5T-4 "DinnerBell+NameMaster+ObjectPeeper"による統合プログラミング環境 － 全体構成

神田陽治、土井晃一、田中英彦、元岡達
東京大学・工学部

1° 焦点 － 個人用途なプログラムを素早く作る技術

基本的な問題ほど，核心を突く解決が必須であり，真に創造的な思索が要求される。プログラムを正しく作る技術もこの一つで，その難しさは，ソフトウェア工学という学問を成立させた事実からも推し量ることができる。

知的生産ツールとして，コンピュータが個人の領域にまで定着しつつある今日，各個人が直面する多様な情報処理要求（経営管理から量子力学の計算まで）をこなすプログラムを，状況を考慮して素早く作る技術が，新たな焦点となってきた。よくあるように，簡単なプログラムにまる一日を費やすようでは，何ものも望むことはできない。（それで仕事をしたと思う錯覚も招き易い。）

2° 解答 － ORAGA 計画

我々は，この課題に過去3年間[1-3]，色々な角度から検討を進めてきた。問題の根本から始め，その解決は常に革新であるように努めてきた。結果は，ハードウェアからユーザインタフェスに至る総合的な計画である。

この報告の前半で，我々の方法を支える事項を確認し，残りで，計画の各部の概略・特徴と互いの連携をまとめる。

3° 基点 － アプローチ概説

① プログラミング言語ではなく，プログラミング作業自体を支援する。

高級言語マシンのアプローチは，「最新の技術を盛り込んだ最新のプログラミング言語の実行支援をすれば，全て良し。」で終わることが多かった。しかし，プログラムの作成には，言語の知識だけでは不十分で，アルゴリズムやライブラリを使いこなす必要がある。モジュール化を中心としたプログラミングの作業それ自体（言語支援も含んで）ORAGA では支援する。

② ハードウェア、ソフトウェアのステップの機能向上を果す。

ソフトウェア工学の実験が教えるところでは，人間の処理能力は基本ステップあたりでほぼ一定である。人間は複雑な事柄を抽象して理解する。この塊りがステップであり，生産総量の増加にはステップの機能向上が必要となる。

4° 主張 － ORAGA テーゼ

① 記号†であるネームと，実体であるオブジェクトの重視。

我々は実体を「記号」を通して操作している。プログラムに出現する識別子を一斉に取り換えると，実行の結果は同じでも，読み易さは大きく影響される。従来，無視されてきたプログラムの記号的扱いを，コンピュータ処理の俎上に載せることは，計画の中で最も挑戦的な技術開発である。

実体の処理単位（3°の②のステップ）をオブジェクトの処理にする。オブジェクトの概念は，情報隠蔽・抽象データ型と発展してきた，モジュール化規準の自然な延長上にある。

② 並列処理は、プログラミングを簡単にする。

並列処理導入の目的は，多く実行効率の向上にあった。ORAGA では、

† Lisp などでの記号処理でいう記号ではなく，記号論でいう記号である。

並列性を十分に整備されたツールを前提に，プログラミングに利用する。 動かないプログラムを目の前に，コンピュータの中が見えたならと誰しも願ったことがあるだろう。 対象とするプログラムにできる限り手を加えないで，有用な情報を引き出す鍵として，並列処理を使うことができる。

5° 実現 － **ORA6A** 4つのサブシステム

① **DinnerBell** 並列オブジェクト指向システム記述言語 [4]

オブジェクト化は，モジュール化規準であり，データと関連の操作及び実行環境をパックすることを勧めている。 この結果，オブジェクトは基本的に副作用を持つとみなすべきで，従って（実行環境を持つから可能な）オブジェクトの並列動作を統制なしに行うと，実行結果は非決定的になってしまう。 **DinnerBell** には，並列性を高く保ちつつオブジェクトの並列実行を決定的にする新技術が組み込まれている。

② **NameMaster** 記号であるネームの管理者 [3]

記号には実体を強く想起させる能力があり，この能力の利用を目的に，プログラムに出現する識別子を記号として扱うことが狙いである。 自然言語パーザは文を，文法と語彙に基づいて，意味表現に変換する。 現在のコンパイラはプログラムを，文法に基づいて，実行に必要なだけの意味表現に圧縮する。この観察から，目的達成のためには，プログラム用の語彙を設定し，この語彙と文法に基づいて，プログラムをより精密な意味表現に変換する。

プログラムの記号性を重視する結果，プログラム全体を通して，意味が直観的に了解されると期待できる。 日常言語の範囲で，「私はカモメ」と言って理解されニュースになるのは，テレシコワさんだけである。 プログラムで，clear(Counter1)はすぐ了解できるが，injure(Counter2) は手続き injureの本体を何回も調べる羽目になる。 プログラム語彙を（個人ごとに）設定し，プログラム全体で意味の通ることを保障することで，可読性が高まるのである。 従来のタイプ宣言・検査は，主にコンパイラのための可読性向上技術であった。

③ **ObjectPeeper** 実体であるオブジェクトの表示・修正の管理者 [5]

オブジェクトをビットマップウィンドウに，効果的に表示することを狙う。 スクロールの手間を省くために，圧縮表示の技術を開発した。 また，オブジェクトの実行履歴をデータベース化し，問合せ言語で色々な局面からの情報を得る手法で，並列処理下での有効なデバッガとすることも検討している。

④ **Dragaltself** データ駆動方式・マルチプロセッサ実行機構 [6]

DinnerBell は単一代入言語でもあり，その並列性を生かす実行機構である。

6° 統合プログラミング環境 － **"DinnerBell+NameMaster+ObjectPeeper"**

経験によれば，プログラム作成で無駄な時間と感じるのは探索に費す時間である。 「虫」の居所の探査や必要な機能を持つモジュールの選択などに，そう感じる。 時間短縮には，良いユーザインタフェースが不可欠である。 虫の発見に，オブジェクトを調べる **ObjectPeeper** が役に立つだろう。 モジュールの素早い検索とプログラムの読解に，**NameMaster** が効果を示すだろう。

[1] 情報処理学会 WGSF 11-3，大会 1F-7(25), 4N-8(26), 3P-1(27), 1F-6(28), 2R-6(29)
[2] 電子通信学会 EC-83-55 [3] 日本ソフトウェア科学会第一回大会 1C-3
以下，本論文集 [4] 3R-6 [5] 5T-5 [6] 3R-7