プログラミング言語　11－3
（１９８７．５．２２）

第４世代言語の構造

石田康勝
コンピュータソフト（株）

第３世代言語までの言語は、コンピュータ自体の機能と低レベルのシステム・ソフトの機能を代表するのに対し、第４世代言語はアプリケーション開発／運用維持管理上必要な、よりハイレベルの機能を代表するものであり、ローレベル機能群、ハイレベル機能群と言語の間を、ゼネレータ、インタプリータ、（中間コード）コンパイラなどのインタフェイスで接続した構造をもつ。第４世代言語に関して非常に重要なのは、完全性と統合性である。第４世代言語のもつべき主要なハイレベル機能の要点については本文中で解説してある。

Architecture of 4-th generation language systems.

Yasukatsu Ishida
Computer Software Organization Inc.
17F Shinjyuku-Sumitomo Bldg Nishi Shinjyuku Tokyo 163 Japan

A fourth generation language system has architecture to interface between highly application orinted descriptions and internal functionnalities by a generator , an interpreter , an intermedilate code compiler or a compiler while a third generationlanguage represents the functionalities of a computer itself or a low-level system software.

The very charateristics which make a fourth generation language truly usefull are completeness and uniformity. The descripthons about the important high-level functionalities for fourth generation language systems will be found in the text.

第 4 世 代 言 語 の 定 義

第3世代までの言語　　基本的にはコンピュータ自体の機能と低レベルシステムソフトの機能を代表する

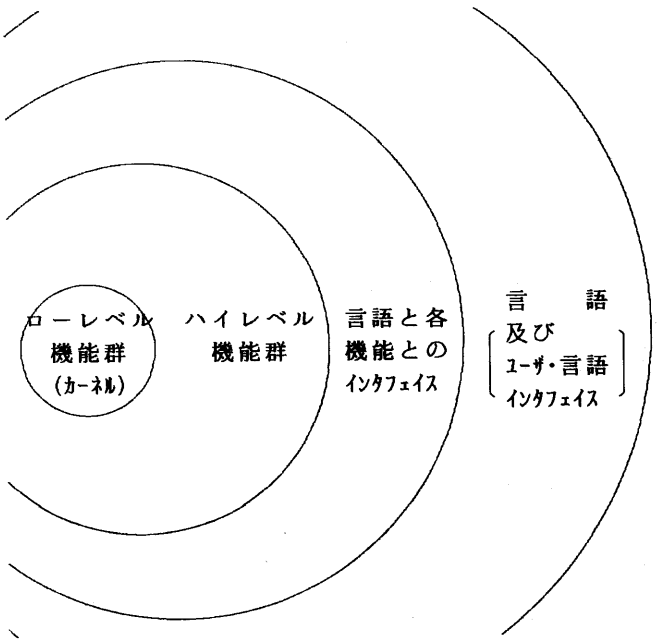
第4世代言語　　アプリケーション開発、運用維持管理上必要なよりハイレベルの機能を代表する

代表的ハイレベル機能
①データ管理機能
②単純ファイル　ハンドリング機能
③データベース　ハンドリング機能
④レポート編集機能
⑤ディスプレイ入出力制御機能
⑥グラフィックスレポート編集機能
⑦マルチユーザトランザクション制御機能
⑧セキュリティ管理制御機能
⑨集合演算機能
⑩通信分散処理機能
⑪他言語アプリケーションとのインタフェイス機能

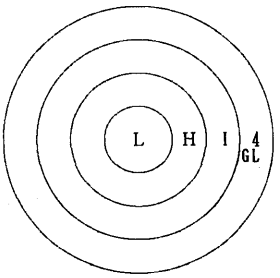
第4世代言語の効果と限界

- ソフトウェア生産性の大幅向上　３ＧＬの10倍
- ハイレベル機能といっても、まだハード＋ハイレベルコンポーネントという拡張されたコンピュータ世界の概念に立脚
- 完全な問題オリエンティドとまでは言い切れない　→　第５世代へのステップ

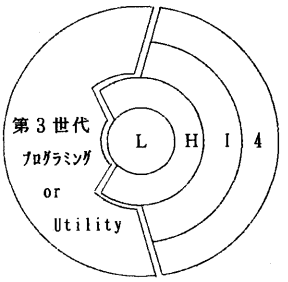
第4世代言語の一般的構造



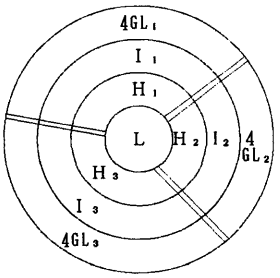
第４世代言語の完全性・統合性



完全で統合された４GLシステム



不完全な４GL



完全だが統合されていない４GLシステム

L　ローレベル機能群
H　ハイレベル機能群
I　言語と各機能とのインタフェイス
４GL　第４世代言語

・　完全で統合された４ＧＬシステムだけが真に有用であり、更に高レベルの第５世代言語の出現まで使われ続けるだろう

第４世代言語と各機能のインタフェイス

○ ゼネレータ
○ インタプリータ
○ 中間コードコンパイラ＋中間コードインタプリータ
○ コンパイラ

ゼネレータタイプの４ＧＬ

- スケルトン＋パラメータ　　３ＧＬコンパイラ機能に依存
- ３ＧＬ混在型　　ＣＯＢＯＬコード中に４ＧＬコマンドを置く
  ３ＧＬで機能不足を補う
- セパレート型　　限られた機能　レポートプログラムゼネレータ型
- ３ＧＬ世界を背景とする（既存の資産の利用）
- ４ＧＬ世界を背景とする新世代のゼネレータタイプも出現しつつある
- 変換前のチェックの水準が成否をわける

インタプリータタイプの４ＧＬ

- ２ＧＬ／３ＧＬ／４ＧＬで作成されたコンポーネント
- ステートメントを解釈し、コンポネントを呼出す
- ステートメントの集積度が低いとパフォーマンスに問題が出る
  → 中間コードコンパイラ
- 実行時に解釈されるので、変更に強い

中間コードコンパイラタイプの４ＧＬ

- インタプリティションオーバヘッドの低減
- オブジェクトコード化によるアプリケーション・プロテクション
  マシンコードより解読しやすいが
- 中間コードインタプリータのファームウェア化可能　PRIMEにおけるPICK
- データベース等の変更によるRe Compile管理　　中間コードデザインにもよる
  → 集中されたＤＤ／ＤＳが必要

コンパイラタイプの４ＧＬ

- インプリメンテーションはもっとも難しい
- ステートメントの集積度が高ければあまり必要はない
- データベース等の変更によるRe Compile管理
  → 集中されたＤＤ／ＤＳが必要

理想型は？

- 中間コードとマシンコード併用のコンパイラが理想
- マシンコードは、演算やループ制御
- ミニ・インラインコンパイラの開発が必要

データベースハンドリング機能

- 論理データベース
- 動的VIEWの設定
- 仮想ＤＢ　　実際のDBMSの選択自由度

レポート編集機能

- ディフォールトのほど良い設定と任意設定様式の併用
- 機能の豊富さと使い良さの調和
- ブレークレベル依存制御
- データ依存制御（値、件数）
- 動的／静的グルーピング
- レポートとは何かの充分な整理

主 要 機 能 の 解 説

データ管理機能

- 集中型と分散型
- （中間コード）コンパイラタイプでは集中型としないと変更時に問題をおこさないための管理が問題になる。
- 全てを集中管理すると使いにくい場合もある。オーバヘッドも問題。
- データディクショナリにどこまで情報を登録するかが問題
- ２レベル以上に分割する考え方もある。　-ex. STYLE

単純ファイルハンドリング機能

- 非４ＧＬとの容易なバッチモードインタフェイス
- バッチ処理パフォーマンスでメリット

ディスプレイ入出力制御

- 仮想端末機能　　多様な機種への対応
- 端末通信オーバヘッド削減　ＩＢＭ3270に見られるような変分検出と、より高い使い易さの両立が求められる
- 高水準端末入出力　　集中処理には限界、新しい端末制御装置が求められよう

グラフィックスレポート編集機能

- 現状ではＰＣリンクで実現しているものが大部分
- 今後の大きな課題だが方向としてはワークステーション処理か
- プロトコルが出来るか？

マルチユーザトランザクション制御機能

- フェイルセーフ的　自動的　Transaction/Commitの設定
- File Level, Record Level Lockでは不充分　→　論理フィルターロックの必要性

セキュリティ管理制御機能

- ファイルレベル　レコードレベル　フィールドレベル
- データ依存制御
- パスワードだけでは不充分、簡単でも暗号化が望ましい

集合演算機能

- ミニマムコーディングとパフォーマンスの両面に効果
- データ依存制御の効率化が必要

通信分散処理機能

- 異機種・異ＯＳ混合ネットワーク
- 異ＤＢに対するアクセス能力　　バッチだけでなくオンラインで
- 通信効率への配慮　　差分送信

他言語アプリケーションとのインタフェイス機能

- ファイルインタフェイス　　単純な相対ファイルの利用
  既存ＤＢへのバッチインタフェイス
- ＤＢＭＳや他のトランザクション制御システムとのオンラインインタフェイス
  （ライブなＤＢのシェアリング）
- 相手のシステムを変更しない一般化されたインタフェイス