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

2006－DBS－140 (II) (95)
2006／7／14

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

石川憲一[†] 森嶋厚行[†] 田島敬史[††]

個人のデスクトップやチームの共有ファイルサーバなど，ファイルシステムに格納されるデータ量は年々増加している．しかし，現状のファイルシステムにはファイルコピーや移動といった低レベルの操作しか用意されておらず，大規模なファイル群を管理するための機能はほとんど提供されていない．本論文では，大規模なファイル群の管理機能の開発を目指して，その基盤となりうるシステムの構築について述べる．

Development of a Semantic File System for Managing Large Document Spaces

KEN'ICHI ISHIKAWA,[†] ATSUYUKI MORISHIMA[†] and KEISHI TAJIMA [††]

The amount of data stored in file systems, namely, those stored in personal desktops and file servers, is rapidly increasing. The current file systems, however, only provide low-level operators, such as file copy and move, and have few functions for efficiently managing large collections of files. This paper describes the construction of a system, which is intended to be used as a basis for developing management functions for large file collections.

1. はじめに

ファイルシステムに格納されるデータ量は年々増加している．文献[1]によると，近年，世界のハードディスク容量は劇的に増加している．同時に，格納されているデータ量も増大している．例えば，小さな組織のファイルサーバであっても数十万のファイルを格納していることは特別なことではない．したがって，ファイルシステムが扱っている情報を効率よく管理，検索する手法は重要な問題であるといえる．

この問題に対するアプローチの一つとして，近年，デスクトップ検索システムが注目を集めている．代表的なシステムとしては Google Desktop Search[2] や Spotlight[3] などがある．以前からファイルシステム中のファイルに対する検索機構は存在したが，これらはインデックスを利用して高速にキーワード検索を行うのが特徴である．

一方，ファイルシステムに格納されている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ず，現実のファイルシステムでもそのための機能は用意されていない．その結果，検索にかかるコストよりも，続いて述べるような，多量のファイル群の整理や，ファイル間の関連一貫性の維持にかかるコストの方が，利用者にとって負担になっている．

ここでいうファイル群の「整理」とは，ディレクトリ構造を利用してファイル群を組織化するという意味だけでなく，不必要なファイルを削除したり，アーカイブしておくべきファイル群を選択することといった，より広い意味を含んでいる．また，ここでいうファイル間の関連とは，あるファイルのコンテンツから，他のファイルへの参照があるような場合の関係や，あるファイルのコンテンツが他のマスターファイルのコピーである，といった関係のことである．より具体的な例としては，次のようなものがあげられる．(1) あるプロジェクトに関連するファイル群が，そのプロジェクトのディレクトリだけではなく，プロジェクトの構成メンバのホームディレクトリなどに分散してしまっているとする．このとき，関連するファイルだけを集めたディレクトリを作成する．(2) 同じファイルのコピーが多量に存在するとき，不必要なファイルを削除する．(3) 古いバージョンのファイルを削除し，最新のファイルだけを残す．(4) ファイルサーバから，あるディレクトリのコピーを自分の PC にコピーしたいとき，そのディレクトリに含まれているファイルから参照されている画像ファイル等も同時にコピーする．(5)

† 筑波大学大学院 図書館情報メディア研究科
Graduate School of Library, Information and Media Studies.
†† 京都大学大学院 情報学研究科
Grad. Sch. of Informatics, Kyoto University

あるディレクトリを削除したいとき，それを削除しても他のファイルの利用に問題がないことを確認する．(6) 昔に設計したディレクトリ構造とは異なる，現在の状況に合わせた新しいディレクトリ構造ビューを構築する．(7) 現在のファイルサーバから削除して，三次記憶にアーカイブしておくファイル群を選択する．(8) ファイルが他のファイルのコピーであるとき，元のファイルの更新結果を反映させる．

現在一般に利用されているファイルシステムでは，これらを支援するための機能が用意されていない．したがって，手作業で行わざるを得ない．ファイル数が少ない場合には可能であるが，ファイル数が多量になると対処できなくなり，あきらめたり，放置してしまうことが一般的である．したがって，ソフトウェアによる何らかの支援が重要であると考えられる．

本稿では，ファイルシステム中に多量のファイル群が格納されているときに，これらの整理や参照関係の一貫性の管理を支援する機能の実現について議論し，それらの機能を実装したシステムの開発を行う．本研究のアプローチのポイントは次の通りである．

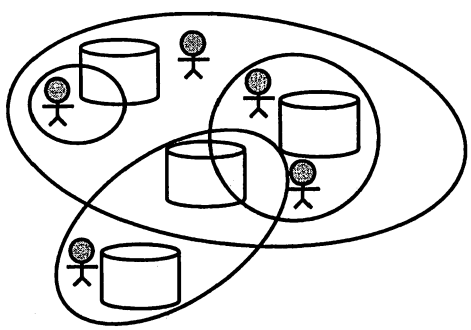

図 1　複数の利用者，情報源とドキュメント空間

(1) 複数の利用者，複数の情報源から構成されるドキュメント空間の統合管理． 本研究におけるドキュメント空間とは，特定のディスクに格納された単一のディレクトリ構造だけを指すのではなく，一般には複数のファイルサーバやデスクトップなどにまたがるより広い領域に含まれるファイル群からなる空間を指す．また，利用者が複数であることを想定している（図 1）．一般には，利用者はこのようなドキュメント空間上で仕事をおこなっているため，統合して管理できるような仕組みの開発に取り組むことが重要と考えている．

(2) ファイルに関するリッチなメタデータを管理するためのレイヤの構築． ファイルの整理や一貫性の管理をおこなうためには，現在広く利用されているファイルシステムが明示的に管理しているメタデータだけでは不十分である．例えば，ファイル間の参照関係などはファイルのディレクトリ構造には反映されていないため，それらのチェックを行うことは簡単ではない．そこで，本システムでは，ファイルのコンテンツから抽出したり，ファイルの操作をフックすることによって入手したメタデータを明示的に管理するレイヤーを用意し，その上で各種管理機能をはじめ，各種機能を実現できるような仕組みを構築する (図 2)．

(3) 大規模ドキュメント空間におけるファイル群の整理，一貫性管理機能を実現するためのオープンなアーキテクチャの開発． 大量のファイル群の整理，一貫性管理などの機能は確立されていないため，開発の基盤となるプラットフォームを構築することは重要である．そこで我々は，システム上で，容易に大量ファイル群の整理，一貫性管理機能を開発，追加できるような，オープンなプラットフォームの開発を目指す．実現にあたっては，可能な限り既存技術や各種標準との互換性を重視し，広く利用可能なシステムを構築する．具体的には，外部とのインターフェースとして NFS[4] に準拠する．また，メタデータレイヤーの構築には RDF[5] 関連技術をベースに，我々の目的に必要と思われる機能を追加することによって行う．これらにより，他のソフトウェアや環境などと容易に結合可能なシステムの構築を目指す．

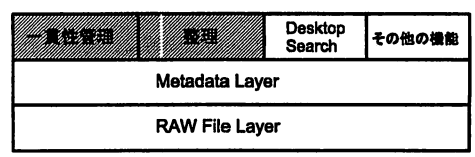

図 2　アーキテクチャの概念図

本稿では，ファイル群の整理，一貫性管理機能の実現を支援するためのメタデータレイヤの構築について主に説明する．本稿の構成は次の通りである．2 章で関連研究について説明する．3 章ではシステムの概要について説明する．4 章では，メタデータレイヤ実現のための用件と我々の設計について説明する．5 章では，あるファイルサーバ上に構築されたメタデータレイヤを用いて実験を行い，本研究を進めていくための問題のいくつかについて議論を行う．

2. 関連研究

意味ファイルシステム (Semantic File Systems) の研究自体は以前からあり，新しいものではない．Gif-

ford らの研究[6] では，本研究で開発するシステムと同様に，NFS に準拠した意味ファイルシステムを構築している．そこでは，各ファイルは実体としてのディレクトリに所属せず，その代わりに複数の属性をもつ．ディレクトリは，属性の値に関する問合せで表現される仮想ディレクトリに所属する．Presto[7] では，同様に属性を用いてファイルを管理するファイルシステムを構築している．この研究の焦点は，このようなファイルの管理を行う際に，どのようなユーザインターフェースが必要かというものである．OMX-FS[8] では，OM data model と呼ばれるオブジェクトモデルを用いて，ファイルを永続オブジェクトとして管理する．本研究との共通点は，ドキュメント空間を単一の情報源に限定していないことと，ファイル間の関連についても扱うことである．近年，マイクロソフトは WinFS[10] の開発を発表している．WinFS は，ファイルシステム中のファイルだけでなく，RDB エンジンも統合した総合情報管理システムである．ファイルのメタデータが RDB に格納されており，それらを用いて既存のファイルシステムにはない機能の実現が図られている．

以上の研究やシステムと本研究が異なる点は次の二点である．(1) これらの研究では，本研究の目的である大規模ファイル群の整理や関連の一貫性の管理などの機能の実現が目的ではないこと．上記研究で実現されているものは，ファイルの属性を用いた問合せによる仮想ディレクトリの構築など，限定的な機能にとどまる．本論文では，特に整理，一貫性維持などの機能を開発する基盤を実現するフレームワークについて説明する．(2) 本研究では，メタデータレイヤの構築に RDF を用いている．これにより，RDF ベースに開発されている各種セマンティック Web 関連技術の応用が容易であると考えられる．

Semex[11] はファイルシステムでないが，ファイルシステムも含んだ各種情報源の上に意味レイヤを作成し，より高度な検索機能を実現するシステムである．特に，ファイルの整理や一貫性管理を目的としたものではない．

各種デスクトップサーチシステム[2][3] は，検索のみを扱っており，本研究とは相補的な関係といえる．

3. システム概要

本章では，我々が開発中のシステムの概要について説明する．図 3 は，システムのアーキテクチャである．システムには，NFS インターフェースを通じて他のソフトウェアよりアクセスする (図 3(a))．ファイルは，RAW ファイルレイヤに格納されている (図 3(b))．RAW ファイルレイヤが実際に何であるかには特に前提を置いていない．DB システムを利用したり，既存のファイルシステムであったり，専用ネイティブストレージを利用することも考えられる．

メタデータレイヤの情報は，RDF データベースに格納する．ファイルやディレクトリは RDF グラフのノードとして表現し，ファイルの属性やファイル間の関連はプロパティとして表現する．ここには，ディレクトリの木構造に加えて，ファイル間の参照関係や同一内容のファイルであることを示す関係を明示的に格納する．RDF データベースに格納するための情報は，NFS インターフェースの処理をフックするか，RAW ファイルレイヤに格納されているファイルコンテンツから抽出する (メタデータの抽出，及びファイル間の関連の発見手法については本稿では議論しない)．

既存のファイルシステムと同様，ファイルの cp や mv といった基本的なコマンドは提供される．異なる点は，各コマンドの実行時には，必要に応じて RDF データベースに格納されている情報にアクセスされることである．例えば，あるファイル群をファイルサーバから個人の PC にコピーをおこなうために cp コマンドを実行したときには，それ以外のファイルをコピーする必要がないかどうか，RDF データベースへの問合せが行われる．もし，必要なファイルが見つかれば，利用者に通知する．

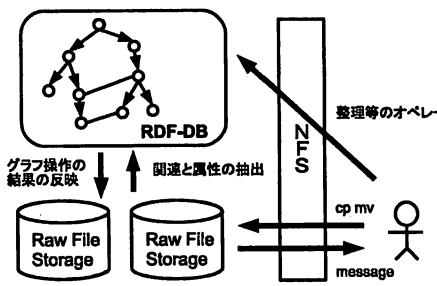

図 3 システムの実現

4. Metadata Layer の実現

本章では，図 2 のメタデータレイヤの実現について説明する．具体的には，メタデータを表現するための言語である Document Space Language (以下 DSL) および，DSL によって記述された RDF データに対する問合せ言語である SPARQL/DS の設計を行う．

4.1 Document Space Language の設計

DSL 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定義した言語である．DSL で独自に定義したクラスを図 4 に，プロパティを図 5 に示す．DSL で新たに定義したクラスのうち説明が必要なものは dsl:phantom である．これが必要な理由は次の通りである．すなわち，本システムではファイルの削除コマンドの実行によってファイルの参照関係を表すプロパティが dangling link となる可能性があるが，RDF ではそのような状況を想定していないため，参照先が存在しないリンクを維持することが出来ない．したがって，存在しないファイルを表現するためのノードとして dsl:phantom を導入する．

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

図 4 dsl で用意するクラス

ディレクトリの親子関係は，dsl:contains を用いて表現する．それ以外の関係は dsl:relatedTo を用いて表現する．dsl:relatedTo は，サブプロパティとして，そのファイルが他のファイルを参照することを表現する dsl:refersTo と，同じファイルのコピーであることを表す dsl:equals を持つ．図 6 に，DSL による RDF グラフの例を示す．

4.2 SPARQL/DS の設計

現在，SPARQL[12] が，RDF 問合せのための有力な言語となりつつある．標準に準拠するという目的から，本システムでも SPARQL を用いて意味レイヤの操作を行いたい．しかし，SPARQL では RDF データベースの更新が規定されていないため，このままでは我々の要求を満足しない．なぜなら，本システムでは，ファイルシステム中に格納されているファイルを操作した結果を反映させるために RDF データベースの更新を行う必要があるからである．したがって，SPARQL をベースとした拡張言語 SPARQL/DS の設計を行う．

SPARQL/DS の構文を説明する前に，SPARQL/DS に組み込まれる各更新機能をそれぞれ説明する．

プロパティ	説明
dsl:contains	ディレクトリ辺
dsl:name	ディレクトリ名もしくはファイル名
dsl:extension	拡張子
dsl:ftype	ファイル形式
dsl:owner	所有者
dsl:lastupdate	ファイルの最終更新日時
dsl:relatedTo	関連がある
dsl:refersTo	参照している．relatedTo のサブプロパティ
dsl:equals	同一ファイルのコピーである．relatedTo のサブプロパティ

図 5 dsl で用意するプロパティ

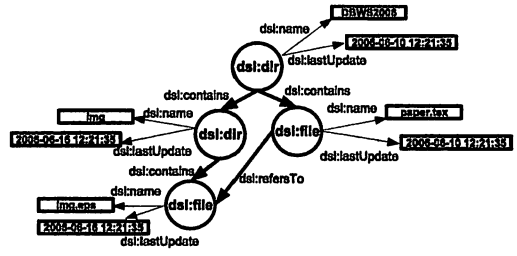

図 6 DSL による RDF グラフの例

[**create_node**(*id*)] 新しいノードを追加する．

[**del_node**(*id*)] ノード *id* が phantom 以外の他ノードから参照されている場合は，*id* の type を phantom に変更する．そうでなければ，*id* を削除する．

[**create_property**(*source-id*, *p-name*, *dest-id*)] *sourcce-id* から *dest-id* への，*p-name* で指定されたプロパティを新しく生成する．

[**del_property**(*source-id*, *p-name*, *dest-id*)] *sourcce-id* から *dest-id* への，*p-name* で指定されたプロパティを削除する．

[**change_property_val**(*source-id*, *p-name*, *dest-id*1, *dest-id*2)]
property の値を変更する．**del_property**(*source-id*, *p-name*, *dest-id*1); **create_property**(*source-id*, *p-name*, *dest-id*2) と等価である．

[**del_graph**(N)] ノード (id) 集合 N に含まれるノードと関連するプロパティを削除する．この操作は，**del_property** を用いて図 7 のように実現できる．

[**copy_graph**($o \subseteq \{\mathtt{I}, \mathtt{O}\}$, N)] ノード (id) 集合 N

```
1  delete_graph(set of nodes N) {
2    for each node n in N {
3      for each property (p, dest) in properties(n) {
4        del_property(n, p, dest);
5      }
6      del_node(n);
7    }
8  }
```

図 7 del_graph の実現

```
1  copy_graph(option:{I,O}, set of Nodes N) {
2    for each node n in N {
3      create_node(n');
4    }
5    for each node n in N {
6      for each (p, dest) in properties(n) {
7        if (dest in N) {
8          add_property(n',dest');
9        } else if ('O' in option ) {
10         add_property(n', dest);
11       }
12   }
13   if ('I' in option) {
14     for each (p, source) in comingProperties(n) {
15       change_property(source, p, n')
16     }
17   }
18 }
```

図 8 copy_graph の実現

に含まれるノードとプロパティのコピーを追加する．N 内外を横断するプロパティの扱いを指定するオプションとして，$\{\mathtt{I},\mathtt{O}\}$ の部分集合 o を受け取る．I が指定されている場合には，N 外のノードから N 中のノードを指すプロパティエッジの先をコピー先の対応するノードに変更する．O が指定されている場合には，N 中のノードから N 外のノードを指すプロパティエッジの元をコピー先の対応するノードに変更する．**copy_graph** は，他の関数を用いて図 8 の様に実現できる．

SPQRQL/DS は以上の RDF データベース更新機能を SPARQL に追加したものである．SPQRQL/DS の構文を図 9 に示す．“select” で始まる通常の SPARQL の問合せに加えて，更新の指定が可能なようになっている．copy graph と del graph については，ノード集合は SPARQL の where 節により指定されるように設計されている．

5. 予備実験

本章では，これまで設計してきた枠組みを基盤として大規模ドキュメント空間管理の研究を進めるためのいくつかの予備実験を行う．具体的には，(1) あるファイルサーバに格納されているファイル群の性質に関する予備実験，(2) 本フレームワークでの処理における RDF データベースエンジンの性能に関する予備実験，について報告する．

```
<expression> := <select> | <update> { ';' <update> }
<update> :=  <create_n> | <del_n>
           | <create_p> | <del_p>
           | <change_pv> | <cp_g> | <del_g>
<select> := ''select'' { <var> } <where_expr>
<create_n> := ''create node'' <id>
<del_n> := ''del node'' <id>
<create_p> := ''create property'' <id> <label> <id>
<del_p> := ''del property'' <id> <label> <id>
<change_pv>:= ''change property value'' <id> <label> <id>
<cp_g> := ''copy graph'' <option> <where_expr>
<del_g> := ''del graph'' <where_expr>
```

図 9 SPARQL/DS の構文

項目	数
リソース総数	171,686
dsl:dir	23,997
dsl:file	146,154
dsl:phantom	1,523
プロパティ総数	1,114,272
dsl:contains	170,116
dsl:refersTo	7,477
dsl:equals	30,561

図 10 メタデータレイヤ統計情報

5.1 あるファイルサーバ中のファイル群について

筑波大学森嶋研究室のファイルサーバ (コンテンツ総容量 126GB) について，メタデータレイヤのデータ構築を行った．その概要を図 10 に示す．図 10 で特に興味深いのは，phantom の数が 1,523 もあることである．ファイル間の関連の一貫性が保たれていないことがわかる．

コピーファイル数．ファイルサーバ中で，コピーを持つファイルの数は 23,138 であり，コピーで作られたファイルの総数は 53,698 であった．ここでは，同じファイル名を持ちかつ同一サイズのファイルをコピーとして数えた．

同一名ディレクトリの数．図 11 は，同一ディレクトリ名の個数に関する度数分布を示す．X 軸は，同じディレクトリ名がいくつあるかという数ごとのカテゴリを表している．すなわち，X=5 は，同一ディレクトリ名が 5 つであるディレクトリ名のカテゴリを表している．図では，X=1〜6 のカテゴリは省略している．興味深い例としては，X=86 のカテゴリに figure という名前のディレクトリが存在する．また，X=33 に research, X=25 などに個人名のディレクトリ，X=22

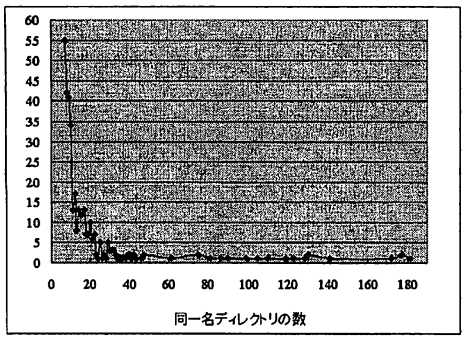

図 11 同一名ディレクトリの度数分布

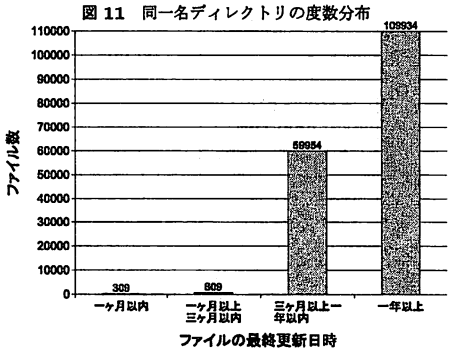

図 12 最終更新日時の度数分布

に seminar, X=17 に projects, X=15 に people 等がある.

最終更新日時. 図 12 は, ディレクトリとファイルの最終更新時刻である. このファイルサーバは 2004 年の設置からまだ 2 年弱しか経過していないが, それでも, ファイルサーバに格納されているうちの 60%以上が 1 年以上更新されていない.

5.2 本フレームワークでの処理における RDF データベースエンジンの実行性能

提案システムは, 大規模ドキュメント空間における整理機能や一貫性管理機能の実現のための基盤として提案しているものである. 例えば, 論文[9] で提案しているようなドキュメント空間上のビュー構築機能等を実現するための実装基盤として役立つことを期待している. したがって, 実行性能は重要な問題の一つである. 本節では, 簡単な予備実験を行う.

図 13 は SPARQL/DS を用いてファイルの一貫性に関する操作を行うための問合せである. 具体的には, (1) ディレクトリの削除を行う前に, そのファイルが他のファイルから参照されていないかどうかをチェックするための問合せと, (2) ディレクトリのコピーを行う問合せである.

問合せについて簡単に説明する. WHERE 節には

(1) ディレクトリ削除のための被参照チェック

```
SELECT ?node
WHERE {
    ?node dsl:path ?path .
    ?node2 dsl:refersTo ?node .
    ?node2 dsl:path ?path2 .
    FILTER (REGEX(?path, "^/test") .
    FILTER (!REGEX(?path2, "^/test")
}
```

(2) ディレクトリのコピー

```
COPY GRAPH {}
WHERE {
    ?node dsl:path ?path .
    FILTER (REGEX(?path, "^/test"))
}
```

図 13 SPARQL/DS による操作の例

RDF データベースに格納されている RDF グラフにマッチするためのパターンを記述する. このパターンは, 各行に記述されている (ノード, プロパティエッジ, 値) を表すトリプルの組合せと, FILTER で始まる行に記述されている選択条件で表現される. 例えば, 問合せ (1) においては, ディレクトリ/test に含まれる各ノードに対して, 外部から参照しているノードが存在しないかどうかのチェックをしている. このように, path プロパティに対する文字列マッチングを用いている理由は, SPARQL ではパターン指定において閉包 (この場合は `dsl:contains` の閉包が必要) が指定できないからである. 問合せ (2) は, `/test` 以下のファイルのコピーを作成している. RDF グラフ上の処理としては, コピーの作成と path プロパティのを変更を行う必要がある.

既存の RDF データベースエンジンを用いてメタデータレイヤでこれらの処理を行うための実行時間を計測した. 実行環境は, CPU Pentium III 933MH, メモリ 512MB, Windows XP Professional SP2, JDK1.5.0, Jena 2.4, MySQL 5.0.21 である. ディレクトリに含まれるノード数は 55 である. 結果を図 14 に示す. このように, わずか 55 ノードで数十秒の時間がかかるため, このままではとても実用的とはいえない. 遅くなった理由の一つは, ディレクトリに含まれるディレクトリやファイルを検索するために, 文字列のマッチングを行っているからである.

したがって, 解決のための一案として, 閉包を計算

問合せ	処理時間 (ミリ秒)
(1)	77,020
(2)	75,318

図 14 問合せ (1)(2) の処理時間

する問合せのためのインデックス構造の利用の検討などが考えられる．参考のために，ディレクトリ構造に含まれるノードを全て参照するような特別なノードをRDF データベースに追加し，それを用いて文字列マッチを利用せず問合せ (1) と同等の結果を返す問合せを実行したところ，551 ミリ秒の時間で実行された．

6. おわりに

本稿では，大規模ドキュメント空間の管理のための基盤アーキテクチャの提案を行った．特に，大規模なファイル群の整理や，ファイル間の関連の一貫性の維持管理技術などの開発に焦点を置いている．既存の情報源やアプリケーションとの互換性を重視し，標準技術に基づくことにより，高度機能を開発するためのプラットフォームとして機能することを目標としている．予備実験を通じて，実際のファイルサーバに格納されているファイル群に，参照先ファイルの存在しない参照が多く存在し，関連するファイル群が散らばって格納されているなどの問題があることを確認した．また，本アーキテクチャで利用する RDF データベースエンジンに必要な実行性能の一部について簡単な議論を行った．今後の課題としては，(1) この基盤を利用した高度機能の実現，および，(2) 基盤そのものの機能や性能の向上，特にスケーラビリティの実現，があげられる．

謝　辞

ゼミなどで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，阪口哲男助教授，永森光晴講師に感謝致します．

参考文献

1) L. Sweeney. Information Explosion. Confidentiality, Disclosure, and Data Access: Theory and Practical Applications for Statistical Agencies, L. Zayatz, P. Doyle, J. Theeuwes and J. Lane (eds), Urban Institute, Washington, DC, 2001
2) Google. Google Destop Search. http://desktop.google.com/ .
3) Apple. Spotlight http://www.apple.com/ macosx/features/spotlight/ .
4) S. shepler et al. Network File System(NFS) Version 4 protocol. RFC 3530, 2003.
5) W3C. Resource Description Framework (RDF). http://www.w3.org/RDF/.
6) D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole, Jr.(1991). Semantic File Systems. 13th ACM Symposium on Operating Systems Principles, October 1991
7) P. Dourish, W.K. Edwards, A. LaMarca, and M. Salisbury, (1999). Presto: An Experimental Architecture for Fluid Interactive Document Spaces. ACM Transactions on Computer-Human Interaction, 6(2), 133-161.
8) G. Rivera, M. C. Norrie: A Database Approach to Global Document Spaces: Replacing Files with Shared, Connected Objects. CoopIS/DOA/ODBASE 2002: 526-527
9) K. Ishikawa, A. Morishima, S. Sugimoto: New Functions of File Systems to Manage Information Shared by Communities. In Proc. SWOD 2006.
10) Microsoft. WinFS http://msdn.microsoft.com/data/ref/winfs/ .
11) Y. Cai, X.L. Dong, A. Halevy, J. M. Liu and J. Madhavan (2005) Personal Information Management with Semex. SIGMOD 2005.
12) W3C. SPARQL http://www.w3.org/TR/rdf-sparql-query/ .