中の計算機は個人用ゆえにアイドル時間の利用が図れる。設計指針は次の通り。ガーベッジコレクタなどのNameManagerデーモンを走らせる。Nameは短いビットパタンとし, 再使用を許し, テーブルを用い(即ち、間接ポインタ方式), 一つのObjectを指すNameが複数あってもよいとする。NameはObjectの存在場所によって内容を変えない(one-level address)。なお、Objectの存在場所を導き出すテーブル — Directory — の他にContextに応じてNameを修飾するテーブル — Dictionary — がある[2]。

10° NameMaster —— SymbolicNameの管理者

簡単な実現方法は、ユーザにかわり適切なSymbolicNameをNameMasterがユーザと対話して発生し, SymbolicNameとNameを1:1対応に保つことであろう。NameMasterはSymbolicNameを系統的にまた意図を表明するように付けるのでプログラムの読み易さも向上する。例えば重要でない変数名はめだたない名とする。完全な解決は難しいが、AIの知識の利用を考えたい。

[1] 25回情報処理全国大会 1F-7 [2] 26回情報処理全国大会 4N-8
[3] *the Cambridge CAP North-Holland (1979)* [4] *Hydra/C.mmp McGraw-Hill (1981)* [5] *Smalltalk Byte (Aug. 1981)*
[6] *Smalltalk-80 Addison-Wesley (1983)* [7] *A Programmer's View of the intel 432 System McGraw-Hill (1983)*

† 米国でAdaの支援環境に知識ベースの利用を試みる研究が始まると聞く。