社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2007－DBS－143 (32)
2007／7／3

# メタデータを利用した高度ファイル操作のためのミドルウェアの提案

三森祐一郎† 森嶋 厚行†,††

†筑波大学大学院 図書館情報メディア研究科 〒305–8550 茨城県つくば市春日 1–2
††筑波大学 知的コミュニティ基盤研究センター 〒305–8550 茨城県つくば市春日 1–2
E-mail: †{mitsu,mori}@slis.tsukuba.ac.jp

**あらまし** 近年，計算機に格納されるファイル数は飛躍的に増大しているが，現状のファイルシステムには低レベルの操作しか用意されていため，大量のファイル群の管理が困難になっている．本稿では，このようなファイル群の管理を支援するためのミドルウェアの提案を行う．特徴は，既存のファイルシステム操作系との互換性を維持すること，および，大量のメタデータを保存・利用することにより高度なファイル群操作や管理の実現を目指すことである．
**キーワード** ドキュメント空間管理，ファイルシステム，ミドルウェア

# Proposal of Middleware for Metadata-Powered Advanced File Operations

Yuichiro MITSUMORI† and Atsuyuki MORISHIMA†,††

† Grad. Sch. of Library, Information and Media Studies, Univ. of Tsukuba, 1–2 Kasuga, Tsukuba, Ibaraki, 305–8550 Japan
†† Research Center for Knowledge Communities, Univ. of Tsukuba, 1–2 Kasuga, Tsukuba, Ibaraki, 305–8550 Japan
E-mail: †{mitsu,mori}@slis.tsukuba.ac.jp

**Abstract** Recently, the number of files stored in computers is rapidly increasing. But the current file systems provide only low-level operations. Therefore, managing files is becoming a very difficult task. This paper proposes a middleware to support the management of a large collection of files. Its features include the compatibility with the current file systems, and the preservation and utilization of a large amount of metadata for realizing advanced functions for operating and managing files.
**Key words** Document Space Management, File Systems, Middleware

## 1. はじめに

近年，計算機による情報処理が日常のものになり，計算機が格納するファイル量が飛躍的に増大している．また，コンピュータネットワークや各種デバイスの発達により，これらのファイルは複数の機械に分散して格納されていることが一般的である [1]．例えば，ある大学の研究室のファイルサーバに格納されているファイル数は約 20 万であり，また，それらのファイルのコピーや関連ファイルはファイルサーバ内に留まらず，研究室の構成員のノート PC をはじめとして様々な機器に分散して格納されている．その結果，例えば，あるプロジェクトに関連するファイルはどこに散らばっているのか，あるファイルの最新バージョンはどれか，バックアップの管理はどうすればよいのか，など，計算機に格納されているファイル群の管理はますます困難になっていると言える．驚くべき事に，このような状況にもかかわらず，ファイルシステムの基本機能は，階層ファイルシステムの登場以来ほとんど変わっていない．近年，Google デスクトップサーチ [2] などのデスクトップサーチシステムの登場により，ファイルの検索は高速に行なえるようになったが，ファイル群の管理的側面に関してはあまり注目されてこなかった．

本稿では，このように分散管理された大量のファイル群の管理を支援するためのミドルウェアの開発について述べる．本ミドルウェアは，今後，ファイル群管理のために必要とされるコンピューティングリソースとアプリケーション開発コストの増大に対応するために，全てのアプリケーションから利用可能な汎用性の高いソフトウェア基盤を開発しようとするものである．これは，過去に計算機処理においてユーザインターフェースに必要とされるコンピューティングリソースと開発コストの増大に対応するために，各種 GUI ライブラリが開発されたことに類似する動機である．本ミドルウェアの特徴は，(1) 分散管理されている多量のファイル群の管理機能を提供する共通ソフトウェア基盤を目標としていること，(2) 既存のファイルシステム API と互換性を持ち，既存のアプリケーションでも機能の一部を利用可能なこと，である．

本稿では，このミドルウェアの概要，API 設計，ミドルウェアの実装について説明する．本論文の構成は次の通りである．2 章では，現在我々が取り組んでいるコミュニティ情報空間ガ

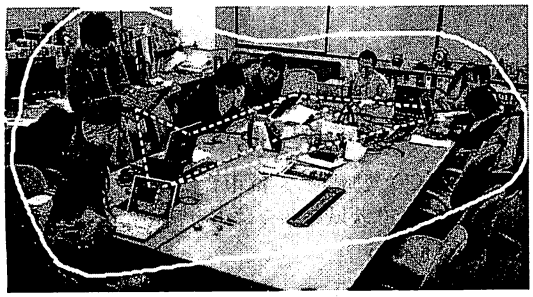

図1 インフォメーションスペースの例

バナンスプロジェクトについて説明する．本ミドルウェアはこのプロジェクトの中核を成す技術の一つである．3章で，提案ミドルウェアの概要を説明する．4章で，設計したミドルウェアの利用例を紹介する．5章で，ミドルウェアのAPIの設計について説明する．6章で，ミドルウェアの実装の方法を示す．7章は，まとめと今後の課題である．

## 2. コミュニティ情報空間ガバナンスプロジェクトの概要およびミドルウェア開発の動機

現在，我々が進めているコミュニティ情報空間ガバナンス(Governance of Community Information Spaces)プロジェクトでは，各クライアントPCやファイルサーバに格納されている多量のファイル群の管理を行うフレームワークである，InfoSpace Governorを開発している．

本フレームワークの第一の特徴は，個々のファイルシステムではなく，**コミュニティ情報空間** (Community Information Spaces, CIS) と呼ぶ複数のファイルシステム群を含む空間を管理の範囲としていることである．図1はある大学の研究室におけるCISの例である．一般に，CISには複数の計算機が含まれており，それぞれが情報を分散して管理している．例えば，この研究室で行われているプロジェクトに関連する情報は，ファイルサーバと参加メンバのPCに分散されて管理されている．それらに管理されている情報は，お互い関連しているものの，それらはシステムレベルでは明示的には関連づけられていない．例えば，そのプロジェクトに関連するあるファイルの最新バージョンはどれか，といった情報や，そのファイルが参照するファイルはどれか，といった情報をシステムはもっていない．したがって，これらの間にシステムが答えることは出来ない．ネットワークを通じてこれらの情報源が物理的には接続されているにもかかわらず，先に述べたような関連のレベルでは実は接続されていないのである

本フレームワークの第二の特徴は，これらの関連を明示的に表すメタデータを保持することにより，情報空間の統治を行う事である．具体的には，今説明したような情報資源間の関連を明示的に保持するためのInfoSpace Serverと呼ぶメタデータDBを利用し，コミュニティ情報空間の管理を行う (図2)．このメタデータDBでは，RDF[3]によるラベル付きエッジを持つグラフ構造によってファイル間の関連を保持している (図3)．ここでは，ファイル，ディレクトリ，プロセス，ユーザをRDFグラフのノードとして表現する．(以下ではDSLノードと呼ぶ)

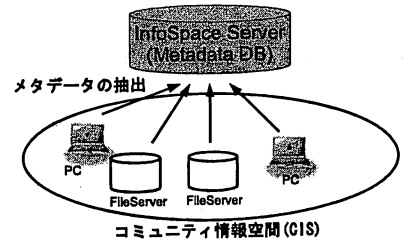


図2 InfoSpace Governorのアーキテクチャ

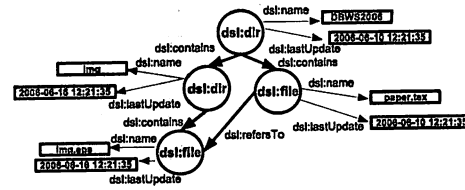


図3 情報空間コミュニティメタデータの例

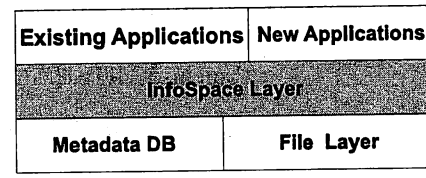


図4 提案ミドルウェアの位置付け

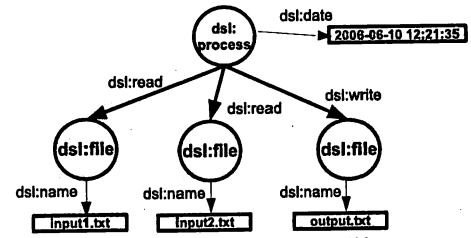


図5 プロセスとファイルの関係

このようなメタデータは，アプリケーションが直接メタデータDBと通信することによって登録・利用することも可能であるが，あくまでもデータを保持しているだけであり，このメタデータを利用した機能は個別に実現しなくてはならない．本稿で提案するミドルウェアは，あらかじめこれらのメタデータを利用して高度なファイル管理機能を実現するための機能を実装したライブラリを用意し，アプリケーションから容易に利用できることを目指すものである．

## 3. 提案ミドルウェアの概要

本ミドルウェアは，前章で説明したメタデータDBとファイルシステムの上に，新たなレイヤを構築する (図4)．本ミドルウェア・アーキテクチャの大きなポイントの一つは，ファイル操作関係の既存APIと互換性のある形で実現を行うことである．これにより，既存のアプリケーションからでもミドルウェアの機能の一部を利用可能にし，利用者が高度なファイル管理機能の恩恵を受けれるようにする．

ミドルウェアの機能は次の3つである．(1) メタデータの入手：アプリケーションや利用者によるファイル操作を監視し，ファイル操作の履歴などのメタデータをメタデータDBに格納する．具体的にはファイル，プロセス，ユーザに対応するDSLノードをRDFのリソースとして表現し，それらの関連やプロパティなどを保存する．例えば，あるプロセスにおいてどのファイルが読み込まれ，書き出されたかという情報が保存される(図5)．これらに関しては6章でより詳細に説明する．(2) メタデータへの問合せ: メタデータDBに格納された各種メタデータへの問合せを行う．大量のメタデータへの問合せを高速に処理するため，専用の問合せ処理エンジン[4]を利用する．(3) 高度ファイル操作・管理機能: メタデータDB中のデータを利用して高度なファイル操作・管理機能を提供する．例えば，最新バージョンファイルの発見や，ファイル自動内容更新，異種形式ファイルのビュー作成などの機能を提供する．

これらの機能を利用する方法は複数ある．利用者は状況に応じて下記のようなインターフェースを通じて本ミドルウェアを利用する．

**(1) アプリケーションからの利用.** アプリケーション側から本ミドルウェアを利用するためには，既存のライブラリを拡張する形で用意されたインターフェースを利用する．例えば，Java言語における java.io パッケージ等の代わりに，それと互換性のある infospace.*パッケージを利用する．これにより，既存のアプリケーションからでも容易に利用ができる．

**(2) OS での対話型ファイル操作からの利用.** OSでのファイル操作で提案ミドルウェアの機能を利用するためのインターフェイスは，2つある．まず，コマンドラインインターフェースに関しては，それをフックするための仕組みを組み込むか，専用のコマンドラインインターフェースを利用する．また，GUI型のインターフェースについては，操作をフックしミドルウェアと通信するためのソフトウェアを常駐させ，それを通じてミドルウェアの機能を利用する．

## 4. 高度ファイル操作・管理の機能例

提案ミドルウェアの利用に関しての具体的なイメージを示すため，幾つかの具体的な機能例を紹介する．現段階では必ずしも全ての機能は実装できていないが，本提案ミドルウェアを利用することによって実現可能になると考えられる機能である．

**(1) 関連ファイル，コピーファイル，最新バージョンなどの発見.** メタデータDBに格納されている関連を利用して，あるファイルに関係するファイルを発見する機能である．より具体的な例としては，既に削除してしまったファイルが必要になったとき，インフォメーションスペースに含まれる他の人のノートPCにそのファイルのコピーがあることを発見すること，などがある．また，ファイルをオープンするときに，実はそのファイルにはさらに新しいバージョンが存在する場合，それを報告する機能が実現できる．これらの機能は，OSにおけるインタラクティブな操作や新しく開発するアプリケーションだけではなく，既存のアプリケーションからも一部利用できる．この利用は，ファイルオープン用のウィンドウ(図6)を差し替えた拡張ファイル

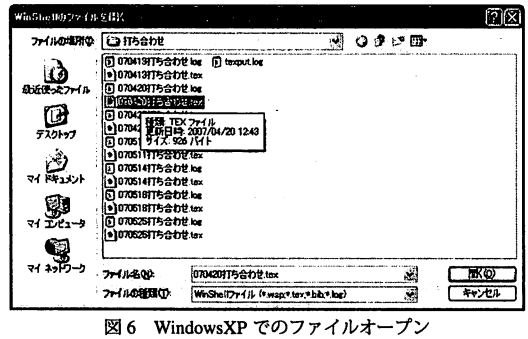

図6 WindowsXP でのファイルオープン

オープンウィンドウを通じて行う．

**(2) セマンティックバックアップ** ファイルのバックアップはディレクトリ単位で行われることが通常であり，アプリケーションが利用しているファイル群の単位とはミスマッチしている．本ミドルウェアを利用することにより，アプリケーションの利用において関連するファイルの記録が残るため，これらを単位としたバックアップを行うことが出来る．もう一つの利用は，アプリケーションを超えたバックアップの再利用である．通常，アプリケーションが自動バックアップ機能などによってファイルのバックアップを行った場合，その情報は異なるアプリケーションでは共有されない．そのため，あるファイルの以前のバージョンのバックアップが存在したとしても，それをユーザが知らなければ他のアプリケーションからはそのようなファイルを利用することはできない．本ミドルウェアを利用することにより，他のアプリケーションからもほかのアプリケーションが作成したバックアップを参照することができる．

**(3) 自動更新コピー.** コピーの際に，内容同期に関する制約を指定することが出来る．具体的には，マスタとスレーブの関係を指定することにより，スレーブと指定されたファイルの内容がマスタファイルの内容に同期するように設定される．これは，事務書類のフォーマットが年度ごとに変更されるような場合に有用であると考えられる．

**(4) ファイルの関連に基づく警告.** 例えば，あるファイルを，ファイルサーバから個人のPCにコピーをおこなう操作を行うと，それ以外のファイルをコピーする必要がないかどうかを調べ，必要なファイルのコピーを推薦する．また，ファイルを削除する場合，そのファイルを参照する別ファイルが存在する場合には削除の警告を行う．

**(5) 異種形式ファイルのビュー.** ある形式のファイルを別の形式に変換してから処理したい状況は多く存在する．例えば，CSV形式のファイルをXML形式に変換して処理をしなければいけない場合などがあげられる．通常，そのような場合には変換ソフトウェアなどを利用して一時的なファイルを作成する．しかし，そのような一時的なファイルを乱造することはファイル管理の観点から望ましくない．また，アプリケーション側で対応するためにはそのための機能を個別に実装する必要がある．提案ミドルウェアでは，この問題への解として，幾つかの形式に関して異種形式のファイルビューを用意する．これにより，一

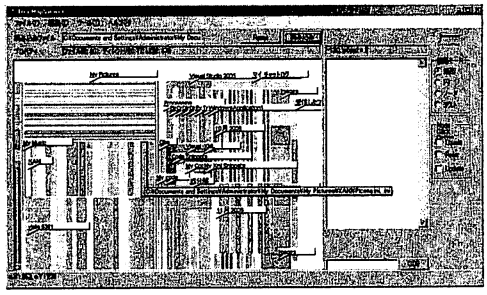

図 7　InfoSpace Maps

時的ファイルを作成することなく，アプリケーションが異種形式のファイルを容易に扱うことができる．

**(6) 各種制約の設定と制約維持の監視．** ファイルやディレクトリに関する制約を指定し，違反があったとき，もしくは違反の可能性がある時にアクションをとる．例えば，あるディレクトリに特定のファイル形式以外のファイルやサブディレクトリを追加することを禁止するという制約を設定しておくと，違反があった場合にその場で警告をしたり，あらかじめ設定しておいたメールアドレスにメールを送ったりする．

**(7) 新しいアプリケーションの開発．** InfoSpace API が提供する機能を用いて，新しいアプリケーションを開発することが出来る．この場合，本ミドルウェアが提供する機能をフルに活用することが出来る．例えば，InfoSpace Maps（図 7）[5] は，コミュニティ情報空間の視覚化ツールである．これは，コミュニティ情報空間に含まれるファイル群とそれらの関連を視覚化する．利用者は，ファイル群をそれらの関連を通じてブラウジングすることが出来る．

このような，ファイル間の関連等を利用するアプリケーションの構築が，InfoSpace API を通じて提案ミドルウェアを利用することにより可能になる．本ミドルウェアが高度ファイル操作を実現するライブラリを提供するため，アプリケーションの開発者が個々に機能を実装する必要はなく，アプリケーションプログラムのコード量の削減に寄与できる．

## 5. InfoSpace API の設計

本章では，アプリケーションからの提案ミドルウェアの機能を利用するための API である，InfoSpace API の設計について説明する．ここでは，API は特定の言語とは独立であるが，議論を具体的にするため，Java 用のパッケージを参照モデルとし，特に重要な java.io 以下のファイル入出力に関するクラスに関する議論を行う．

InfoSpace API を java.io 互換で実現するためには，(1) 既存のクラスの拡張と，(2) 新規のクラスの追加，が共に必要である．しかし，互換性を重視するためには，(2) を最小限にしたうえで高度な機能を利用可能にすることが望ましい．図 8 は，具体的に拡張するクラスと，新規で追加するクラス (InfoSpace) を示したものである．

**(1) 拡張クラス：File クラス以外．** 拡張クラスのうち，File クラス以外のクラスについては，メソッドの変更はない．ただし，

| クラス名 | 種類 | 拡張機能 |
|---|---|---|
| File | 拡張 | ファイルに関するメタデータの DB への保存と高度ファイル操作機能 |
| FileReader | 拡張 | 読み込まれたという情報をメタデータ DB に保存する |
| FileWriter | 拡張 | 書き込まれたという情報をメタデータ DB に保存する |
| FileStreamReader | 拡張 | 読み込まれたという情報をメタデータ DB に保存する |
| FileStreamWriter | 拡張 | 書き込まれたという情報をメタデータ DB に保存する |
| RandomAccessFile | 拡張 | 読み書きしたという情報をメタデータ DB に保存する |
| InfoSpace | 新規 | 高度ファイル操作機能を提供するためのクラス |

図 8　java.io 互換の infospace パッケージクラスの一覧

java.io のクラスが持つ機能に加えて，それらを実行することによって分かるファイルやプロセス間の関係をメタデータ DB に格納することが行われる．例えば，FileReader, FileWriter などのクラスにおいて，ファイルの生成や入出力が行われた場合，メタデータ DB にそのプログラムを実行しているプロセスノードを生成し，ファイルノードとの入出力の関係を追加する．また，入出力でつながったファイルの内容が同じ場合は，copy ラベルを持つエッジでファイルノードを接続する．

**(2) 拡張クラス： File クラス．** 他の拡張クラスと同様，File クラスのメソッドは，処理に応じてメタデータ DB にファイル間の関連などを追加する．ただし，他の拡張クラスと異なり，新規で追加されるメソッドが幾つか存在する (図 9)．追加されるメソッドの機能は，メタデータ DB に格納されたファイルノードの属性を調べるものと，次に述べる InfoSpace クラスの機能を使って高度ファイル操作を行なうものがある．

**(3) 追加クラス：InfoSpace クラス．** これは，高度なファイル操作を行う機能を提供する．InfoSpace クラスで実現される機能について図 10 で紹介しておく．メソッドは 3 つに分類できる．(a) メタデータ DB への問合せメソッド (getXXX メソッド) (b) メタデータ DB の更新メソッド (c) データ形式変換メソッド (view メソッド) (d) 制約メソッド（setConstraint メソッド）(e) (a)〜(d) を内部的に利用して高度機能を実現するメソッド．

InfoSpace クラスのメソッドは，拡張クラスのメソッドからも内部的に利用されている．例えば，FileReader や FileStreamReader の拡張クラスでは，オプションで指定することにより，CSV ファイルを XML ファイルとして開く，といった機能を提供するが，それは内部的には (c) のメソッドを呼び出すことによって定義されている．

### 5.1 プログラム例

提案 API を用いたプログラムの例を図 11 に示す．このプログラムは，与えられたファイルの最新バージョンを開くためのプログラムである．例えば，複数のユーザによってある文書を共同で校正している場合，他の人が別の場所にコピーして編集したものが最終版であるという状況がありうる．このような時，

| メソッド名 | 種類 | 説明 |
|---|---|---|
| boolean createNewFile() | 拡張 | この抽象パス名が表すファイルを生成．その情報をメタデータ DB に保存する |
| File createTempFile(String prefix,String suffix) | 拡張 | 一時ファイルを生成．その情報をメタデータ DB に保存する |
| boolean delete() | 拡張 | この抽象パス名が示すファイルまたはディレクトリを削除．その情報をメタデータ DB に保存する |
| boolean mkdir() | 拡張 | この抽象パス名が示すディレクトリを生成．その情報をメタデータ DB に保存 |
| boolean mkdirs() | 拡張 | この抽象パス名が示すディレクトリを生成．親子関係を保持する |
| renameTo(File f) | 拡張 | この抽象パス名が示すファイルの名前を変更．その情報をメタデータ DB に保存する |
| File[] getPerentNode(int *DSL_PROPERTY*) | 新規 | この抽象パス名が示すファイルの DSL ノードに対して関連 *DSL_PROPERTY* を持つ親ノードのファイルの配列を返す |
| File[] getChildNode(int *DSL_PROPERTY*) | 新規 | この抽象パス名が示すファイルの DSL ノードに対して関連 *DSL_PROPERTY* を持つ子ノードのファイルの配列を返す |
| boolean setChild(File f, int *DSL_PROPERTY*) | 新規 | この抽象パス名が示すファイルの DSL ノードに対して関連 *DSL_PROPERTY* を持つ子ノードとしてファイル f を関連づけます |
| boolean copyOf(File f) | 新規 | この抽象パス名が示すファイルがファイル f のコピーか調べる |
| boolean isPhantom() | 新規 | この抽象パス名が示すファイルの DSL ノードが，存在しないファイルを表す phantom タイプであるか調べる |
| String[] listAll() | 新規 | この抽象パス名が示すディレクトリにある DSL ノードを示す文字列の配列を返す |
| File[] listAllFiles() | 新規 | この抽象パス名が示すディレクトリにある DSL ノードが示す抽象パス名の配列を返す |

図 9　infospace.File クラスで変更および追加されたメソッド

| type | メソッド名 | 説明 |
|---|---|---|
| (a) | File[] getChildren(File f, int *DSL_PROPERTY*) | ファイル f の関連 *DSL_PROPERTY* である子ノードの配列を得る |
| | File[] getDescendants(File f, int *DSL_PROPERTY*) | ファイル f の関連 *DSL_PROPERTY* である子孫ノードの配列を得る |
| | File[] getPerents(File f, int *DSL_PROPERTY*) | ファイル f に関連 *DSL_PROPERTY* である親ノードの配列を得る |
| | File[] getAncestors(File f, int *DSL_PROPERTY*) | ファイル f に関連 *DSL_PROPERTY* である祖先ノードの配列を得る |
| | int getType(File f) | ファイル f のノードタイプを返する |
| (b) | boolean makeNode(File f) | ファイル f のノードを生成する |
| | boolean setRelation(File f1,File f2, int *DSL_PROPERTY*) | ファイル f1 からファイル f2 へ，関連 *DSL_PROPERTY* をつける |
| (c) | File view(File f, int *VIEW_TYPE*) | ファイル f を *VIEW_TYPE* の形式に変換した一時ファイルをビューとして返す |
| (d) | boolean setConstraint(Cond cond, Act act) | 制約 cond に違反した場合，act を実行することを要求する |
| (e) | boolean msCopy(File f1, File f2) | ファイル f1 をマスターとして，スレーブコピーを f2 に行う |
| | File[] getUpdated(File f) | ファイル f もしくはそのコピーの派生の末端である DSL ノードの配列を返す |
| | boolean backup(File f, URI uri) | uri が示す領域にコピーを作成し，バックアップの関連をつける |

図 10　InfoSpace クラスのメソッド

最新版を発見し，入手するプログラムを作成することは困難であった．本ミドルウェアは，このようなよく利用されると考えられる機能を直接提供する API を用意している．したがって，プログラマはメタデータ DB などに直接アクセスせずに，これらの機能を組み込んだアプリケーションを簡潔に実現できる．

## 6. ミドルウェアの実装

ミドルウェアは，次の 3 種類の部分に分けて実装される (図 12)．(1) 機能実現部，(2) アプリケーション連携部，(3) ユーザインタラクション部．

機能実現部は，本ミドルウェアの中心となる部分であり，5 章で説明したような機能を実際に提供する．アプリケーション連携部は主に API を実装したライブラリとして実装され，アプリケーションプログラムの動作からメタデータを入手したり，アプリケーションと機能実現部のインタフェースとして動作する．ユーザインタラクション部は主に OS のコマンドラインや GUI への追加モジュール等で実装され，OS レベルでのユーザによるファイル操作からメタデータを入手したり，機能実現部が提供する機能のユーザへの提供を行う．

### 6.1 機能実現部の実装

**メタデータ DB への格納．** 機能実現部は，アプリケーション連携部およびユーザインタラクション部からの要求に従い，メタデータ DB にデータを格納する．メタデータ DB への格納データは DSL (Document Space Language) [6] で記述する．DSL とは，RDF 上にファイル群管理に必要なクラスや関連を表現するボキャブラリを独自に定義した言語である．DSL では，リソースとして，ファイル，プロセス，ユーザなどを持つ (図 13)．

**高負荷処理の遅延実行．** 提案ミドルウェアではファイル操作のパフォーマンスを落とさずに高度機能を提供する必要がある．そこで，処理の一部はファイル操作後に遅延して行うよう実装し，応答時間の短縮を図る．例えば，同じプロセスから入出力された 2 つのファイルの内容が同一である場合，メタデータ DB に格納されているこれらのファイルを表すノード間に copy という関連を追加する必要があるが，このファイルの同一性のチェックは時間がかかる．そこで，このチェックは行わずにアプリケーション連携部に制御を返し，この同一性のチェックは

```
import infospace.*
public class sample {
  static void main(String[] args){
    File[] farr;
    File file = new File(args[0]);
    try{
      InputStreamReader input =
         new InputStreamReader(System.in);
      farr = InfoSpace.getUpdated(file);
      for(int i=0; i<farr.length; i++){
        System.out.println
                  (i+":"+farr[i].filename());
      }
      System.out.println
                  ("Select a file to open:");
      String str = input.readLine();
      int n = Integer.valueOf(str);
      FileStreamReader output =
        new FileStreamReader(farr[n]);
    ...
    }catch(Exception e){
    ...
    }
  }
}
```

図 11 infospace パッケージを利用したプログラム例

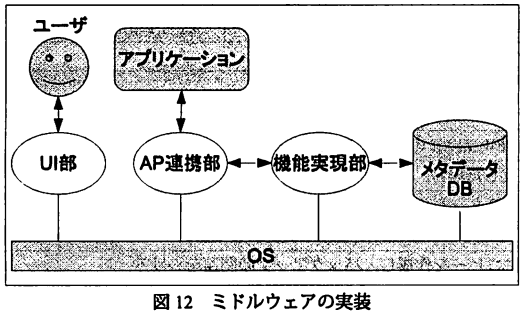


図 12 ミドルウェアの実装

その後実行するなどの工夫を行う．

| クラス | 説明 |
|---|---|
| dsl:dir | ディレクトリを表すクラス |
| dsl:file | ファイルを表すクラス |
| dsl:symbolicLink | シンボリックリンクを表すクラス．dsl:File のサブクラス |
| dsl:phantom | 存在しないが参照されるディレクトリもしくはファイルを表すクラス |
| dsl:process | プロセスを表すクラス |
| dsl:user | ユーザを表すクラス |

図 13 DSL で用意するクラス

### 6.2 アプリケーション連携部

アプリケーション連携部では，InfoSpace API を実装したモジュールとして実装される．本モジュールの主要な機能の一つは，API に基づくファイル操作にしたがって，適切なメタデータを入手し，機能実現部に送信することである．特に，アプリケーションの実行時に読み書きしたファイルの関連を作るために，アプリケーションの実行開始時にジョブノードを作成し，ファイルの読み書きに応じて read および write 関連の作成を行う必要がある．

### 6.3 ユーザインタラクション部

ユーザインタラクション部の実装では，GUI および CUI によるファイル操作の情報を入手しなければならない．その実現方法は幾つか考えられるが，現在は下記の方法で実装を進めている．

**Windows GUI.** Windows GUI を対象としたユーザインタラクション部を実現するために，C#のライブラリを用いて実装を行っている．

**コマンドラインインターフェース.** Unix の Shell や Windows のコマンドラインなどの CUI インタフェースのシステムでは，基本コマンドの置き換えや追加によってユーザインタラクション部を実現する．例えば，UNIX の cp や mv といった基本的なコマンドはメタデータ DB と連携するコマンドの実装と置き換える．また，高度ファイル操作・管理の機能を実現する新しいコマンドを追加する．

## 7. ま と め

本論文では，複数の計算機から構成されるコミュニティ情報空間に含まれるファイル群の高度操作・管理を実現するためのミドルウェアの提案を行った．本ミドルウェアでは，既存のファイルシステムと比較して大量のメタデータを保持し，それらを有効に活用する．今後の課題としては，提案ミドルウェアの実装の完成度の向上，パフォーマンスの検証，実利用におけるユーザビリティの検証，等があげられる．

## 謝　辞

ゼミ等においてコメントいただきました，筑波大学大学院図書館情報メディア研究科 杉本重雄教授，阪口哲男准教授，永森光晴講師に感謝いたします．

## 文　献

[1] Daniel J.Gonçalves, Joaquim A. Jorge: An Emprical Study of Personal Document Spaces, Springer LNCS 2844, Interactive Systems: Design, Specification and Verification, DSV-IS 2003 Proceedings, pp 46-60 Funchal, Portugal, 4-6 June 2003.

[2] Google. Google Destop Search. http://desktop. google.com/ .

[3] W3C. Resource Description Framework (RDF). http://www.w3.org/RDF/.

[4] 石川憲一，森嶋厚行，田島敬史：メタデータ DB のための到達ノード問合せと更新処理の効率化，電子情報通信学会第 18 回データ工学ワークショップ (DEWS2007), 2007 年 3 月.

[5] 望月祥司, 児玉麻莉子, 石川憲一，森嶋厚行, 田島敬史：大規模ディレクトリ空間視覚化のための論理構造抽出, 電子情報通信学会第 18 回データ工学ワークショップ (DEWS2007), 2007 年 3 月.

[6] 石川憲一，森嶋厚行, 田島敬史：大規模ドキュメント空間管理のための意味ファイルシステムの構築, 電子情報通信学会技術研究報告, Vol.106, No.150, DE2006-115, pp. 139-144, 2006 年 7 月.