2005－OS－99（8）
2005／5／26

携帯電話の Java で稼働するファイルシステム『FML』の開発

小高健二[†]　阿部大将[††]
山口真吾[††]　並木美太郎[†††]

携帯電話の Java において，利用可能な２次記憶のサイズは数百 KB 程度にまで拡張されたが，２次記憶装置に対するアクセスメソッドは携帯電話の Java プロファイルごとに異なる上，大容量化にも対応できないものとなっている．そこで，本研究では携帯電話の Java で稼働するファイルシステム「FML」の開発を行った．FML は，携帯電話の２次記憶へのアクセスメソッドの違いを吸収し，ファイルとディレクトリ構造による管理により２次記憶の大容量化に対応する．実装の結果，J2SE のサブセットの API によるプロファイル独立のアクセスメソッド，JAR ファイルで 38KB のサイズ，そして 10KB の書き込みで 1 秒以下の性能を実現した．

A Development of File System “FML” for a Java Runtime Environment on Cellular Phone

Kenji Odaka,[†] Daisuke Abe,[††] Shingo Yamaguchi[††]
and Mitaro Namiki[†††]

Although secondary storage devices such as flush ROM become available and extend to hundreds of kilo bytes on Java runtime environments for cellular phones, the Java’s access methods for secondary storage are so different in each java profile that development is difficult to read and write data on the storage. In this research, we developed a file system “FML” that absorbs the difference of the access methods for secondary storage and achieves management with file and directory. “FML” has the access methods that is independent to the java profile and is a subset of J2SE. As the result, we achieved that the performance is one second or less in writing 10KB and the size is 38KB in the JAR file.

1. はじめに

携帯電話に Java 実行環境が搭載され，年々その性能は強化されてきた．携帯電話の Java プログラムから利用可能な２次記憶装置の容量も，Java 実行環境の進化にともない，当初の 10KB から数 100KB 程度にまで増加してきた．しかし，２次記憶装置に対するアクセスメソッドは最初期に定められたものがそのまま利用されており，J2SE の File クラスによるアクセスメソッドとは大きく異なるものとなっている．このため，携帯電話の２次記憶が大容量化した現在において，独自メソッドによる２次記憶装置の管理はプログラマにとって大きな負担となっている．

† 東京農工大学 大学院工学教育部
Graduate school of Engineering,
Tokyo University of Agriculture and Technology
†† 東京農工大学 工学部
Tokyo University of Agriculture and Technology
††† 東京農工大学 大学院共生科学技術研究部
Graduate school of Engineering,
Tokyo University of Agriculture and Technology

また，２次記憶に対するアクセスメソッドの仕様は携帯電話キャリアによって大きく異なっている．NTT ドコモの i アプリで利用されている DoJa 仕様[1)] と，Vodafone の V アプリ[2)] などで利用されている J2ME MIDP 仕様では，ストレージの構造やデータに対するアクセスモデルから大きく異なっており，データの保存を行うアプリケーションはそれぞれの携帯電話キャリアの端末専用のプログラムとして開発せざるを得ない状況になっている．さらに，miniSD カードなどの外部メモリカードへのアクセス方法も各携帯電話キャリアが独自のものを定義しており，複数のアクセスメソッドの習得はプログラマにとって煩雑なものとなっている．

そこで，本研究では携帯電話の Java 実行環境ごとのプロファイルの違いを吸収し，プロファイル独立な API をプログラマを提供すること，データをファイル単位で取り扱い，ディレクトリによる階層構造を持たせることにより拡大する２次記憶容量に対応すること，この２つを目標とし，携帯電話の Java で稼働するファイルシステム「FML」の開発を行った．

2. 携帯電話向け Java プロファイル

携帯電話に搭載された Java 実行環境に用いられる API には，大きく分けて２つのプロファイルがある．キャリアによる独自拡張が行われている場合もあるが，J2ME MIDP 仕様をベースとして拡張が行われているため，以下の２つのプロファイルに対応することにより，ほぼすべての携帯電話に対応できる．

(1) **DoJa プロファイル**

NTT ドコモが独自に策定したプロファイルで，同社が提供するサービスである i アプリで利用されている．ただし，NTT ドコモだけでなく，海外で i モードサービスを提供しているキャリアでも利用されており，日本でのみ利用されているものではない．

DoJa プロファイルにおいて，端末内の２次記憶はアプリ毎に区切られており，複数のアプリで同じ２次記憶領域を共有することはできない．アクセスメソッドは，GenericConnection フレームワークに基づいた平坦なメモリ領域であるスクラッチパッドに対し，バイト単位でアドレスを指定して読み出しと書き込みを行う方式となっており，プログラマは自ら平坦なメモリ領域を管理しなくてはならない，という問題がある．

また，505i シリーズ以降の端末で標準搭載されている miniSD やメモリースティック Duo などの外部メモリカードに対し，Java プログラムから直接アクセスすることはできないが，端末内の画像フォルダ内の画像の取り込みや書き出しは可能となっている．そこで，画像フォーマットのコメント領域などにデータを埋め込むことにより，間接的にメモリカードを経由した PC などとのデータのやりとりを行う方法が考え出されている．ただ，アクセスメソッド，方法ともかなり特殊なものとなっているため，広く利用されてはいない．

(2) **J2ME MIDP プロファイル**

携帯電話などの携帯端末向けに JCP(Java Community Process) により策定されたプロファイルで，Vodafone の V アプリや au の EZ アプリ (Java)，海外で端末メーカーが直接販売している端末などで利用されている．

端末内の２次記憶がアプリ毎に区切られているのは DoJa プロファイルと同様だが，アクセスメソッドは可変長のレコードと呼ばれるデータ領域の集合体であるレコードストアを，レコードに割り振られた番号で管理する方式が用いられている．可変長のレコードという概念はファイルに近いが，読み出し，書き込みともにレコード単位でしか行えないため，性能やメモリ使用量などの点で大きなサイズのデータには適さない，という問題がある．なお，V アプリや EZ アプリ (Java) ではこの MIDP プロファイルをベースとしてキャリア独自の拡張が行われているが，２次記憶へのアクセスメソッドはアプリ毎に区切られた MIDP の仕様のままである．

外部メモリカードに対しては，Vodafone の V6 シリーズに搭載された Java 実行環境において，Java.io.File クラスに似た独自の API によってアクセスが可能であるが，フォルダ構造などは機種依存性の高いものとなっている．また，セキュリティなどの観点からネットワーク接続機能との同時使用は行えない仕様となっている．

3. 設 計 方 針

前節で示した通り，携帯電話に搭載されている Java プロファイルには大きく分けて２つの種類があり，２次記憶に対するアクセスメソッドは大きく異なっている．また，どちらのプロファイルの２次記憶に対するアクセスメソッドも大容量化には対応できないものとなっている．そこで，これらの問題を解決するため以下のような設計方針を立てた．

(1) **プロファイル独立な API**

携帯電話には様々な２次記憶装置が存在し，アクセスメソッドもそれぞれ独自のインタフェースを持つが，FML ではプログラマが携帯電話のすべての２次記憶装置を同様に取り扱えるようにすることを目標とし，統一したインタフェースによるプロファイル独立の API を提供する．これは，携帯電話の Java プロファイルごとの２次記憶へのアクセスメソッドの違いを FML で吸収することにより実現する．

(2) **デバイス独立，拡張可能な構造**

２次記憶装置へのアクセスメソッドはプロファイルによって大きく異なるが，FML ではプロファイルごとの違いを吸収する部分と，２次記憶に対する管理・操作を行う部分を分離することで，デバイス独立な設計とする．また，新たな２次記憶装置が登場した場合にも対応できるようにするため，拡張性を持つ構造とし，SD カードなどの外部メモリカードや，端末ネイティブのストレージへの画像を使ったアクセスなど，携帯電話キャリア独自の２次記憶装置，さらにはネットワーク上のサーバも同様に取り扱うことができるものとする．

(3) **ファイルとディレクトリ構造による管理**

携帯電話の高機能化や第三世代携帯電話の普及により，携帯電話上の Java プログラムで利用可能な２次記憶の容量は増加しているが，そのアクセスメソッドは大容量化に対応できないものとなっている．そこで，FML ではデータをファイル単位で取り扱い，ディレクトリによる階層的なデータ管理を実現することで，携帯電話の２次記憶の大容量化に対応する．

(4) **java.io.File クラスのサブセットの API**

FML のアクセスメソッドは，基本的に J2SE でファイルを扱う java.io.File の入出力クラスライブラリの

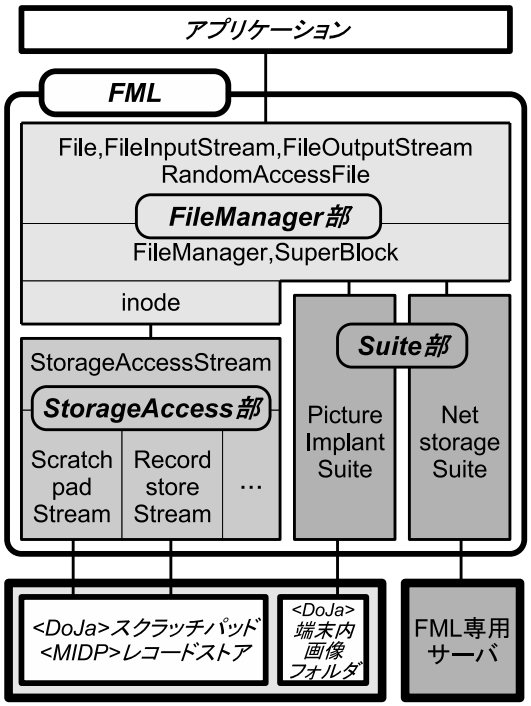

図 1 システムの全体構成

サブセットとし，すべての機能は携帯電話の Java 実行環境に搭載された Java クラスライブラリにより実現する．既存の入出力クラスライブラリのサブセットとすることで，プログラマは携帯電話独自の２次記憶装置へのアクセスメソッドを習得する必要がなくなり，簡便に携帯電話上の２次記憶装置を取り扱うことができるようになる．また，既存のソフトウェアの再利用性を向上させることができる．

4. システムの全体構成

システムの全体構成は**図 1** のようになっている．FML は大きく分けて３つの部分に分かれており，プログラマに対するインタフェースを持ち，ディレクトリやファイル名管理を行う FileManager 部，プロファイルごとの２次記憶に対するアクセスメソッドの違いを吸収する StorageAccess 部，ネットワークなど特殊な管理構造を持つ２次記憶に対する拡張構造を提供する Suite 部から構成されている．

携帯電話の Java 実行環境には，共有のクラスライブラリの追加ができないため，本ライブラリはアプリケーションの中にコンポーネントとして組み込むことによって動作する．以下に各部分の構成とクラスの概要を述べる．

(1) **FileManager 部**

FileManager 部は java.io.File のサブセットの API を持ち．プログラマに対するインタフェースを提供する部分と，ディレクトリやファイル名の管理などを行う部分で構成されている．

- **FileInputStream/FileOutputStream, File, RandomAccessFile**

FileManager 部のうち，J2SE の java.io.File のサブセットの API を持ち，プログラマに対するインタフェースとなるクラスである．File はインタフェースクラスとしてファイルとディレクトリを定義するクラス，FileInputStream/FileOutputStream は InputStream/OutputStream インタフェースによるファイル入出力クラス，RandomAccessFile はランダムアクセスを行うためのクラス，となっている．

- **FileManager, SuperBlock, inode**

FileManager 部のうち，FML 独自の API を持つクラスである．FileManager はディレクトリやファイル名を管理するクラス，SuperBlock はデータ構造記述クラスとして FML で扱う２次記憶ごとの利用情報を管理するクラス，inode はファイルのメタデータを管理するクラスである．

(2) **StorageAccess 部**

StorageAccess 部は，DoJa のスクラッチパッドや MIDP のレコードストアなど，ファイルやディレクトリによる管理構造を持たない２次記憶を，ブロック単位の読み書きに仮想化することで，プロファイルごとの２次記憶に対するアクセスメソッドの違いを吸収する部分である．StorageAccess 部でプロファイルの違いを吸収し，FileManager 部に対して StorageAccessStream インタフェースによる統一したアクセスメソッドを提供することで，FileManager 部はプロファイル非依存な実装を実現している．

- **StorageAccessStream**

FileManagaer 部に対するインタフェースクラス．実際の２次記憶へのアクセスは実装クラスが行う．現在の２つの実装クラスが設計･実装されているが，今後，新たな２次記憶装置が登場した場合にも StorageAccessStream インタフェースに従った実装とすることで，FileManager 部からは同様に扱うことができる設計となっている．

- **ScratchpadStream, RecordstoreStream**

実際の２次記憶へのアクセスを行うクラス．ScratchpadStream は DoJa プロファイルのスクラッチパッドについての実装クラス，RecordstoreStream は J2ME MIDP プロファイルのレコードストアについての実装クラスである．

(3) **Suite 部**

外部メモリーカード上のファイルシステムや，ネットワーク上のサーバなど，既にファイルやディレクトリによる管理構造を持つ２次記憶や，i-node と StorageAccess 部の組み合わせとは異なる管理構造を利用する場合において，FileManager 部のファイル名管理との間のインタフェースを持ち，その２次記憶へのア

クセスを行う部分である．

設計やテストが進んでいる実装クラスとして，DoJa プロファイルにおいて，携帯電話内部の画像フォルダへのアクセスメソッドを用い，画像にファイルを埋め込むことによって間接的にメモリカードを経由した PC などとのデータのやりとりを実現する PictureImplantSuite と，FML 専用サーバを用いたネットワークストレージに対するアクセスメソッドとして設計された NetstorageSuite がある．

5. プログラマに対するインタフェース

本節では，FML のプログラマに対するインタフェースとして，まずファイルやディレクトリの操作を行う java.io.File のサブセットの API について，次に FML 独自となる 2 次記憶装置の管理を行う FileManager クラスについて概要を述べる．

5.1 java.io.File クラスのサブセットの API

FML がプログラマに対して提供する API は Java.io.File クラスのサブセットとなっているが，Input/OutputStream，Reader/Writer などのストリームによる入出力に加え，RandomAccessFile クラスのサブセットによるランダムアクセス機能を提供することがファイルシステムとして大きな特徴となっている．ランダムアクセス機能を実現することにより，データベースなどにおいてファイル中の特定の場所へ高速にアクセスすることが可能になる．java.io.File のサブセット化においては，J2SE 上で作成したプログラムの流用が行いやすいように配慮した．利用可能なクラスとメソッドについては**表 1** に示す．

File クラスについては，ディレクトリやファイルの作成や削除，リネームなどの基本操作，パス名取得やファイルとディレクトリの判別などの基本メソッドをそのまま利用可能とした．ただ，ファイルの検索機能はサイズ削減のため FileManager クラスの search メソッドにより実現する方針とした．InputStream/OutputStream，Reader/Writer クラスについては read，write，flush など最低限必要なメソッドのみ，RandomAccessFile クラスではバイト単位の読み書きとランダムアクセスに必要な基本メソッドのみの実装することでサイズを削減したが，利用可能なメソッドの仕様は java.io.File と同じである．

5.2 FileManager クラス

FML はファイルやディレクトリの操作を行うため，J2SE の java.io.File のサブセットとすることでプログラムの再利用性を向上させている．しかし，FML は Java 上で動作するファイルシステムであるため，2 次記憶装置の初期化やマウント，フォーマットなどの管理 API が必要となるが，これらの API は java.io.File の中にはないため，FML では FileManagaer クラスを独自に定義し，その中で提供している．

表 1 java.io.File サブセットのうち利用可能なメソッド

「File」
createFile,delete,exist,getName,getParent, getParentFile,getPath,isDirectory,isFile,lastModified, list,listFiles,mkdir,renameTo,setLastModified
「FileInputStream」
close,read,skip,mark,reset
「FileOutputStream」
close,write,flush
「FileReader」
close,read
「FileWriter」
close,write
「RandomAccessFile」
close,length,read,readFully,seek,skipBytes,write

FileManagaer クラスは FML の統括クラスとして，データ構造クラス SuperBlock で構成される複数のスーパーブロック情報を管理し，これらの操作を行うメソッドを提供する．2 次記憶装置の初期化，マウントなどの操作を行うメソッドを全て FileManager クラスに集約することで，プログラマは FileManager クラス以外のクラスのメソッドを新たに覚える必要がないように配慮されている．以下に FileManager クラスのメソッドの概要を示す．

- **create/open/exist**

FML を使ったアプリケーションの起動時に実行するメソッドである．create メソッドは FML のブートストレージとなる 2 次記憶装置に FML 管理情報，ルートディレクトリを作成し，FML で利用可能な領域とするため初期化を行う．初期化された 2 次記憶へのアクセス経路などの情報は，FileManager クラスによって作成される fstab と呼ばれる管理ファイルに保存される．open メソッドは，FileManager を開き，指定した 2 次記憶装置から FML 管理情報を読み出し，FileManager を利用可能状態にする．exists メソッドは，FML のブートストレージとなる 2 次記憶装置に FML 管理情報が存在するかどうかを判別することで，FML が初期化されているかどうかを判別する．

FML を Java アプリケーション内で利用する場合，パッケージ FML をインポートし，プログラム起動時に実行されるメソッド，もしくはコンストラクタ内でこの 3 つのメソッドを用いた**図 2** のようなコードを実行し，初回起動時は初期化，2 回目以降の起動時は FileManager クラスのインスタンスの取得を行う．

- **mount/umount**

mount/unmount メソッドは指定したデバイス名が指す 2 次記憶装置上の FML 管理情報を用いて，FML の管理するディレクトリ構造の中でのマウント/アンマウントを行うメソッドである．mount メソッドでは，実行されたときに 2 次記憶装置が初期化されていない場合，内部で format メソッドが呼び出され，フォー

```
try {
  if (FileManager.exists(property) == false) {
    // 管理情報作成，初期化
    FileManager.create(property);
  }
  // FileManager のインスタンス取得
  fm = FileManager.open(property);
} catch (IOException e) {
}
```

図 2 FML の初期化コード

マットが行われる．

- **format**

format メソッドは主に create や mount メソッド内で実行されるが，既存の管理情報と保存されているデータをすべて破棄したい場合にも用いることができる．

- **getSize/getSizeAvailable**

getSize メソッドは指定した 2 次記憶装置の容量を返す．この値には，各種ファイル管理領域の大きさも含まれる．getSizeAvailable では，指定した 2 次記憶装置の利用可能な容量を返す．この容量は，ユーザプログラマがファイル保存に利用できるものとなる．

- **search**

search メソッドは FileManager 上のファイルを検索するメソッドで，指定したディレクトリの中から指定された名前を持つファイル，ディレクトリを検索し，条件に合うものの名前を要素とした String 型の配列を返す．

6. ファイル格納構造と StorageAccess 部

FML では，まず StorageAccess 部によりプロファイルごとの 2 次記憶装置へのアクセスメソッドの違いを吸収，ブロック単位の読み書きへ仮想化を行い，その上に i-node と多段間接表によりファイルおよびディレクトリの情報を管理するファイル格納構造を構築している．

本節では，FML のファイル格納構造の概要と，StorageAccess 部の設計について述べる．

6.1 FML のファイル格納構造

携帯電話の 2 次記憶装置としてはフラッシュメモリが利用されているが，フラッシュメモリ上にファイルシステムを構築するときの問題点として，書き込み速度が遅いことや書き換えの回数に制限があることが挙げられる[4]．また，[5] ではリンク構造を持つファイルシステムを提案しているが，書き込み時にガベージコレクションが必要になるという問題が発生している．そこで，FML のファイル格納構造は UNIX のファイルシステムで用いられているスーパーブロック，i-node などのデータ構造を参考としたものとすることでガベージコレクションを不要とし，データに対するアクセスの高速化を目指した．

FML のデータ構造は**図 3** のようになっている．スーパーブロックや i-node ビットマップ，データブロックビットマップについては，変更が加えられた場合に即座に 2 次記憶装置内のデータとの同期をとることにより，突然の電源断などによるデータの消失を極力回避している．以下に各領域について概要を述べる．

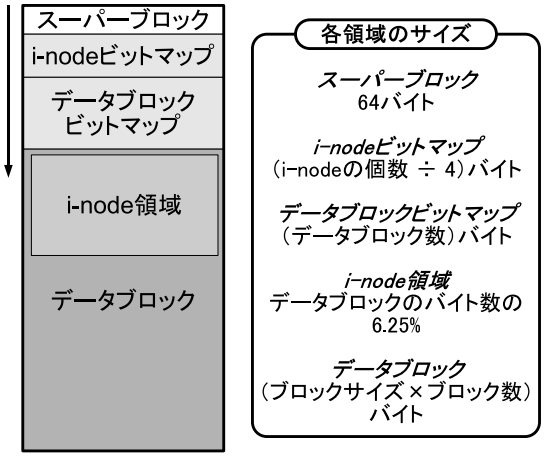

図 3 データ構造と各領域のサイズ

- **スーパーブロック**

スーパーブロックは FML 全体のメタデータを保存する領域で，サイズは 64 バイト．StorageAccess 部によってアクセスされる 2 次記憶装置の先頭に，2 次記憶装置につき 1 個ずつ作成される．スーパーブロックは SuperBlock クラスによって定義され，FileManager クラスによって管理される．

- **i-node/データブロックビットマップ**

i-node，またはデータブロックが使用中であるかどうかを判別するために用いるもので，1 つの i-node/ビットマップにつき 1 ビットを割り付けたビット配列となっている．

- **i-node**

i-node はファイルの実体を表現する構造体で，サイズは 32 バイトとなっている．FML では，i-node と多段間接表によりファイルおよびディレクトリの情報を管理する．ディレクトリ木構造については，i-node によりリンクを形成している．なお，ファイル名は directoryMetadata クラスによって定義されるディレクトリファイル内に格納され，ヒープ上のディレクトリエントリに記述されるため，i-node では管理しない．

- **データブロック**

ファイルの実体を保存する領域である．この領域は内部的には StorageAccess 部によりブロック単位で読み書きが行われる．ブロックサイズは，StorageAccess 部により決定される．なお，FML では i-node 領域もこのデータブロック内に作成される．

6.2 StorageAccess 部

DoJa プロファイルのスクラッチパッド，MIDP プ

ロファイルのレコードストアはアクセスメソッドと管理構造が大きく異なるため，FileManager 部でこれらに対するアクセスメソッドをそのまま扱うと，プロファイルごとに FML の設計を大きく変更しなければならない．これを避けるため，プロファイルの違いを吸収するのが StorageAccess 部である．

StorageAccess 部ではファイルやディレクトリによる管理構造を持たない２次記憶を，スクラッチパッドと同様の平坦なメモリ空間として仮想化することでプロファイルの違いを吸収，アクセスメソッドを統一する．また，ブロック単位でデータへのアクセスを行うことによって性能向上を図る．ブロックのサイズは，各ストレージを StorageAccessStream 向けに初期化する際に決定する．

StorageAccess 部はインタフェースクラスとなる StorageAccess クラスと，各種２次記憶装置向けの StorageAccessStream クラスのスーパークラスとして構成される．各２次記憶装置向けの StorageAccessStream は StorageAccessStream クラスを継承する別クラスとして構成され，StorageAccessStream 内のメソッド getInstance により呼び出され，StorageAccessStream 型の入出力オブジェクトとして取り扱われる．

- **「StorageAccessStream」**

FileManager 部に対するインタフェースとして，２次記憶装置のフォーマットを行う format や，open/close メソッド，ブロック単位での読み書きを行う readBlock/writeBlock メソッドによって構成されている．StorageAccessStream クラスを継承する別クラスは，これらのメソッドに合わせて２次記憶装置に対するアクセスメスッドを仮想化する．

- **DoJa 向け「ScratchpadStream」**

StorageAccess 部は平坦な２次記憶装置の空間を利用することを想定して設計されるが，DoJa 向け StorageAccessStream サブクラスである ScratchpadStream は，もともと平坦なメモリ空間であるスクラッチパッドを取り扱うため，基本的な動作は DoJa でのストレージアクセスメソッドをラッピングし，ブロック単位の読み書きに仮想化したものとなる．

ブロックサイズは，最小サイズを 32 バイトとし，利用する領域が大きくなればそれに合わせてブロックサイズも大きくする．

- **MIDP 向け「RecordstoreStream」**

MIDP のレコードストアは，複数の可変長レコードから構成されるが，MIDP 向け StorageAccessStream サブクラスである RecordStoreStream では，可変長のレコードを固定長で取り扱い，１個のレコードを StorageAccessStream における１ブロックに対応させることにより，レコード単位でしか読み書きできないレコードストアの弱点を補っている．

ブロックサイズは，DoJa 用と同じく初期化時に利用する領域によって最小サイズを 32 バイトとして，利用する領域が大きくなればそれに合わせて大きく設定する．また，データサイズ，ブロック情報，パーテーション情報などの全体情報は，レコードストアの先頭レコード (１番レコード) に保存する．

7. Suite 部

FML は，SD カードなどの外部メモリ，ネットワークストレージなど，携帯電話上の Java 実行環境が管理するストレージ以外の記憶装置も，ファイルシステムの一部として利用可能することを目標としている．

しかし，これらの特殊なストレージからデータを読み書きする場合において，NTT ドコモと Vodafone では外部メモリカードへの対応が異なっていたり，DoJa と MIDP ではネットワーク接続の仕様が違ったりと，統一した対応を行うことができない．そこで，特殊な２次記憶装置に対応するための部分として Suite 部を設けた．

7.1 SD カード向け「PictureImplantSuite」

携帯電話において，アドレス帳など端末が保存するデータや，SD カードなど外部メモリカードに対する Java プログラムからアクセスは制限されている．また，その取り扱いも内部ストレージと同様にプロファイルごとに異なる．

たとえば，Vodafone ではネットワーク接続を行わない場合にのみ，端末内データや SD カードへのアクセスが許可されている．DoJa プロファイルではアドレス帳など端末内データへのアクセスは NTT ドコモと契約した企業が，ネットワーク認証とユーザへの許可を得た場合にのみアクセス可能なものとなっており，利用する場合もデータをネットワークを通じて外部へ持ち出すことはできないようになっている．

しかし，DoJa プロファイルでは端末内の画像を保存するフォルダへのアクセスが可能になっている．この画像フォルダへのアクセスでは，画像イメージとして保存する方法のみが許可されている．また，この方式は画像を新たに追加することができるのみで，既存の画像イメージに上書き保存することは出来ない．画像イメージの整理は端末ネイティブの管理からでしか実行できず，Java から自動的に追加することは不可能である．画像イメージの読み出しも端末の画像データ管理機能を用いる必要があるが，この読み出しは自由に行うことができる．そこで，画像ファイルにファイル管理情報とファイル本体に埋め込むことにより，間接的に PC などとのやりとりを行うのが PictureImplantSuite である．

設計としては，画像の偽装や取り出し，データの入出力を行う PictureImplantStream と，ファイル管理情報とファイル本体を含んだデータ領域を一枚の画像に偽装したものとして入出力を行うために，FileManager

部とのインタフェースとなる PictureImplantSuite に分けることにより，今後同様の構造の特殊２次記憶が現れた場合にも設計を流用可能な構造としている．

7.2 ネットワークストレージ向け「NetstorageSuite」

ネットワークストレージ向けの Suite クラスである NetstorageSuite は，FML からの要求があったサーバへの HTTP 接続，Basic 認証ダイアログの発行，HTTP 接続でのファイルのやり取りなどを行う．インターネット接続中に回線がタイムアウトなどを起こした場合，専用の例外をスローし，ユーザに状況を伝える．NetstorageSuite クラスは，FML 専用サーバとの通信プロトコルを持ち，これを利用してサーバに対する読み書きを行う．また，一般のサーバからは HTTP メソッドの GET コマンドを利用してファイルを取得し，データを送ることも可能となっている．

FML 専用サーバは，FML が読み書き可能なネットワークストレージとして用意されたサーバで，NetstorageSuite と通信を行い，データをやりとりする．接続には Basic 認証を利用し，安全性をある程度確保するとともに，ユーザ固有の領域を提供する．また，DoJa プロファイルのように，通信先のサーバがダウンロード元のサーバなどに限定されている場合には，一般のサーバからファイルを取得する場合のゲートウェイとしての役割も担う．

NetstorageSuite は１つのファイルに対する逐次，ランダムアクセスによる読み書きを提供することを想定し，専用の通信プロトコルを設計している．専用通信プロトコルにおけるレスポンスボディは，携帯電話で利用可能な HTTP メソッドの関係から，いずれも text/plain を想定し，content-type に付加されるものとする．

8. 実　　装

FML で取り扱うストレージはフラッシュメモリタイプのものとなるため，書込みや消去に伴うストレージへの負荷を考慮に入れ，次の２点について配慮して実装されている．

(1) **書き込み頻度を可能な限り少なくする**

フラッシュメモリにおいてデータの書き込みを繰り返すことはメモリを物理的に劣化させる．ファイルシステムという性質上，書き込み回数は多くなりがちだが，出来る限り書き込み回数を少なくするよう努めた．

(2) **メモリ上からのデータ消去回数を減らす**

データの消去は，データの書き込み以上にフラッシュメモリに負荷を与える．ファイル消去の際にメモリ上からデータを消去すると，フラッシュメモリへのアクセス回数は書込み回数と消去回数を合計したものになり，フラッシュメモリへの負荷を高める結果となる．そこで，ストレージの利用に対しビットマップを使ったフラグ管理を行い，ファイルの消去は使用フラグを未使用にすることでフラッシュメモリへの消去アクセスをしないこととし，データブロックに対する負荷を減らした．

表 2　FML のサイズ

コンポーネント名	クラスファイル容量 (KB)	JAR 容量 (KB)
FML(DoJa)	76.7	36.8
FML(MIDP)	77.3	37.9
StorageAccess 部 (DoJa)	6.5	
StorageAccess 部 (MIDP)	7.1	
FileManager 部	70.2	

実装状況としては，FileManager 部については基本的な部分の実装が終了し，現在はデバッグを行っている．StorageAccess 部については Scractchpad-Stream，RecordstoreStream ともに実装が完了しており，DoJa プロファイルと MIDP プロファイルどちらの携帯電話でも利用可能となっている．

Suite 部については，PictureImplantSuite のうち画像の偽装や取り出し，データの入出力を行うクラスとして PictureImplantStream の実装が完了し，単体でのテストが完了している．NetstorageSuite についてはプロトコルの基本設計が終了し，突然の通信路の切断といた携帯電話の無線ネットワーク特有の問題について検討を行っている段階である．

FML のサイズを**表 2** に示す．総ステップ数は DoJa，MIDP 共に約 2500 行となっている．表 2 において，FileManager と StorageAccessStream の Jar ファイル容量は，FML の DoJa ないしは MIDP にパッケージとして統合されるため，記載されていない．サイズとしては現在販売されている携帯電話で動作可能なサイズをターゲットとしており，最新の端末ならば動作可能なものとなっている．

9. 評価と考察

FML の入出力性能について考察する．まず，ブロックサイズと入出力速度について，**図 4** にブロックサイズを 32 バイトから 1KB まで変化させた結果のブロックサイズと入出力速度の関係を示す．一般にブロックサイズが大きい方が入出力性能がよいが，２次記憶の無駄が増えるようになる．しかし，数百 KB という現在の携帯電話の２次記憶のサイズを考えると，1KB 単位でもそれほど無駄は生じないと予測される．

次に，ブロックサイズを 1KB に固定し，ファイルの大きさを変えて FML の入出力の速度を実測した値を**図 5** に示す．130KB のファイルの読み込みは 1 秒強，書き込みは最良 1 秒程度，最悪で 13 秒となっている．結果を見ると実用的な速度が得られているが，機種依存性が大きい．これは主に搭載されている Java

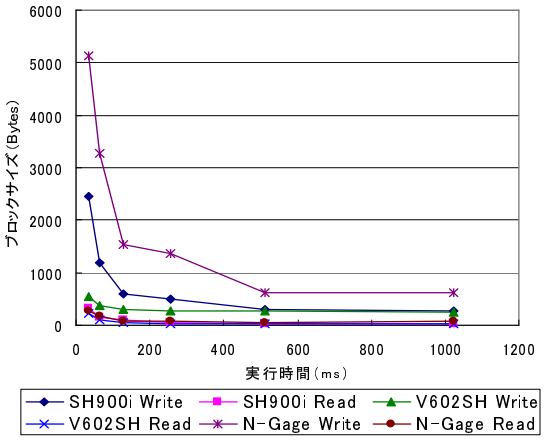

図 4　ブロックサイズを変化させた場合の性能

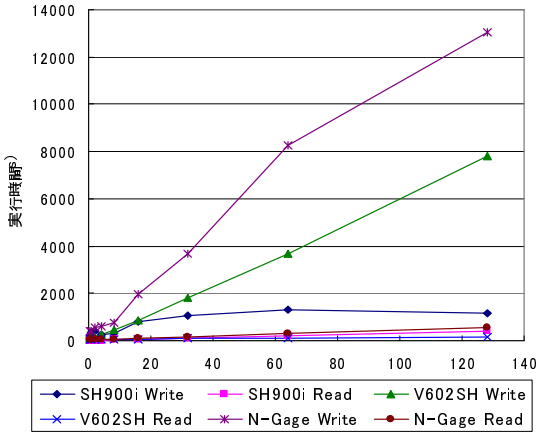

図 5　ブロックサイズ 1KB における FML の性能

実行環境の実装の違いによるものであると考えられる．V602SH と SH900i がメモリが足りなくなるまで実行環境である JavaVM のガベージコレクションを実行しないようになっているため，この 2 機種の性能には端末の I/O 性能の違いが影響していると考えられるが，N-Gage の JavaVM は適宜ガベージコレクションを実行するような実装となっているため，FML による入出力時に行われるメモリ管理が性能低下の原因となっていると考えられる．また，現状ではキャッシュ機構が搭載されていないため，性能はデータの内容に関わらずデータのサイズに応じてリニアに実行時間が増加しているが，キャッシュ機構の搭載によりさらなる性能向上を図れる，と考えている．

10. おわりに

本稿では，携帯電話の Java で稼働するファイルシステム『FML』の開発について述べた．

携帯電話の Java で利用可能な 2 次記憶は，搭載されたプロファイルによってアクセスメソッドが大きく異なっており，大容量にも対応できないものとなっているが，ブロック単位 I/O への仮想化と i-node と多段間接表をベースとした管理構造によってプロファイルの依存を吸収し，java.io.File クラスのサブセットの API によるプロファイル独立な API を提供することにより，簡便で高機能な 2 次記憶管理を実現した．また，特殊な 2 次記憶への対応として，Suite 部による拡張構造を提供することで，将来の拡張にも対応した構造とすることができた．

今後の課題としては，ファイルシステムとしての信頼性を高めるためのデバッグや，性能向上のためのキャッシュの実装，ネットワークストレージ向け「NetstorageSuite」の実現などにより，携帯電話上のファイルシステムとしての利便性を高めること，などが挙げられる．

謝　　辞

本研究開発は IPA 未踏ソフトウェア創造事業「携帯電話向けモバイル XML データベース用ミドルウェアの開発」，ノキア・ジャパン社「SymbianOS 大学教育支援プログラム」の支援により行われた．携帯電話の実機を用いたテスト・評価を行うことで，より実用的なファイルシステムとして開発することができた．関係者の方々に感謝する．

参 考 文 献

1) NTTDoCoMo: i アプリコンテンツ開発ガイド for DoJa-4.0 詳細版, 2004.
2) Vodafone: ボーダフォンライブ！向け V アプリ開発ガイド概要編, 2004.
3) J2ME Mobile Information Device Profile (MIDP),http://java.sun.com/products/midp/.
4) Mei-Ling Chiang, Paul C. H. Lee, Ruei-Chuan Chang; Flash Memory Management for Lightweight Storage Systems, Technical Report, TR-IIS-98-003, Institute of Information Science, Academia Sinica. 1998.
5) 最所 圭三，福田 晃: フラッシュメモリファイルシステムにおけるメモリ割当ての効果，情報処理学会論文誌，Vol.42, No.6, pp.1525-1534, 2001.
6) Java 2 Platform, Standard Edition (J2SE),http://java.sun.com/products/j2se/.
7) Martin Hinner，藤原輝嘉 訳: Filesystems HOWTO, http://www.linux.or.jp/JF/JFdocs/Filesystems-HOWTO.html, 2000.