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

2008－DBS－146 (54)
2008／9／23

分散ファイル群高度管理を目的としたファイル関連の発見支援機構

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 我々は，複数の計算機と構成員からなる情報空間で扱われる分散ファイル群を対象に，ファイルやそのユーザに関する大量のメタデータをグラフデータベースに格納することで，情報空間の高度管理を行うためのシステムの研究に取り組んでいる．本グラフデータベースはファイル間の関連を格納するが，これらの関連の発見に必要な計算コストは大きい．本稿はこれらの関係の発見を支援する機構について提案する．本機構では，関連の発見プロセスを次の 2 つの部分プロセスに分解する．(1) 関連の候補を計算する Candidate Generation プロセス，(2)Candidate の検証を行う Inspection プロセス．これらの組合せにより，計算コストの削減を図る．また，これらのプロセスで用いられるプログラムを拡張可能とすることにより，様々な手法を組み合わせた関連の発見を容易に実現する仕組みを提供する．
キーワード ファイル管理，メタデータ

A Mechanism to Support File Relationship Discovery for Advanced Management of Distributed Files

Yuichiro MITSUMORI† and Atsuyuki MORISHIMA†,††

† Grad. Sch. of Library, Information and Media Studaies, 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 We have been developping an advanced management system for distributed files in information spaces containing multiple computers and people, where we store a large size of metadata in a graph database system. The graph database manages relationships between files, but it takes a large cost to find relationships among many files. This paper proposes a mechanism to support the discovery process of such relationships. The mechanism divides the discovery process into the following two processes, in order to reduce the computation costs. (1) The candidate generation process to enumerate relationship candidates, and (2) The inspection process to inspect the generated candidates. In addition, the mechanism has a program interface to easily incorporate different program codes for the discovery into the system, to give us opportunities to employ various types of discovery algorithms.
Key words File Management, Metadata

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，計算機が格納するデータ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，これらのデータ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．例えば，小規模な大学の研究室に格納されているファイルサーバに格納されているファイル数であっても数十万ある事は珍しくなく，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．また，DBMS を用いたデータベース管理とは異なり，ファイル群の管理には人的な要素も大きい．そのため，人の流動性の高い組織などではますますその管理が困難になってしまう．

そこで，我々は，複数の構成員と計算機を含むような知的活動を行うコミュニティの情報空間を対象とした管理システム

InfoSpace Governor の研究を進めている [3] [4] [5] [6] [7]. 本システムでは，既存のファイルシステムでは明示的に扱うことのできない，ファイルのコピーや派生，参照関係などといったファイル間の関連やユーザとの関係を収集・利用することで，効果的な情報空間の管理を目指している．

詳細は 3 章に述べるが，本研究ではファイル間の関係を図 1 のような RDF グラフにより管理している．例えば，同一コンテンツを持つファイル間には Copy というラベル付きエッジが付けられている．Copy エッジを生成する単純な方法としては，すべてのファイルの組合せで比較を行い，内容が同一のものを探し，古いファイルから新しいファイルにエッジを張るという方法が考えられるが，単純なやりかたでは計算に膨大なコストがかかってしまう．また，Copy 以外にも様々な関係を表すラベルがあり，これらをエッジに付与できる仕組みが必要である．

本稿では，このようなファイル間の関連の発見を支援する機構の開発について述べる．本機構には次の 2 つの特徴がある．(1) 関連発見のプロセスを，関連の候補を生成する Candidate Generation プロセスと，そこで生成された候補の検証を行う Inspection プロセスの二つの組合せとすることで，計算コストの削減を図る．(2) 異なるラベルを持つエッジを計算するための様々な手法を実装したプログラムを，上記 2 つのプロセスに容易に組み込み，本システムを拡張可能とするようなプログラムインターフェースを定義する．これにより，様々な関連発見手法を組み合わせて容易に利用できるようにする．

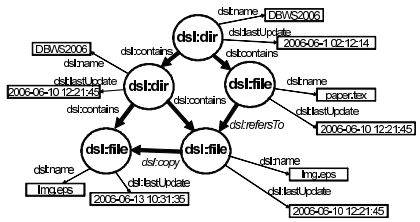

図 1　情報空間メタデータの例

本稿では，この機構の設計と実装を説明し，本システムを用いた幾つかの実験結果を示す．本論文の構成は次の通りである．2 章では，関連研究について紹介する．3 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バナンスプロジェクトについて説明する．4 章で，支援機構の概要と設計について説明する．5 章で，実験結果を示す．6 章は，まとめと今後の課題である．

2. 関 連 研 究

ファイル間の関連を発見し，応用しようという試みは多くあり，それらのほとんどは情報検索の文脈で行われている．渡部らは，ファイルサーバのアクセスログからファイル間関連度を検索に利用する FRIDAL(File Retrieval by Inter-file relationship Derived from Access Log) [8] の研究を行っている．プロセスで使用されているファイルの共起時間の累計，共起回数，間隔等を利用して，共起検索時にファイル間に重みづけをすることで，

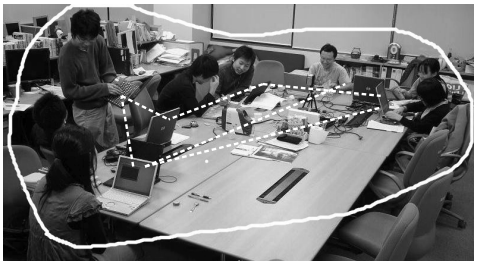
図 2　インフォメーションスペースの例

キーワードを含まない関連ファイルの検索の実現を可能としている．また，Craig らは read や write などのシステムコールの履歴を検索に利用する Connections [9] の研究を行っている．

我々のプロジェクトの特徴は，このようなファイル間の関連を情報検索を含め様々な応用に積極的に利用可能な共通の基盤を構築しようとしていることである．特に，次章で説明するように，我々はこれらの関連を検索だけでなく管理に利用することに重点を置いている．また，既に提案された様々な関連発見手法を，本システムのプログラムインターフェースを用いて本システムに組み込むことにより，多様な手法を統合的に利用可能なフレームワークを提供している．

3. コミュニティ情報空間ガバナンスプロジェクト

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．

3.1 InfoSpace Governor の特徴

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) と呼ぶ複数の構成員とファイルシステムを含む空間を管理の範囲としていることである．図 2 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，これらの問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

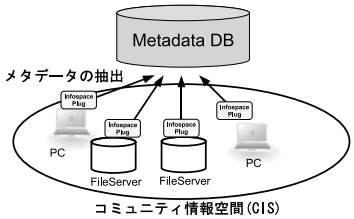

図 3　InfoSpace Governor のアーキテクチャ

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 3)．図 1 はその内容の一部を示したものである．ファイルやディレクトリ以外にも，プロセス，ユーザといった要素も RDF グラフのノードとして表現しており，ユーザとファイルの関連やプロセスとユーザの関連といった情報も利用することができるが，本稿ではファイル間の関連に絞って議論を行う．

3.2　ファイル間の関連

ファイル間の関連のうち主要なものを次に説明する．

Copy: 同じコンテンツを持つ古いファイルから新しいファイル間に成立する関連.

Move: 削除されたファイルを表す特別なノード「ファントム」と，移動先のファイルを表すノード間に成立する関連.

RefersTo: ファイルの参照関係．例えば，tex ファイルや html ファイルから別のファイルを参照している場合に成立する．

Derive: あるファイルとそのファイルから生成されたファイル間の関連．例えば，tex ファイルとそのコンパイル結果の dvi ファイルとの間に成立する．

3.3　InfoSpace Plugs

これらの関連を生成するために，InfoSpace Governor では各参加 PC に InfoSpace Plug というソフトウェアモジュールを組み込む．これは下記の二つの機能を持つ．(1) 操作ログ抽出: 利用者の操作ログを記録する．(2) ファイルシステムクローラ: 各ファイルシステムのコンテンツにアクセスし，必要な情報を入手する．

InfoSpace Plug はこれらの情報から関連を生成し，InfoSpace Server に結果を送信する (図 3).

3.4　アプリケーション例

ファイル操作に関する InfoSpace Governor のアプリケーションの例をいくつか紹介する．より詳細な例は [6] にある．

(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：格納された関連情報を用いて，通常の検索やブラウジングでは発見できない関連ファイルの発見支援を行う．(2) 問合せに結びついた仮想フォルダの作成: 例えば，ある特定のファイルの更新履歴を追っていき，常に最新バージョンのファイルにアクセス可能な仮想フォルダなどが実現できる．(3) バックアップの支援: 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るが，あるタスクやアプリケーションに関係するファイルは必ずしも特定のディレクトリの下にあるとは限らない．これらのファイルを抜けなくバックアップすることが出来る．(4) ファイルの関連に基づく警告：例えば，あるファイルから参照されている別のファイルを削除しようとしたときに警告を出す等が出来る．(5) ファイル整理支援：あるファイルをこの PC から消しても問題ないか (例えばファイルサーバに同じものがあるなど) の決断を支援する．

4.　ファイル間関連の発見支援機構の設計

4.1　概　　要

本章では，ファイル間の関連を発見するための機構の設計について説明する．2 章で述べたように，InfoSpace Govenor ではファイルだけでなく，ユーザやプロセスなども含めてノードとして格納をして，その間の関連を利用するが，本稿ではファイル間の関連発見に内容をしぼり，その設計について議論していく．本機構の目的は，図 1 で示したような RDF グラフで表現されるメタデータ DB に対して，特定の関連を表すラベル付きエッジを新たに付与することである．

本機構では，関連発見のプロセスを Cadidate Generation プロセスと Inspection プロセスの 2 つに分解する（図 4）．それぞれのプロセスでは，InfoSpacePlug を用いて計算用に収集したデータを利用して計算する．

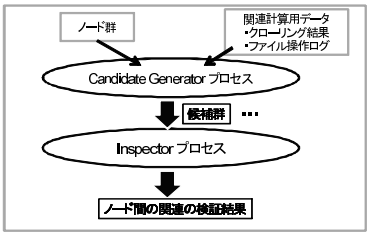

図 4　本機構のアーキテクチャ

Candidate Generation プロセス．Candidate Generation は，与えられたファイルを表すノード群の中から，特定の関連を持つ候補ペアを探すプロセスである．このプロセスでは，ディレクトリのクローリング結果やファイル操作の履歴を元に，関連を持つ可能性のある組み合わせを候補ペアとして生成していく．候補ペアは，実際には 2 つのファイルの識別名とその関係を表すラベル名からなり，RDF 表現におけるトリプルと等価である．最終的に生成した候補ペアの集合は，次の Inspector プロセスへと渡される．出力される関連候補には，前章で説明したラベルを持つ関連の他，特定できないが何らかの関連を持つと考えられるものわを表す特別なラベル"*" を持つもの含まれる．これは，次の Inspection プロセスで通常のラベルに置き換えられる．

Inspection プロセス．Inspection では，Candidate Generation プロセスから渡された候補ペア群のすべてに対して，実際にそのような関連にあるかを検証する．基本的にはその関連が本当に正しいと判定できるような検証方法を採るが，関連の種類によっては完全な検証ができない場合がある．その場合，Inspector プロセスではより可能性の高い候補であるかの検証を行う．

4.2 インタフェース

次に，本機構のアーキテクチャにおけるプログラムインタフェースについて説明する．図 4 の 2 つのプロセスでは，実際の関連の計算処理を図 5 の抽象クラスを実装したクラスのインスタンスを利用して行う．

これまで述べてきたように，ファイル間の関連は RDF トリプルによって表現されており，関連の計算処理を行うクラスでは，次のような Triple クラスによって扱う．

```
class Triple{
  String resorce,  String subject,  String property;
}
```

resorce と subject は，各々に対応するファイルの識別名となっており，property にはその 2 つの関係を表すのラベル名が入る．

CandidateGenerator と Inspector クラスを継承するクラスは具体的には次を行う必要がある．(1) それぞれ CandidateGenerate と Inspect メソッドの実装を用意する．(2) どのラベルのためのクラスであるかを表すための定数を保持する．クラスを表す定数は，コンストラクタを定義するときに指定する．図 7 のコンストラクタ定義にその例がある．クラスを表す定数は，Label クラスで定義している．図 6 に，Label クラスで定義しているクラスの幾つかを示す．

```
package infospace;

Class Label {
  /* Copy - 同一のファイルである関係を表すラベル */
  public static final int COPY = 1;

  /* Move - 移動ファイルやリネームファイルの関係を表すラベル */
  public static final int MOVE = 2;

  /* RefersTo - 参照関係にあるファイルを表すラベル */
  public static final int RefersTo = 4;

  /* Derive - 派生関係にあるファイルを表すラベル */
  public static final int Derive = 8;

}
```

図 6　代表的なラベル名を持つ Label クラス

図 4 における 2 つのプロセスのそれぞれにおいて，上記の用に定義されたクラスのオブジェクトへの参照のリストが管理されている．ある特定のラベルの関連を生成する際には，システムはこれらのリストにアクセスし，そのラベルの定数を持つ Generator と Inspector を用いて処理を行う．このように，提案アーキテクチャは多様な関連発見手法を組み合わせて 1 つの機構で利用することができる拡張可能なインタフェースを提供している．

4.3 インタフェースを用いた関連発見プロセスの主な実装方法

次にインタフェースの実装方法について述べる．

Candidate Generator クラス このクラスでの関連候補発見手法は主に，(1) ルールベース，(2) ファイル操作トレースベース，の 2 つがある．

（1） ルールベース: 特定のルールにしたがって，ディレクトリをクローリングして候補を探す．例えば，同一のファイルの関係であれば，「ファイル名とサイズが同じであれば候補とする」といったルールに従って候補を決定する．

（2） ファイル操作トレースベース: 3 章で述べたように，Infospace Governor システムでは，ファイル操作の履歴をログとして蓄積している．このログを見て候補を発見する方法である．例えば，copy 関連の候補を発見する Candidate Generator の場合には，ほぼ同時刻に同一名のファイルの読み書きが行われたことがログに記録されていれば，それらの間に copy 関連がある可能性があると考え，その関連を候補とする．

Inspector クラス. このクラスでの検証では，コストが大きくても確かにその関係だと判断できる検証を行う．例えば，同一ファイルの関係であれば，候補が実際に同一であるかをファイルの中身を比較することで確認する．すべてのファイルに対してこのような処理を行えば，膨大な計算コストがかかってしまうが，Candidate Generator の段階である程度可能性の高い候補のみに絞られているので，比較的少ない計算コストで済ますことができる．

4.4 各ラベルに対する実装例

各ラベルに対して，どのような CandidateGenerator クラスと Inspector クラスの実装が考えられるのか例を示す．

4.4.1 Copy ラベル

この方向は，ログベースでは明確に決定できるが，クローリングなどによるルールベースでは決定が難しい．

(1) Candidate Generator: 下記の 2 つが考えられる

- ルールベース: 同名かつファイルサイズと最終更新日時が同じものを候補とする．この時，作成日時が古い方をコピー元とする．
- ファイル操作トレースベース: ファイル操作のログにおいて，同時刻に同名のファイルの読み込みと生成があった場合に候補とする．

(2) Inspector: 2 つのファイルの中身を比較し同一か検証し，一致すれば True を返す．

図 7 は，Copy ラベルを対象とした Candidate Generator と Inspector の実装例である．

CandidateGenerator を実装した CopyCandidateGenerator では，対象のディレクトリ以下のファイルを走査する．走査では各ファイルのパスを，ハッシュを用いて，そのファイル名＋ファイルサイズ＋最終更新日時をキーとする文字列のリストに格納していく．すべてのファイルパスの格納が終わったら，makeTriples メソッドによりハッシュテーブルから Triple の配列を生成する．このメソッドのコードは省略するが，ハッシュテーブルに登録された各リストの要素を，作成日時でソートし，隣合う要素の組をコピーの候補としている．Inspector を実装した CopyInspector では，それぞれの候補の中身を 1 バイトずつ比較していき，一致した場合にのみ True を返すようにしている．例ではプリミティブな比較方法を用いているが，実際の実装では比較処理を効率化することで，処理速度の向上を行っている．

4.4.2 Move ラベル

(1) Candidate Generator: ファイル操作のログにおいて，同時刻に同名のファイルの削除と生成があった場合に候補とする．

(2) Inspector: Move ラベルの検証は，元ファイルが存在しない

抽象クラス名	メソッド名	説明
CandidateGenerator	Triple[] candidateGenerate (String[] nodes)	ノード名の集合 nodes から，特定の関連を持つノードの組み合わせを RDF トリプルの集合として返すメソッド.
Inspector	bool inspect (Triple triple)	nodeA が nodeB に label の関係を持つかどうかを検証し，持っていれば True を返すメソッド.

図 5　実装される抽象クラス

ため，厳密な検証は難しい．移動元を表すファントムノードに Copy ラベルのエッジが存在する場合はこれを利用するが，それ以外の場合は，より正しそうな関係であるかを検証する．

4.4.3　RefersTo ラベル

(1) Candidate Generator: 次のルールを用いて候補を発見する．

- tex ファイルに対して，同フォルダ以下に画像ファイルが場合に候補とする．
- html ファイルに対して，同フォルダ以下に画像および html がある場合に候補とする．

(2) Inspector 実際に，参照元候補ファイルを全文検索し，ペアのファイルへのリンクが含まれていれば True を返す．

4.4.4　Derive ラベル

(1) Candidate Generator: ログファイルを利用する．同時刻に，拡張子を除いたファイル名が同じファイルが読まれ，生成されている場合に候補とする．

(2) Inspector: ファイルタイプが異なる場合が多いため，Derive の完全な検証は難しい．2 タイプの Inspector が考えられる．一つはバージョン違いなどのファイルタイプが同じ場合である．テキストベースのファイルであれば差分を比較するなどの方法により完全な検証が行える場合も多いと考えられる．もう一つはファイルタイプが異なる場合である．こちらは正しいことを保証することが難しいことが多いので，その候補が実際に正しい関連である可能性が高いかを判断するプログラムになる．具体的には，ファイルのメタデータを比較したり，過去の操作ログからも同じファイル間に Derive 関連がある可能性を示す根拠が数多く存在するかを調査する等である．

4.4.5　＊ラベル

アスタリスクに関しては Candidate Generator のみが定義される．

Candidate Generator:　2 章で紹介した研究を含む，既存の関連発見手法を利用して候補を決定する．

5. 実　　験

5.1 概　　要

refersTo と copy ラベルを持つ関連に対して CandidateGenerator クラスと Inspector クラスを実装し，実験を行った．

5.2 実験方法

4.4 節で説明した実装例の中で，4.4.1 節の Copy ラベルのルールベースの Candidate Generator の実装 (以下 GenC) および Inspector の実装と，4.4.3 節の RefersTo ラベルのルールベースの Candidate Generator の実装 (以下 GenR) および Inspector の実装を用いて実験を行った．この 2 つのラベルは，Inspection プロセスにおいて完全な検証が行える関連である．したがって，実験は Candidate Generator の結果の評価を行うものになる．

```
import infospace.Label;

/* Copy ラベル用の CandidateGenerator クラス */
class CopyGenerator extends CandidateGenerator{
  CopyGenerator(){
    super(Label.COPY);
  }

  /* クローリング結果から候補を探す */
  Triple[] candidateGenerate (String[] nodes){
     HashTable<String, ArrayList<String>> table
                  = new HashTable<String, ArrayList<String>>();

     /* ルールベースのクローリング*/
     for (String node :nodes){
       File f = new File(node);
       // ファイル名+サイズ+最終更新日時をキーとする
       String key
            = f.getFileName() + f.length ()+ f.lastModified();

       if (table.containsKey(key)){
         table.get(key).add(node);
       }else{
         ArrayList<Triple> list = new ArrayList<Triple>();
         list.add(node);
         table.put(key, list);
       }
     }

     //table に格納された候補のペアから Triple クラスの配列を生成
     return makeTriples(table);
}

/** Copy ラベル用の Inspector クラス **/
class CopyInspector extends Inspector{
  CopyInspector() {
    super(Label.COPY);

  /* inspect - inspect メソッドの実装
  bool inspect (Triple triple){
    File r = new File(triple.resorce);
    File p = new File(triple.subject);

    if (compareFiles(r, p))
      return true;
    else
      return false;
    }
  }

  /* compareFiles - f1 と f2 の中身が同一なら True を返す */
  boolean compareFiles(File f1, File f2) throws IOException{
    if (f1.length() != f2.length()) return false;
    FileInputStream fs1 = new FileInputStream(f1);
    FileInputStream fs2 = new FileInputStream(f2);
    int r1, r2;

    while (true){
      r1 = fs1.read();
      r2 = fs2.read();

      if (r1 == -1 && r2 == -1) return true;
      else if(r1 != r2) continue;
      else return false;
    }
  }
}
```

図 7　抽象クラスを用いたプログラム例

実験では，それぞれの関連を持つファイルが一定量存在するディレクトリを対象とした (図 8)．これらのディレクトリから事前に正解セットを作成し，組み込んだ Candidate Generator が出力する関連候補の再現率と適合率を調べた．一般に，Candidate Generator の出力結果に関しては，再現率の方が重要である．なぜなら，Inspector によって正しくない関連は除去されるからで

ある．しかし，適合率があまり低いと Inspector の負担が増えるため，適合率も高い方が望ましい．

フォルダ名	内容	正解関連数
tex	Dir: 204　File: 1597 ・tex ファイル: 91 ・画像系 (jpg, png, gif, eps): 357	RefersTo: 202 Copy: 650
Project	Dir: 100014　File: 34816 ・tex ファイル: 793 ・画像系 (jpg, png, gif, eps): 8122	RefersTo: 3712
CIS	Dir:457　File: 8436	Copy: 3712

図 8　対象となるディレクトリ

5.3 実験結果

実験結果を図 9 に示す．

CandidateGenerator	フォルダ名	再現率 (%)	適合率 (%)
GenR	Tex	100.0	8.94
	Project	99.9	22.2
GenC	Tex	95.6	100.0
	CIS	98.9	99.9

図 9　実験結果

それぞれの CandidateGenerator 評価の考察は以下の通りである．

GenR．再現率に関しては，ほぼ全体を網羅できていることが分かる．漏れたのは，上位ディレクトリ下の画像を参照しているものに関してであった．これは GenR の候補決定におけるルールが，網羅性の高いものであったことを示している．しかし，一方で適合率はかなり低くなっており，候補の中に多くのノイズが含まれていたことが分かる．この原因は，今回の実験で用いたルールが，すべてのサブディレクトリのすべての画像を候補としていたためだと考えられる．実際には，階層差が極端な参照関係は少なかったので，階層差を閾値として利用することにより，適合率の上昇が期待できる．

GenC．再現率・適合率どちらも高い結果を得ることができた．特に適合率に関しては，ほぼ 100%に近い結果を得ることができ，GenC の候補決定のルールの精度が高いことを分かる．漏れたのは，別名に変更されているファイルに関してであった．これは，GenC が名前をベースにして候補を決定しているためであり，ファイルのリネームの履歴を利用できる，ログベースの CandidateGenerator を利用により補完ができると考えられる．

6. まとめと今後の課題

本稿では，分散ファイル群高度管理のために，ファイル間の関連発見を支援する機構の開発について述べた．本機構では様々な関連発見手法を統合的に利用可能なフレームワークの実現を目指しており，関連発見のプロセスを，2 つの部分プロセスに分解し，それらのプロセスに多様な関連発見手法を組み込むためのプログラムインターフェースを用意することで，その実現を図る．今後は，より多様な関連発見手法の実装と評価，および組み込まれた発見アルゴリズムの実行タイミングなどの総合的な調整の仕組み等について研究を行う予定である．

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の杉本重雄教授，阪口哲男准教授，永森光晴講師に感謝いたします．本研究の一部は科学研究費補助金若手研究 (B)(#20800076) による．

文　献

[1] W3C. Resource Description Framework (RDF). http://www.w3.org/RDF/.
[2] J.Widom and S.Ceri.“Active Database Systems: Triggers and Rules for Advanced Database Processing”, Morgan Kaufmann, 1996.
[3] 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.
[4] 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.
[5] 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.
[6] 三森祐一郎，森嶋厚行：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，情報処理学会研究報告 Vol.2007, No.65 (2007-DBS-143), pp. 180-194. 電子情報通信学会技術研究報告 Vol.107, No.131, DE2007-53, pp. 189-194, 2007 年 7 月.
[7] 三森祐一郎，森嶋厚行：コミュニティ情報空間管理のための制約記述と処理手法の提案，電子情報通信学会第 19 回データ工学ワークショップ (DEWS2008), 2008 年 3 月.
[8] 渡部徹太郎，小林隆志，横田治夫：キーワード非含有ファイルを検索可能とするファイル間関連度を用いた検索手法の評価，電子情報通信学会 第 19 回データ工学ワークショップ (DEWS2008) 論文集 E10-6, Mar 9-11, 2008.
[9] Soules, C. A. N., Ganger, G. R. : Connections: using context to enhance file search. In Proceedings of the 20th ACM Symposium on Operating Systems Principles, pp.119-132. 2005.