1G−6

# オブジェクト指向に基づいた OSインタフェース構築法

越田一郎　坂本国博
東京工科大学

## 1. はじめに

現在，POSIX をはじめとしてOS の標準化が広く行われつつある．しかし，広範囲な移植性をアプリケーションプログラム（以下AP）に与えるためには，より抽象化された，プラットフォーム非依存のアプリケーションプログラム・インタフェース（以下API）が必要とされる．

本稿では，オブジェクト指向の考えを用いたAPIの構築法を提案する．さらに，ケーススタディとして作成した「共通ファイルシステム・クラスライブラリ」について述べる．

## 2. OS インタフェースとオブジェクト指向

各所で行われているOS の標準化は，アプリケーション・プラットフォームを標準化することによってAP の移植性を高めることを目的としている．だが，パーソナルコンピュータからスーパーコンピュータまでの広範囲な計算機を単一のプラットフォームでサポートすることは難しい．今後とも，複数のプラットフォームが共存し続けると思われる．

そこで問題となるのは，異なるプラットフォーム間におけるAP の移植性である．プログラミング言語の枠内で標準とされている機能以外の部分，すなわちシステムコールを直接用いて実現されている部分は移植が困難である．また，プログラミング言語インタフェースは同じ形式であっても，プラットフォームによって意味が異なる場合もある．異なるプラットフォーム間での移植性を高めるためには，プラットフォームに依存しない，より高度に抽象化されたインタフェースが必要となる．

本稿では，このような抽象化されたインタフェースとして，クラスライブラリを用いることを提案する．ウィンドウ・プログラミング等ではAP の作成にクラスライブラリを用いることは広く行われており，標準的なクラスライブラリを制定しようとする動きもある．しかし，AP の移植性を高めるためには，ウィンドウのように限定された分野に関するクラスライブラリだけでは不十分である．AP が操作する全ての対象について標準的なクラスライブラリを提供しなければならない．

このようなクラスライブラリを作成する際に問題となるのは，プラットフォーム間の機能，性能の違いである．身近な例を挙げると，1) ファイル名の長さ，2) ファイル名の形式，3) ファイルシステムの構造，4) 数値データの表現形式，5) 使用可能なメモリの容量，など多くの問題がある．これらの問題を解決するためには，特定のプラットフォームに依存しないクラスライブラリの仕様を作成し，各々のプラットフォーム上では，その仕様を満足するクラスを実現すれば良い．そのプラットフォームが提供していない機能はエミュレートする必要がある．

もちろん，全てのプラットフォームを単一のクラスライブラリでカバーすることはできない．アプリケーションの種類に応じて，あるいはプラットフォームの種類に応じてカテゴリ分けした，何種類かのクラスライブラリを定める必要があると思われる．しかし，このような場合でも，クラスの継承機能等を使うことによって，体系的なライブラリ作成が可能になる．

## 3. 標準ファイルシステムの構築

これまで述べてきたようなクラスライブラリの実現性を検討するため，移植の際に問題となることが多いファイルシステムについて，基本的なクラスライブラリを作成した．このクラスライブラリでは，次に示す4種類の機能を実現している．

1. ファイル名の共通化
2. ファイル属性の共通化
3. ファイルインタフェースの抽象化
4. ファイルインタフェースの言語非依存化

以下，これらの機能について簡単に述べる．

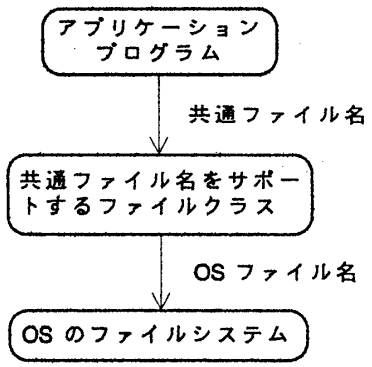


図1　ファイル名の変換

---

Object Oriented OS Interface
Ichiro KOSHIDA, Kunihiro SAKAMOTO
Tokyo Engineering University

### 3.1 ファイル名の共通化

ファイル名の命名規則はOS毎に異なるので，何らかの共通ファイル名命名規則を設けなければならない．しかし，ファイル名に制限の多いOSの場合に問題を生ずる．これを回避するため，共通ファイル名とOSファイル名の変換を行う．

### 3.2 ファイル属性の共通化

ファイル属性はOSによって様々なものが考えられる．どのような属性を共通化する必要があるかは今後の研究課題であるが，ここではアクセス権について共通化した．

アクセス権の与え方はUnixのそれを基本とした．シングルユーザのOSでは，所有者以外の属性が存在しないものもあるが，今回の実現では，OSが認識しない属性は無視することにした．それらの属性が必要ならば，エミュレートすることは可能である．ただし，セキュリティーの問題は存在する．

### 3.3 ファイルインタフェースの抽象化

本クラスライブラリにおいて，ファイルは一つのオブジェクトとして表され，2次記憶媒体上に存在するファイルの実体と1対1に対応する．また，テキストファイルにおけるコントロールコードの扱いは，OSあるいはプログラミング言語によって大きく異なるので，共通のメソッドを用意した．

### 3.4 ファイルインタフェースの言語非依存化

現在，数多くのオブジェクト指向言語が存在しており，言語によってオブジェクトの扱いは異なっている．本クラスライブラリは，オブジェクト指向における基礎的な概念のみを使って実現されているため，多くの言語に移植が可能である．

## 4. 実装

表1に示すメソッドを持つファイルクラスをUnixおよびMS-DOS上のC++，MacOS上のSmalltalk/V上に実装し，簡単なアプリケーションについて動作を確認した．

## 5. おわりに

APIのクラスライブラリ化について，実現性を検討した．今後，さらに広い範囲のAPIについてクラスライブラリ化を検討していく予定である．

表1　ファイルクラスが提供するメソッド

| メソッド名 | 機能 |
|---|---|
| new | インスタンスの生成 |
| named:FILENAME | ファイル名がFILENAMEのインスタンスを生成 |
| named:FILENAME open:MODE | インスタンス生成と同時にファイルをオープン |
| named:FILENAME open:MODE atr:ATTR | オープンと同時にファイル属性も指定 |
| | |
| setName:FILENAME | 操作対象ファイル名の変更 |
| setAtr:ATTR | ファイル属性の変更 |
| fileName | 操作対象ファイルの共通ファイル名 |
| osFileName | 操作対象ファイルのOSファイル名 |
| create | ファイルの新規作成 |
| createAs:OSFILENAME | ファイル名を特定のOSファイル名に対応させる |
| | |
| open:MODE | ファイルのオープン |
| close | ファイルのクローズ |
| getC | 1文字入力 |
| getLine | 1行入力 |
| putC:CHARACTER | 1文字出力 |
| putS:STRING | 文字列出力 |
| cr | 改行 |
| seek:INTEGER | ファイル参照位置の変更（絶対位置） |
| seekR:INTEGER | ファイル参照位置の変更（相対位置） |
| seekE:INTEGER | ファイル参照位置の変更（終端からの相対位置） |
| tell | ファイル参照位置 |
| isError | エラー状態か否か |
| isEOF | 参照位置がファイルの終端か |
| size | ファイルサイズ |