4N－9

通信処理プログラムの移植に関する検討

中村 英児　佐々木 主税　松本 匡通 (NTT情報通信処理研究所)

1．はじめに

従来から既存ソフトウェアの有効利用や生産性向上のためにソフトウェアの移植の必要性が叫ばれてきた。これまでに具体的なシステム間のソフトウェアの移植の事例や移植性をあげるための方策が数多く論じられている[1]。

移植性をもたらすためにコンバータによる自動化や予め移植を考慮したコーディングによる方法がある。コンバータによる自動化については具体例を含め数多く報告されている[2]。また予め移植を考慮したコーディングについてもその方策が多く述べられているが[3]、実例ついてはほとんど報告されていない。そこで本稿では実際に移植の予定がある通信処理システムの開発を例にとり、移植を予め考慮したコーディングの移植性に対する有効性について述べる。

2．通信処理プログラムの移植

移植対象プログラムの構成を図１に示す。

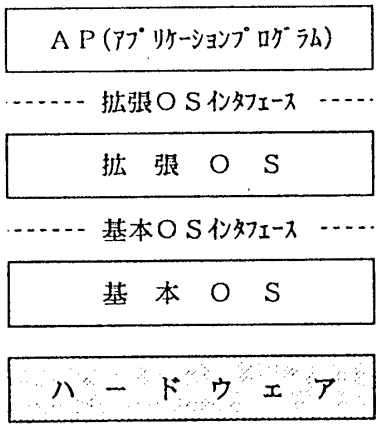


図１　移植対象プログラムの構成

ＯＳはＣＴＲＯＮ*1仕様[4]に準拠し、基本ＯＳと拡張ＯＳの２階層構成をとっている。基本ＯＳはハードウェアアーキテクチャを隠べいし、拡張ＯＳは基本ＯＳの上位に位置し、ＡＰの流通性を保証する。しかし、移植元と移植先での動作環境の相違については移植時に対処する必要がある。

本稿における移植元と移植先との動作環境の相違を表１に示す。現在開発中の通信処理プログラムは移植元の環境で動作するが、移植先環境への移植予定がある。

*1:CTRONは"Communication and Central TRON"の略称です。
TRONは"The Real Time Operating System Nucleus"の略称です。

表１　動作環境の相違

	移植元	移植先
ﾊｰﾄﾞｳｪｱ	交換用ﾌﾟﾛｾｯｻ	汎用ﾌﾟﾛｾｯｻ
ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ言語	CHILL(CCITT一部非準拠)	CHILL(CCITT準拠)
ﾊｰﾄﾞｳｪｱの主な相違	ワードマシン スタック概念無 仮想記憶無など	バイトマシン スタック概念有 仮想記憶有など
ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ言語の主な相違	オプション、組込み関数、待ち行列データ型など特殊な機能有り。	ＣＣＩＴＴ準拠のため特殊機能無し。

3．移植を妨げる要因の使用状況と対処方法

今回開発する通信処理システムと類似のシステムについてサンプル的にいくつかのプログラム（モジュール）を選択して非互換項目の出現頻度について測定した。結果を図２に示す。基本ＯＳ、拡張ＯＳ、ＡＰともに非互換項目の出現頻度は全体の規模の７～８％になっている。また、使用内訳ではハードウェアを隠ぺいする役割を担っている基本ＯＳがかなりハードウェアに依存しており、ＡＰでは基本ＯＳ

A Study on the Migration of Communication Software
Eiji Nakamura　Chikara Sasaki　Masamichi Matsumoto
NTT Communications and Information Processing LAboratories

においてハードウェアが隠ぺいされることにより、ハード依存命令の出現頻度が極端に低くなっている。

これらの非互換項目について対処方法別に測定した結果を図３に示す。移植時に変更が必要となる割合がＡＰ→拡張ＯＳ→基本ＯＳの順に多くなっていることがわかる。

また予め移植を考慮したコーディングを採用することにより、移植時に変更を要する規模は拡張ＯＳでは63.6%(コーディング規定で可能(48.1%)+移植時変換要(15.5%))→移植時変換要(15.5%)と約１／４に、ＡＰでは84.2%(84.2+0.0)→０%となる。すなわち、移植を予め考慮してコーディングすることにより、移植時に変更を要する規模が小さくなることがわかる。つまり、今回開発する通信処理プログラムにおいて、予め移植を考慮したコーディングの有効性が類推される。

４．おわりに

本稿では移植を予め考慮することにより、移植性が大きく向上することを確認し、さらにプログラムの階層構造を採用した場合、上位のＡＰでの移植性が保証されることを示した。今後は、現在開発中の通信処理システムにおいて実際にデータを取得し、本効果を確認する。さらに非互換要素の出現頻度だけでなくデバグ工数等の他の面から移植性を検討する。

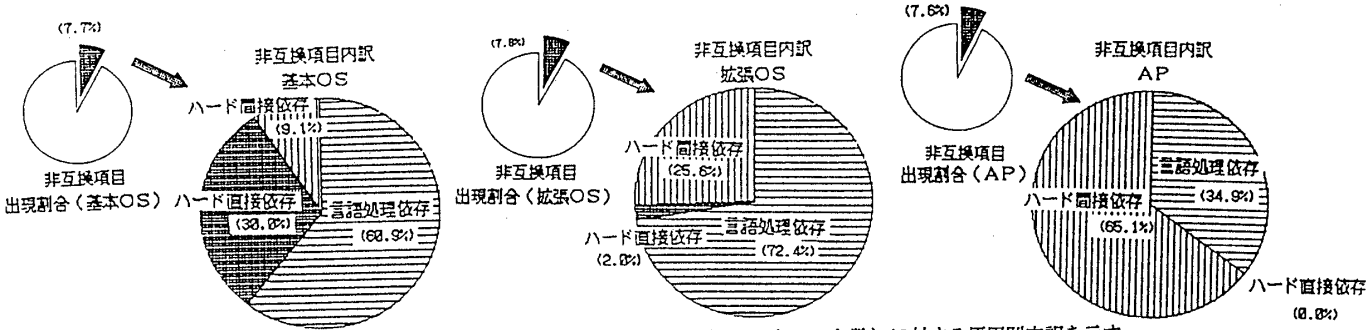
５．参考文献

〔1〕例えば A.S.Tanenbaum,Software-prac. and Exp., vol.8,pp681-698 (1978)
多田,情処論誌,vol.26.No.6,pp1033-1040 (1985)など

〔2〕例えば J.R.Wolberg,SIGPLAN Notices,vol.16, No.4,pp104-110 (1981)
金井ほか,信学研資,SE40-6,(1985)pp31-36など

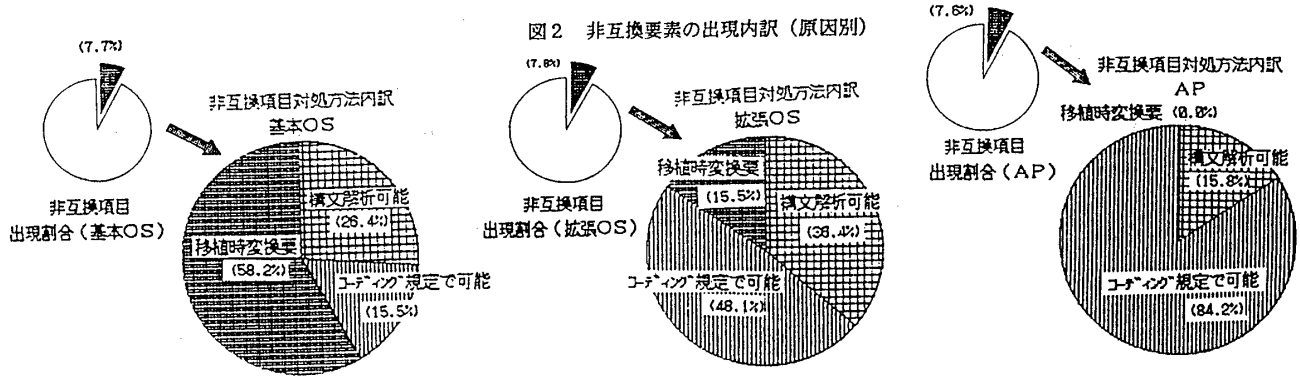
〔3〕例えば藤田ほか,情処会誌,vol.21,No.11,pp1128-1135 (1980)など

〔4〕坂村ほか,原典ＣＴＲＯＮ体系,社団法人トロン協会 (1989)



注）以下の分類で全非互換項目出現数（ステートメント数）に対する原因別内訳を示す。
（１）『言語処理依存』　：ＣＨＩＬＬ言語仕様差による非互換項目
（２）『ハード直接依存』：アセンブラやハード依存命令等の非互換項目
（３）『ハード間接依存』：ワードマシンとバイトマシンの差により間接的に非互換となる項目（ポインタモードへの強制変換など）

図２　非互換要素の出現内訳（原因別）



注）以下の分類で全非互換項目出現数（ステートメント数）に対する対処方法別内訳を示す。
（１）『構文解析可能』：非互換項目の対処方法がコンパイラの構文解析で可能
（２）『コーディング規定で可能』：構文解析はできないが、予めコーディング規定を定める事により、対処が可能である場合
（３）『移植時変換要』：構文解析もコーディング規定による対処も行えず、移植時に変更が必要

図３　非互換項目の対処方法別内訳