平成 20 年度

卒業論文

GnuPG を用いた
研究記録管理・公開・検証システムの構築

指導教員　小林　聡　准教授

島根大学　総合理工学部

数理・情報システム学科　計算機科学講座

s023027　加藤未来

目次

1 はじめに

昨今，研究成果の捏造が社会問題となったことは記憶に新しい．技術的にこれらの問題の発生を食い止める事は困難であるが，改竄が困難な手法によって研究記録が保存されているならば，少なくとも研究記録の検証に関しては大いに助けになるであろう．この際，研究記録の真正性と非改竄性の保証をいかに行うかが課題となる．

このような課題に対して，原田らは電子カルテを念頭に，モデルの提案を行っている[1]．また，陳らは XML 文書の時刻認証を伴った管理を試みている[2]．同様なサービスとして，SNS/blog 機能に電子文書へのタイムスタンプ付加機能を加えた，SNS/blog サービスも存在する[*1]．しかし，これは日本電子公証機構のサービスを利用しているため，比較的高額なサービスとなっている．このような，タイムスタンプを活用したサービスは DVCS（Data Validation & Certification Server Protocol）や TAP（Trusted Archival Protocol）などに分類される．DVCS は，電子文書のアーカイブは行わないことを原則としており，対して TAP はアーカイブも行うことを原則としている．また宇根らは DBCS や TAP で用いられる各種タイムスタンプによる可用性・安全性に関して報告している．[3]

また，今日，e-mail や World Wide Web に代表されるインターネット環境は，多くの人にとって重要なコミュニケーション・メディア/コミュニケーション・ツールとなっている．そのようなネットワーク環境を利用し，関連した研究を行う者同士が，論文の公刊前に，研究の進め方や研究・実験の成果について互いに意見交換が可能な環境が提供されれば，研究を進める上で大きな利点があると考えられる．

そして記録，コミュニケーションと並んで，コンピュータが果たすことができる重要な役割として，ストレージ機能がある．研究資料や実験試料，あるいはプログラムなどを手軽に公開・参照できる機能が実現できれば，研究者間での研究資料や実験試料などの流通も盛んになり，研究自体が活性化するであろう．

ただし，記録機能においても，ストレージ機能においても，研究の遂行途中においては一般には秘匿したい，あるいは秘匿する必要のある情報も存在する．そのため，情報を公開する範囲を柔軟に制御できなければならない．

現在，World Wide Web を用いた SNS などのサービスにおいて，記事の公開範囲はある程度の制御が可能ではあるが，概ねその設定の自由度は低い．

Masui らや江渡らは，QuickML[4]や qwikWeb[5]により，情報にアクセスできる者を柔軟に変更可能とするシステムの構築と運用を試みた．また，SNS を基盤としたシステムとして，永田らは Enzin を[7]，高井らは ACS を開発し[6]，運用実験を行った．高田らは，公開 Web-DB と Web-DB 管理システムを分離可能であり，かつきめ細かなアクセス制御が可能な Web-DB 管理システムを構築している[8]．また，一般の SNS/blog にも，記事ごとに公開範囲の設定を可能とするサービスが登場しており[*2]，公開範囲の柔軟な制御の需要が伺える．

本研究では上述のような研究記録などの管理を主目的として，公開範囲を柔軟に変更可能かつ，記録の真正性および非改竄性を可能な限り保証する，研究記録の管理・公開・検

*1 Synest LaboNote（http://www.labonote.jp）

*2 Media Wagon(株式会社エイミー http://mw.aimy.jp)

証を行うシステム, arχves[*3]の構築を試みた. 特に, 安価かつ, DVCS でもなく, また TAP でもない, 第三のあり方を模索した.

*3 arχves: Archive system for Research logs by Kato=Shima-nuki/Kobayasi Satoshi, and VErification System → ARKSVES. KS を x に, さらに x を類字形の χ に置き換え.

2 システムの特徴

2.1 概要

本システムは，研究記録などの管理を主目的とし，公開範囲を柔軟に変更可能かつ，記録の真正性および非改竄性を可能な限り保証するシステムの構築を目的としている．

そこで，データの記事の真正性と非改竄性の保証をある程度可能とするために，記事データの投稿時に，記事データに対して二重署名の処理を行う実装を行い，また平易な操作で検証を行えるようにした．また，電子情報の簡便な記録・保存と扱いやすさのために，基本的な操作は近年広く普及しているSNS/Blogをベースとした実装とした[17]．そして柔軟な公開範囲の実装のため，一般のSNS/BlogのようにBlogコンテンツ全体の公開範囲を指定するではなく，投稿される記事ごとに公開範囲を指定し，公開範囲の設定段階も自由度の高い指定を可能とした．

特徴を要約すると，以下の3点となる．

- 真正性と非改竄性の一定の保証を可能とする，電子データに対する署名および検証機能
- SNS/Blogをベースとした実装
- 記事ごとに公開範囲の指定を可能

以降の各節で機能の詳細について説明していく．

2.2 利用ツール

本システムで使用したツール等を表-1に示す．

Ruby on Railsは本システムの基幹となるSNS/Blog機能の構築を中心に活用し，データベースにはMySQLを利用した．電子署名の付与および検証については，全てGnuPGを活用している．

表-1 利用ツール

開発言語:	Ruby ver. 1.8.6
フレームワーク:	Ruby on Rails ver. 1.2.6
公開鍵暗号ソフト:	GnuPG ver. 1.4.9
データベース:	MySQL ver.5.0.27
ウェブサーバ:	Apache ver. 2.0.6
SSLライブラリ:	OpenSSL Ver.0.9.8

2.3 記事の真正性と非改竄性の保証ーデータへの二重署名

2.3.1 検印モデル

企業における，記録の真正性および非改竄性を保証する方法として，部下が書いた記録(日報など)に，上司が検印を捺すことが広く行われている．このような，記述者とは異なる者が検印あるいはそれに類する行為/操作を行うモデルを，本論文では「検印モデル」と呼ぶ．検印モデルの例としては，原田らは，公開鍵暗号を用い，記述者が自分自身の秘密鍵で電

子署名をすると共に，文書管理システムに持たせた秘密鍵を使った電子署名により検印をするモデルを提案している[1]．本論文においても，この検印モデルを採用した．ただし，原田らの提案においては，文書管理システムが検印を行うが，本論文で述べるシステムにおいては，記事やデータに検印を行うサーバ(以下「検印サーバ」)と，文書を保管するサーバ(以下「文書サーバ」)」は異なることを想定している．文書サーバは，ユーザが通常，記事を記述する際に用いるサーバとした．

これはタイムスタンプ技術[9]と類似しているが，文書作成者の真正性を，ユーザ自身の電子署名という形で明示的に示している点が異なる．

検印については，大学規模の組織を越えて，相互に検印を行うモデルを想定している．このようにシステムを分散することで，導入および管理コストの低減を期待している．

このようなシステムの利用サイトが相互に検印を行う手法により，真正性や非改竄性の保証の強度は，企業が提供するサービスに比べて幾分低くなると思われる．しかし，企業が提供するタイムスタンプ・サービスは高価であったり，利便性に欠ける部分がある．そこで，本論文で示すシステムと，企業が提供するサービスとは使い分け/棲み分けができると考えられる．また，ラボノートの運用および証拠能力から，本システムも十分な能力を持っていると考える．

2.3.2 検印モデルによる実装

記事を記入後，投稿ボタンをクリックすることで以下のような手順で二重署名が行われる．

① 投稿された記事に対して，文書サーバがユーザの秘密鍵で署名を行う．

② ①で作成されたデータを検印サーバへ転送

③ 文書サーバから送られたデータに対し，検印サーバがタイムスタンプを記録し更にデータ全体に対し署名を行う．そして，署名部（図 1，2 参照）のみを検印サーバは保存する．

④ 検印サーバは文書サーバに 2 重に署名済みのデータを送り返す

⑤ 文書サーバにデータを保存

以上の一連の流れをを図式化したものが図 1 である．図 2 は，実際に二重署名が行われた文書の構成を示している．

文書サーバと検印サーバ間のデータの通信には，データ保護のため SSL を用いている．

また文書サーバと検印サーバのそれぞれで行われる署名の直前に，データに対して時刻情報を書き込み，タイムスタンプのシンプル・プロトコルと同様な作業を行っている．

一般的に普及しているタイムスタンプではデータ漏洩への対策も考慮し，ハッシュ値を用いてデータの非改竄性の保証を行っているが，本研究では閲覧者がスクリプトによらない検証を行えるようにする為，Clear 署名[*4]で 2 重署名作業を行っている（参照：「2.3 記事の真正性と非改竄性の保証」）．

*4 Clear 署名：GnuPG で復号しなくても，文書の中身を確認できる形で残して署名を付与する方式．署名を行う対象そのものは暗号化させず，署名を行う（具体例：図 2）．

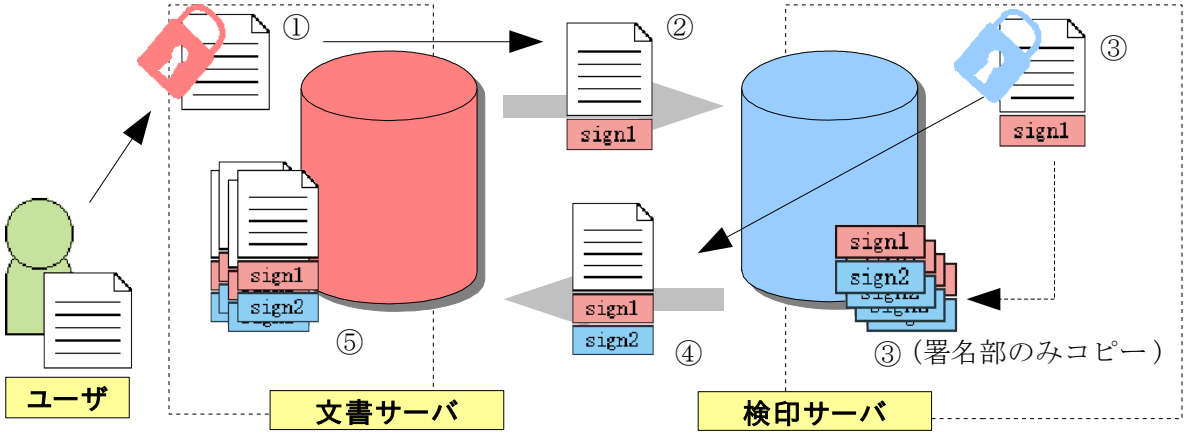


図 1　二重署名の処理の流れ

図 2　2 重署名された記事データの構成

2.4　記事および添付ファイルに対する検証機能

前節で二重のデジタル署名によって，投稿されたデータの真正性および非改竄性の保証が行われているが，それが確かであるかどうかは署名の検証作業によってのみ確認が可能である．

また，電子商取引推進協議会，認証・公証 WG [15]のガイドラインでは，デジタル署名の有効性を長期的に維持する為の用件として，次の 4 つを挙げている．

1. 署名検証時に，署名再検証に必要な情報を明確にしておくこと
2. 署名検証時の時刻を明確にしておくこと
3. 署名再検証時に必要な情報を改竄検出可能な状態にすること.
4. 署名再検証時に必要な情報を保存すること

（1）は，検証を行う際に確かにその署名が有効であることを示す情報（あるいは失効情報）を明示しておくこと. そして，署名の検証が確かに保証されていることを確認した日時を明確にし，（2）その時点まで確かにデータの保証されていることを記録し，（3）再検証を正しく行えるように改竄をの検出を可能にして，(4)署名の再検証に必要な情報を保存して，第三者でも再検証を行えるようにしておくということである. これらの用件を満たすことがデジタル署名の有効性の長期的な維持には必要であるとされている.

このように，本当にその署名が確かであることを確認したり，あるいは長期的な有効性の維持には「署名の検証」が必要となってくる. 本システムでは，その検証を簡単に行える機能を実装している.

各記事およびデータには，「簡易検証」機能および「通常検証」機能が用意され，閲覧者はそれらのリンクをクリックすることで簡便な検証を行なえる. なお，「簡易検証」は文書サーバのみで，ユーザの公開鍵と検印サーバの公開鍵を用いて行う検証である. 通常検証は，検印サーバに記録されている署名データとの照合も含めて検証を行う. 検証を行なった画面例を図 3 に示す. また，改竄が行われた場面の画面例を図 4 に示す.

しかし，本システムはコンピュータ・プログラムとして実現されている以上，スクリプトを書き換えることにより，あたかも検証を行なったように見せかけることも可能である. そのため，閲覧者が独自でも検証を行なえるよう，ユーザおよび検印サーバの公開鍵は画面上から取得可能としている（図 5）. 検証手順については，ブログリストのサイドバーから，手引きを参照できるようにしている.

検証結果

投稿者:管理者

検証記事タイトル:公開鍵暗号を用いた研究記録管理

投稿日時:2008-08-27 13:54:37

ユーザ主鍵フィンガープリント:
F4BB BDCE 6667 7ACB 60AE AC0B 15FB B2C1 8AAE EFB8

検証結果:サーバー署名

gpg: 08/27/08 13:54:39にDSA鍵ID 8696ACBDで施された署名
gpg: “server (sarver's key)”からの正しい署名

検証結果:ユーザ署名

gpg: 08/27/08 13:54:37にDSA鍵ID 8AAEEFB8で施された署名
gpg: “MikuKatou ”からの正しい署名
gpg: 警告: この鍵は信用できる署名で証明されていません!
gpg: この署名が所有者のものかどうかの検証手段がありません。
主鍵の指紋: F4BB BDCE 6667 7ACB 60AE AC0B 15FB B2C1 8AAE EFB8

検印サーバ側の署名部との照合結果

検印サーバ側の署名部と一致しています。
・ユーザ署名とサーバ署名が共に一致しています。

図 3　検証結果(成功時)

検証結果

投稿者:管理者

検証記事タイトル:公開鍵暗号を用いた研究記録管理

投稿日時:2008-08-29 14:56:22

ユーザ主鍵フィンガープリント:
F4BB BDCE 6667 7ACB 60AE AC0B 15FB B2C1 8AAE EFB8

検証結果:サーバー署名

gpg: 08/29/08 14:56:23にDSA鍵ID 8696ACBDで施された署名
gpg: “server (sarver's key)”からの 不正な 署名

検証結果:ユーザ署名

gpg: 08/29/08 14:56:22にDSA鍵ID 8AAEEFB8で施された署名
gpg: “MikuKatou ”からの 不正な 署名

検印サーバ側の署名部との照合結果

検印サーバ側の署名部と一致しています。
・ユーザ署名とサーバ署名が共に一致しています。

図 4　検証結果(失敗例[*5])

*5 本文に対してのみ改竄が行われている場合の検証失敗例. 検印サーバにも保存されている署名部に対しての改竄は行われていない.

図 5　公開鍵のダウンロード画面

2.5　SNS/Blog ベース

2.5.1　概要

本システムは，研究室～学科程度の規模を単位としての記録の保存を想定しており，過度に分散せず，過度に集約せず，適度に分散しつつネットワークを構成するシステムを想定している．　それに加えて電子情報の簡便な記録・保存と扱いやすさのために，基本的な操作は近年広く普及している SNS/Blog をベースに実装している．

本システムにアクセスすると図 6 のようなトップページコンテンツが表示される．ここでユーザとして認証を受け，ログインすると，図 7 のような画面になる．

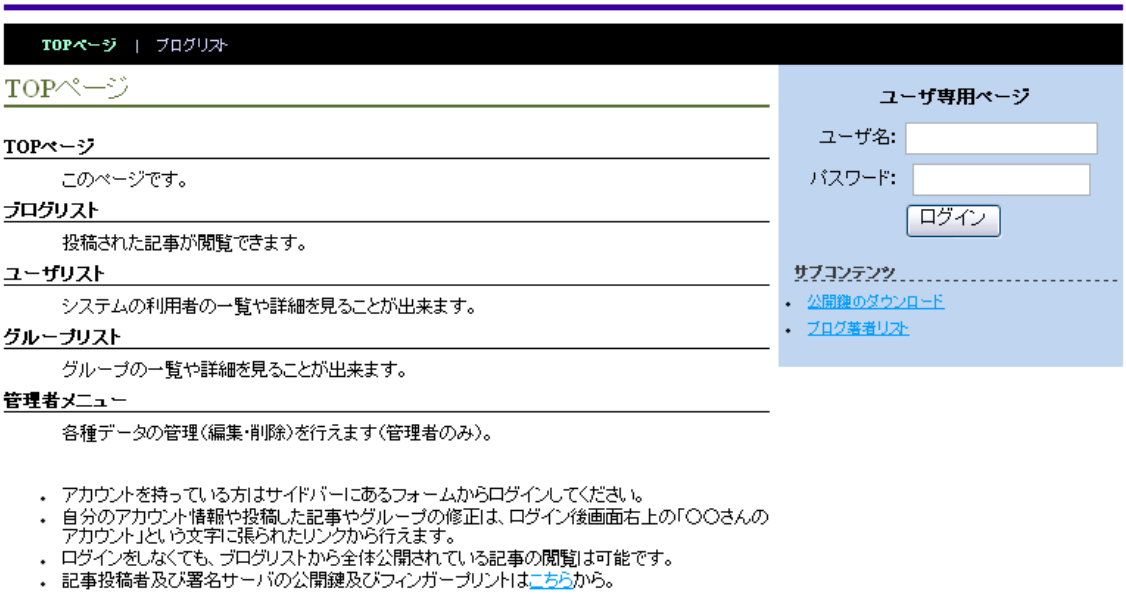


図 6　TOP ページ（初期表示）

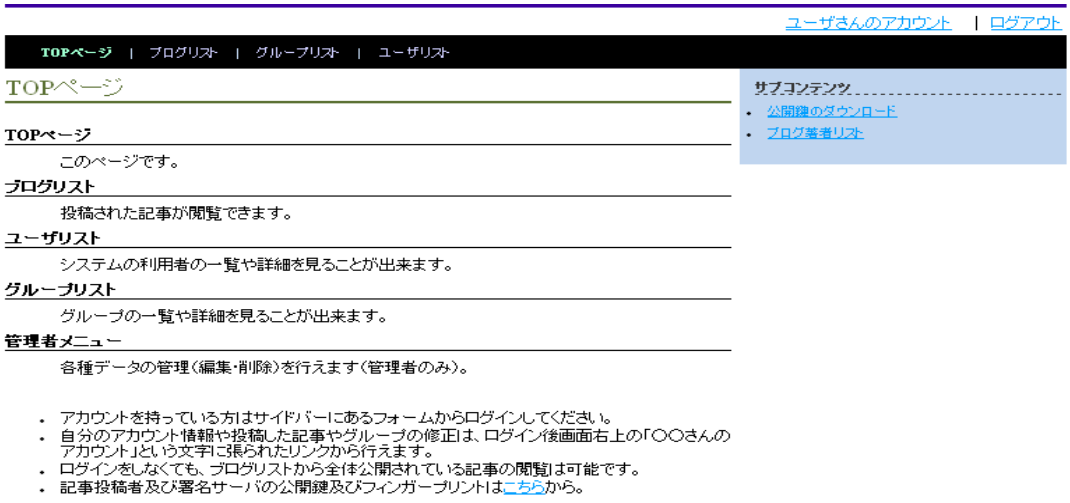


図 7　TOP ページ(ゲスト・ユーザ認証後)

ページは大きく上からメニューバー，メインページ(左)，サイドバー(右)で構成されている(図 8)．

メニューバーには各コンテンツへのリンクが張られており，メインページには各コンテンツの内容が，サイドバーにはコンテンツ内のサブコンテンツ・関連ページへのリンクやコンテンツ内の検索機能が表示される．

またログインすることでメニューバーの右上部にアカウント名が表示され，アカウントメニューへのリンクとログアウト用のリンク*6が表示される(図右上部丸)．メニューバーおよびサイドバーは，ログインした利用者の権限ごとに表示されるコンテンツが異なる(詳細は「2.5.2 利用者の分類」)．

以下でコミュニティを形成する利用者およびグループや，利用者による記事の投稿をメインとした交流とその管理を行う為のコンテンツ毎の機能について示す．

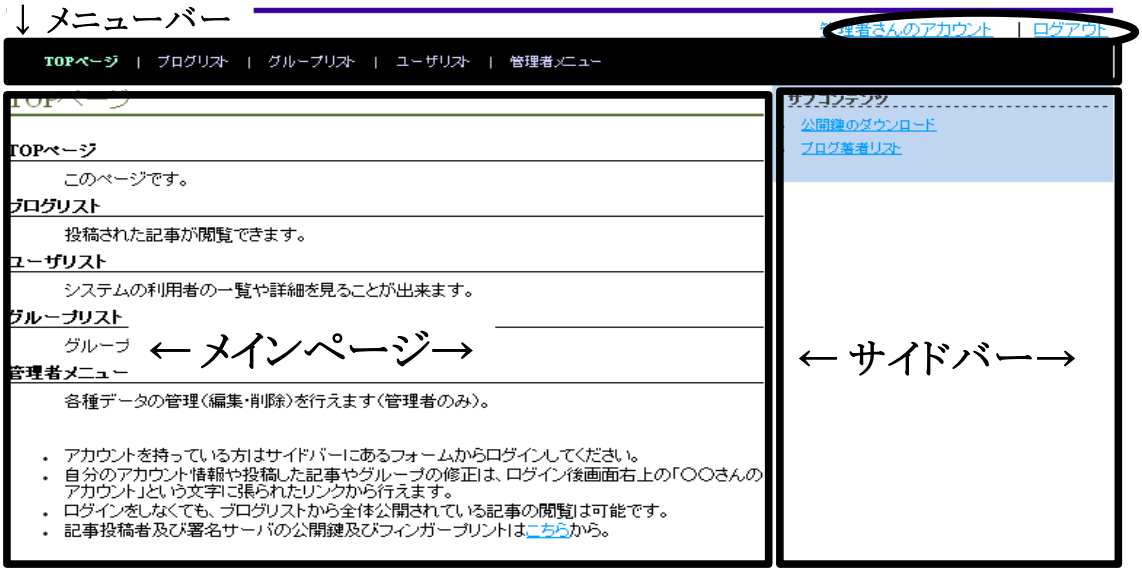


図 8　画面構成(TOP ページ/管理者認証後)

*6 各図とも分かりやすさのために「管理者」「ユーザ」などと表示されているが，実際はハンドル名が表示される．

2.5.2 利用者の分類

本システムの利用者は，本システム上に何らかのアカウントを持っている．

このシステムの利用者は，システムを利用できる権限によって「ゲスト」，「ユーザ」，「管理者」に分類される．

利用者ごとの大まかなアクセス権限を示したものが表 2 である．

「ゲスト」は，システム内の blog 記事の投稿やグループの作成は行えないが，グループに所属し，記事の閲覧は可能な利用者である．

「ユーザ」は，システム内に自身の blog を作成し，記事を投稿することができる利用者で，グループの新規作成や，自身のアカウント情報を修正することができる．また，自身が投稿した記事の各々について公開範囲の変更や，カテゴリの変更が行える．

「管理者」は，システムの管理者であり，管理者メニューにアクセスできる唯一の利用者である．

本システムは TOP ページから ID とパスワードを利用したユーザ認証を受けてログインすることで，利用者の権限に応じてコンテンツが表示される．また，システムにログインしていなくても，公開範囲に制限を加えられていない記事に関しては閲覧が可能である．

表中にある「アカウントメニュー」はログインした利用者自身の情報の編集，削除が行えるコンテンツであり，また「管理者メニュー」はシステム全体の各データの編集，削除が行えるコンテンツである．このアカウントメニューと管理者メニューの項目については表 3 に示す．

表-2 アクセス権限

利用者 / コンテンツ	管理者	ユーザ	ゲスト	一般
TOPページ	○	○	○	○
ブログリスト	○	○	○	○
グループリスト	○	○	○	
ユーザリスト	○	○	○	
アカウントメニュー※1	○	○	△	
管理者メニュー※2	○			

△：アカウント情報の閲覧のみ可能

表-3 メニュー項目

※1アカウントメニュー	※2管理者メニュー
アカウント情報の閲覧 アカウント情報の変更 管理グループの変更 ブログタイトルの変更 カテゴリの作成・変更 ブログ記事の閲覧範囲の変更 添付ファイルの削除	ユーザー管理 グループ管理 ブログ記事管理 コメント管理 カテゴリ,ブログ名管理

2.5.3 グループについて

グループは，一人以上の利用者からなる利用者の集合である．その種類は，階層別に「ルートグループ」，「トップレベルグループ」，「子グループ」に分けられる（図 9）．

ルートグループは，アカウントを持つ利用者全てが所属するグループである．そのため，利用者は必ず一つ以上のグループに所属していることになる．

トップレベルグループは，ルートグループの直下に作られるグループであり，利用者が自分で作成できる一番上位のグループである．

子グループは，トップレベルグループよりも下層に作られるグループを指す．必ず親としてトップレベルグループか，子グループが一つ必要となる．

以上より，一つでもトップレベルグループが存在する時，子グループの作成，つまり階層を持たせる事が可能となっている（図 9）．グループの種類と命名規則についての詳細は表 4 に示す．

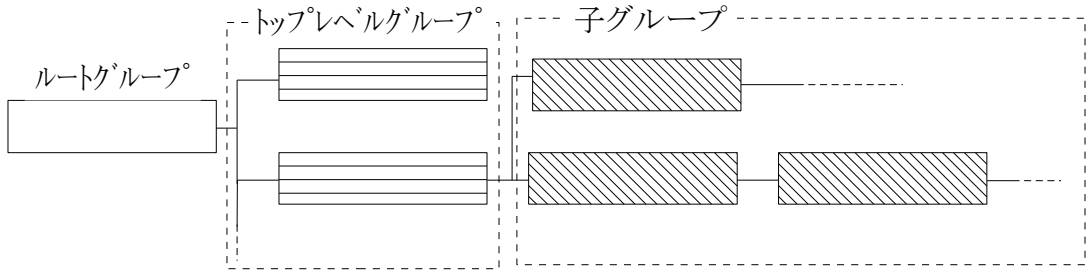


図 9　グループの構成

表-4　グループの種類と名前規則

グループ種類	階層	名前規則	新規作成
ルートグループ	0	::	不可
トップレベルグループ	1	::グループ名	可
子グループ	2～n	::トップレベルグループ名:子グループ名……:グループ名	可

2.5.4 利用者の管理（ユーザリスト）

システム利用者の情報の閲覧は，ログインしている利用者であれば，コンテンツの「ユーザリスト」から誰でも見ることができる（図 10）．TOP ページの名前に張られたリンクから，図 11 のような画面によって各自の詳細な情報を閲覧できる．

ユーザの新規登録および修正や削除は管理者メニューの「ユーザ管理」から管理者のみが行えるようになっている．利用者の，「管理者」，「ユーザ」，「ゲスト」という権限の指定は，管理者のみが設定できる（図 12）．

ただし，利用者自身の情報（アカウント情報）は，ユーザであれば一部の情報（ハンドル名，パスワード，鍵情報，所属グループ，備考）の変更が可能となっている．アカウントメニューの「ユーザー情報の変更」から行える．

各表示画面で太枠で囲まれている箇所は，管理者[*7]のみ表示される．

図 10　ユーザリスト(TOP ページ)

図 11　ユーザリスト(詳細ページ)

*7 ここでの「管理者」は，「グループ管理者」ではなく，利用者の分類(2.4.1)での「管理者」

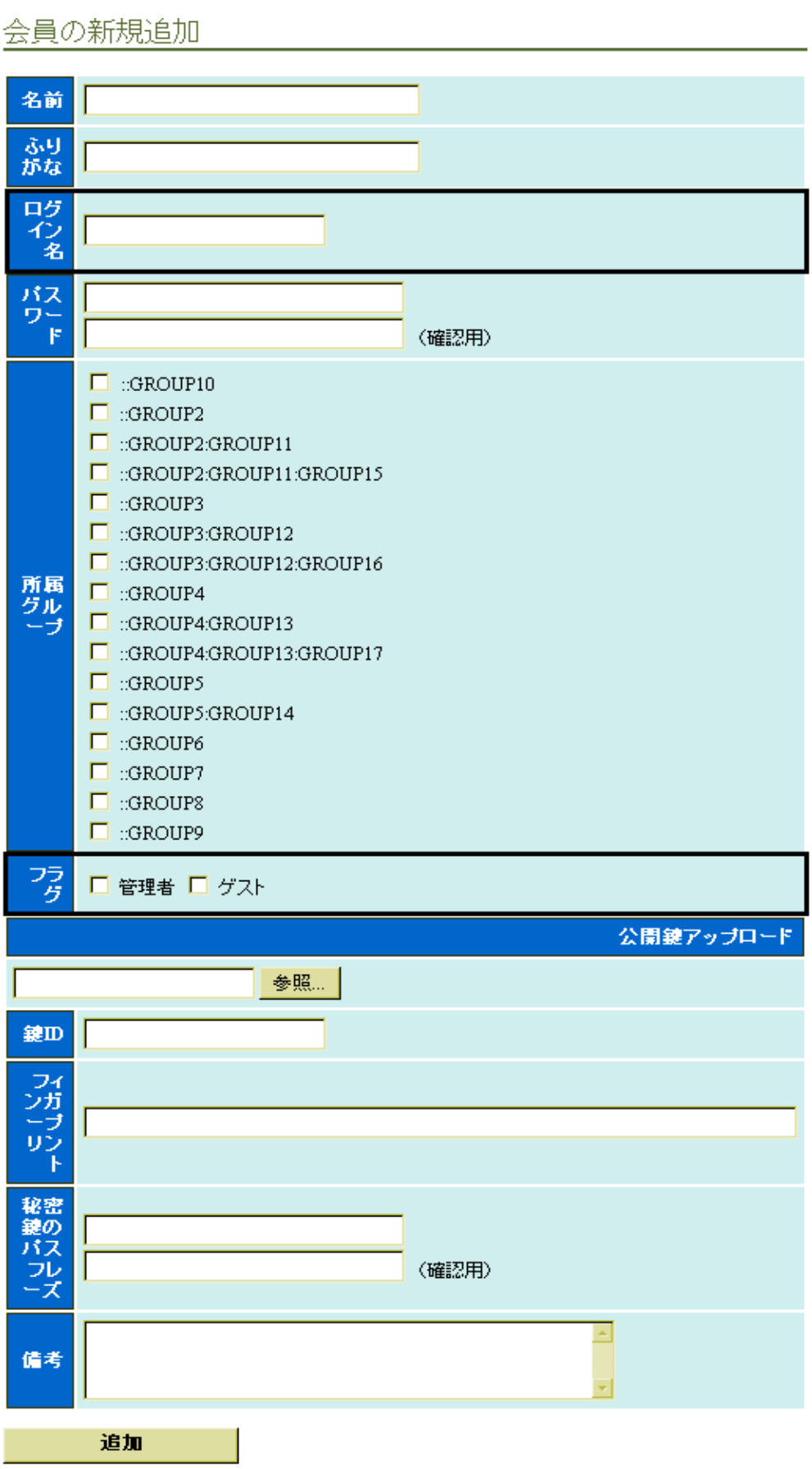


図 12　ユーザリスト(登録・修正フォーム)

2.5.5 グループの管理(グループリスト)

グループの閲覧はログインしている利用者であれば, コンテンツの「ログインリスト」から誰でも閲覧できる(図 13).

グループの新規作成は, ユーザと管理者であれば誰でも作成可能となっている. 新規作成できるグループは, トップレベルグループと子グループの二つがあり, どちらを作成するか選択できる. 子グループは直近の親を指定することで作成することができる. また, 子グループを親とするグループの作成をすることも可能である(図 15).

また, グループの管理は作成したユーザと管理者が行うことができる. この二者を「グループ管理者」と呼ぶ. グループ管理者は, グループの編集, 削除, 参加者の変更(図 17)を行える(図 14). そして管理者は, 上記に加えてグループ作成者の変更項目が存在する(図 16).

各グループの詳細な情報は, TOP ページのグループ名のリンクから閲覧できる(図 18).

各表示画面で点線の太枠で囲まれた部分はユーザと管理者, 太枠で囲まれている箇所は, 管理者[*8]のみ表示される.

図 13 グループリスト(TOP ページ)

*8 ここでの「管理者」は, 「グループ管理者」ではなく, 利用者の分類(2.4.1)での「管理者」

図 14　グループリスト(グループ管理者ページ　*9)

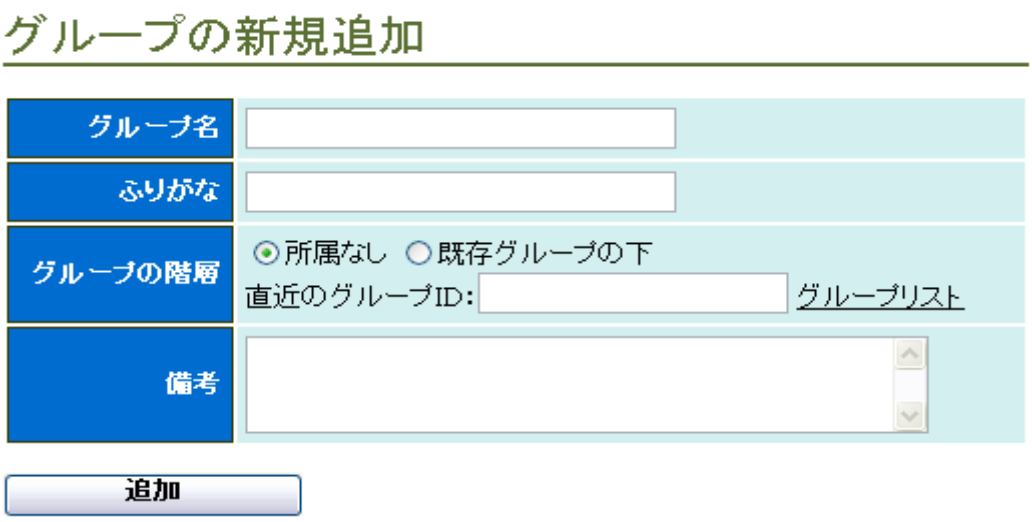


図 15　グループリスト(作成フォーム)

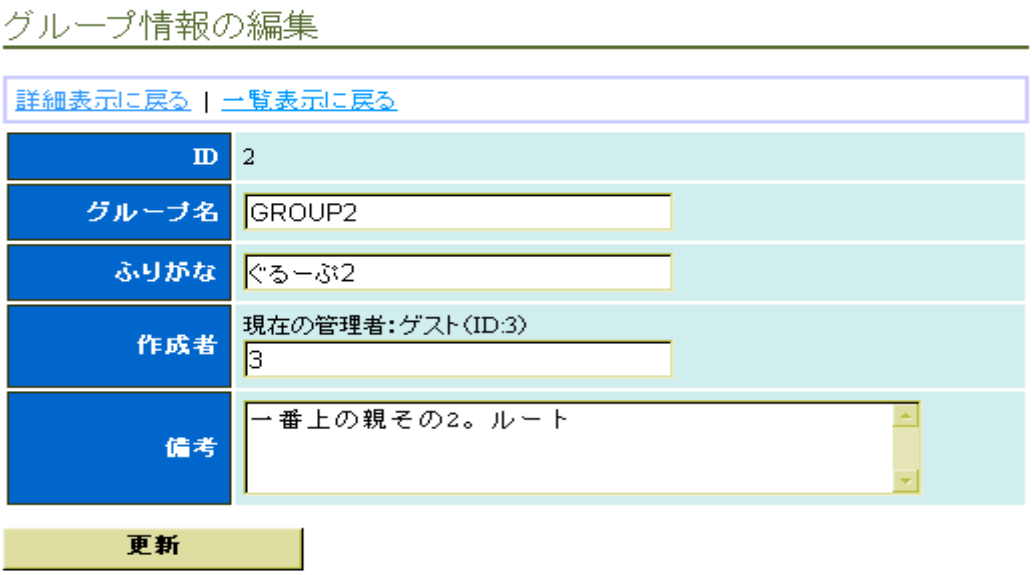


図 16　グループリスト(編集フォーム)

*9 グループ管理者ページ:アカウントメニューの「管理グループの変更」と管理者メニューの「グループ管理」

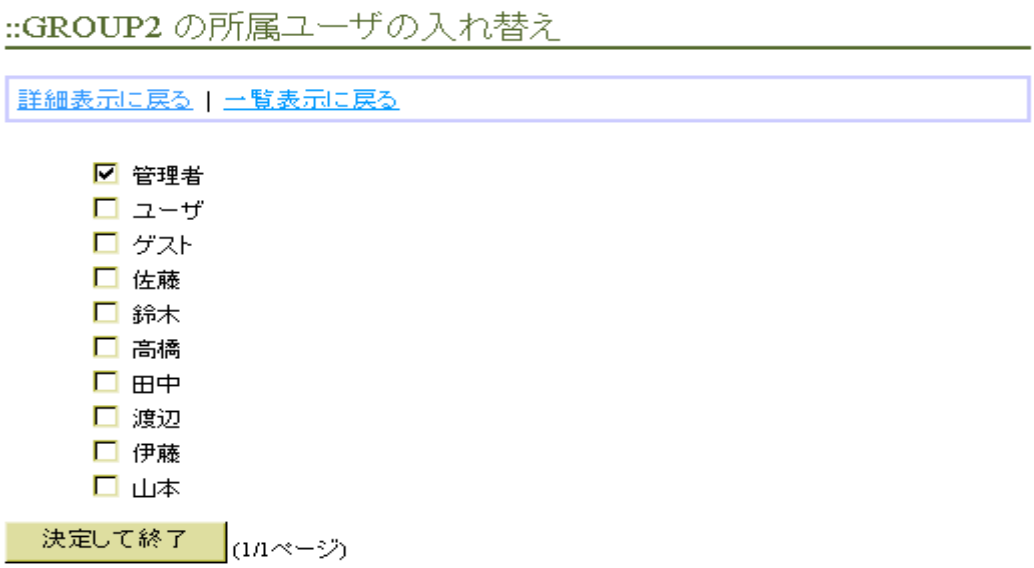


図 17　グループリスト(グループユーザの変更)

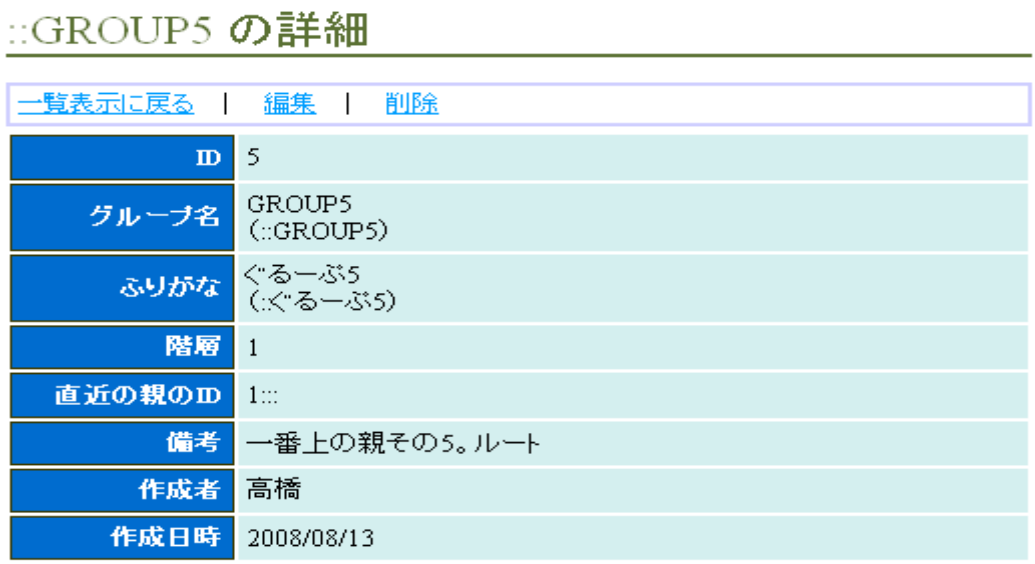


図 18 グループリスト(詳細ページ)

2.5.6　記事の投稿・管理(ブログリスト)

記事の閲覧は，コンテンツの「ブログリスト」からできる．コンテンツへのアクセスは誰でも出来るが，実際の閲覧については記事ごとに指定された閲覧範囲によって制限されている．つまり，公開範囲によってはシステムへのログインをしていなくても記事を閲覧することが可能となっている(公開範囲の詳細は次節「2.6 記事毎の公開範囲の指定」)．閲覧者が公開範囲に含まれていない記事は，TOP ページでは記事内容の代わりに「閲覧権限がありません」と表示されるが，タイトルや投稿日時や記事の検証は行うことができる(図 19)．

記事の新規作成は，ログインしているユーザと管理者であれば「ブログリスト」コンテンツのサイドバーより行える．投稿画面は図 20 のようなものとなる．

タイトル，本文，カテゴリ，公開範囲の指定を行い，記事を記入した後，「プレビュー」ボタンを押すと，投稿はされず，図 21 のような署名前のテキストのチェックおよび公開範囲の確認ができる．なお，カテゴリと公開範囲は，記事の投稿後にも修正可能である（図 22）．

最後に投稿ボタンをクリックすることで，自動的にシステムにより二重署名が行われ文書サーバに保管される．この二重署名の処理についての詳細は「2.3.2 検印モデルによる実装」で述べた通りである．その二重署名も含めた全文を表示したページが図 23 である．

また，各記事にはファイルを添付することができる．このファイルについても記事と同様に二重署名が行われる．添付ファイルのダウンロードは，二重に署名されたままのファイルと，システムが自動的に復号処理を行ったファイルの両方を，記事の詳細ページよりダウンロードが可能となっている（図 24）．

なお，現在，投稿後の記事の修正・再編集は認めていない．記事の削除については，管理者のみが行える．また，添付データの削除は投稿者も自身の記事のアカウントメニューの「ブログ記事の閲覧範囲の変更／添付ファイルの削除 」から行えるが，削除理由を明記した上で削除可能としている（図 25）．また添付データが存在していた記録が残るようにしている（図 26）．これは，研究記録の非改竄性を考慮してのことである．このように編集や削除に制限をかけているが，記事などの管理には DBMS を利用しているので，DBMS を直接操作することにより，記事の修正や削除は可能である．

各表示画面で点線の太枠で囲まれた部分はユーザと管理者にのみ表示される．

図 19　ブログリスト（TOP ページ）

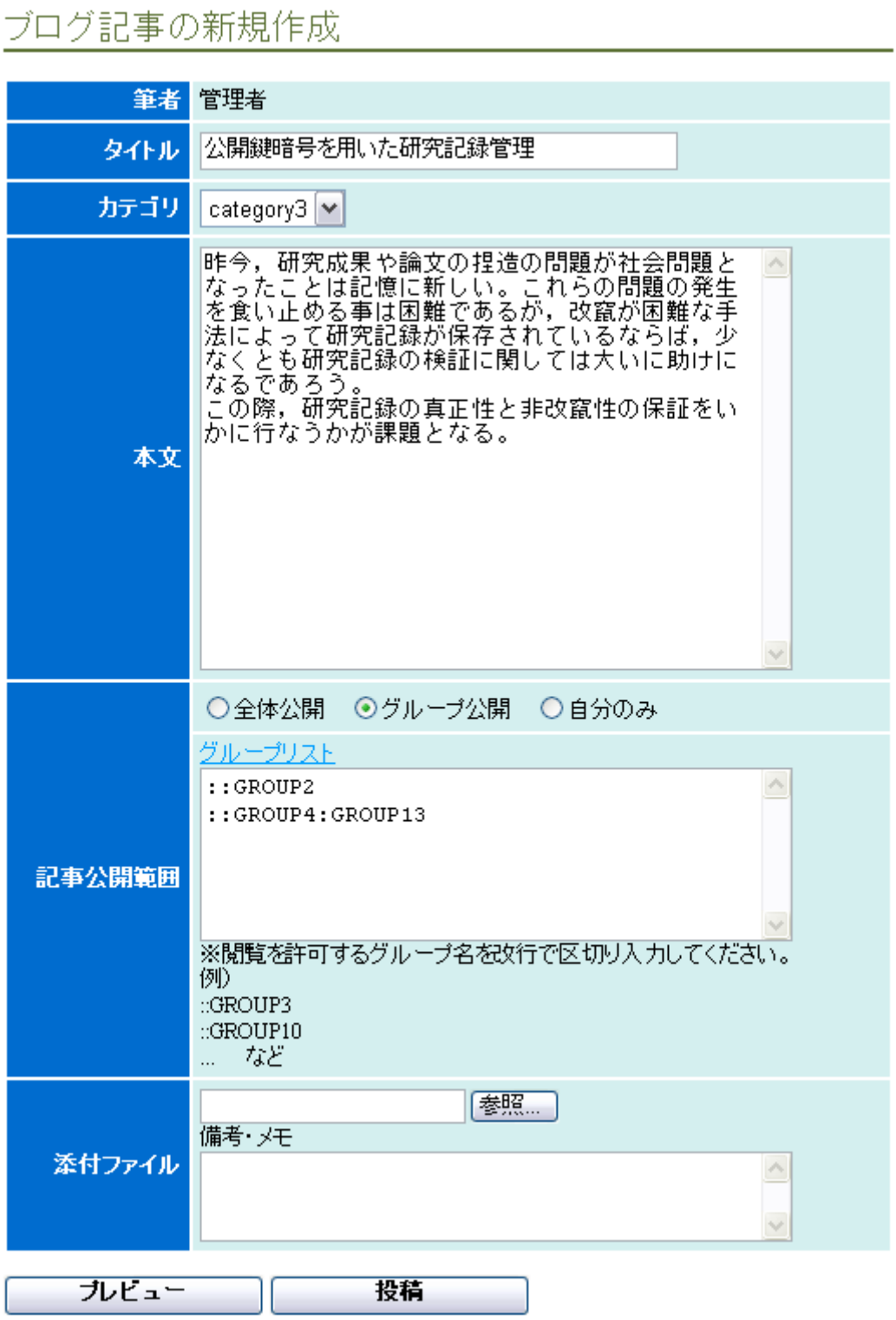


図 20　ブログリスト(投稿フォーム)

タイトル：公開鍵暗号を用いた研究記録管理

カテゴリ: category3

≪本文≫

昨今，研究成果や論文の捏造の問題が社会問題となったことは記憶に新しい。これらの問題の発生を食い止める事は困難であるが，改竄が困難な手法によって研究記録が保存されているならば，少なくとも研究記録の検証に関しては大いに助けになるであろう。
この際，研究記録の真正性と非改竄性の保証をいかに行なうかが課題となる。

≪公開範囲≫

::GROUP2:GROUP11
::GROUP2:GROUP11:GROUP15
::GROUP2
::GROUP4:GROUP13:GROUP17
::GROUP4:GROUP13

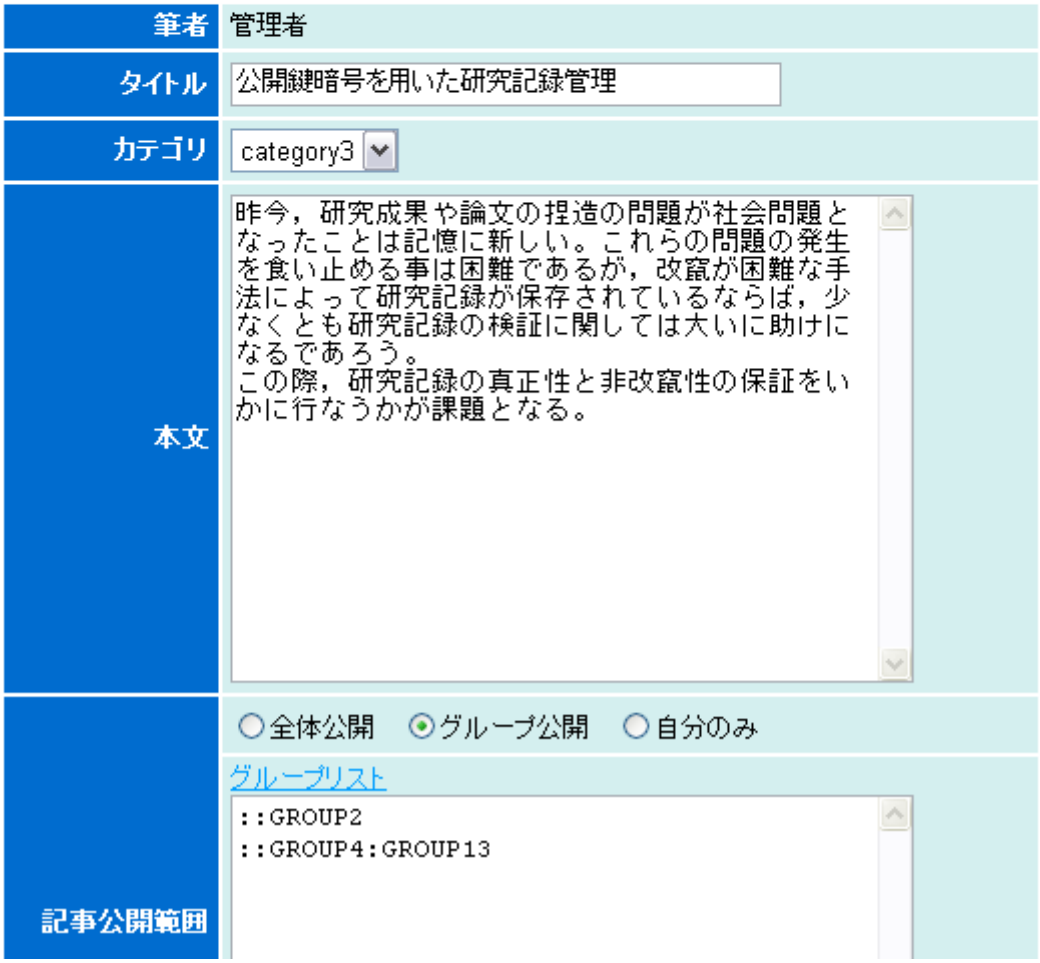


図 21　ブログリスト(プレビュー表示)

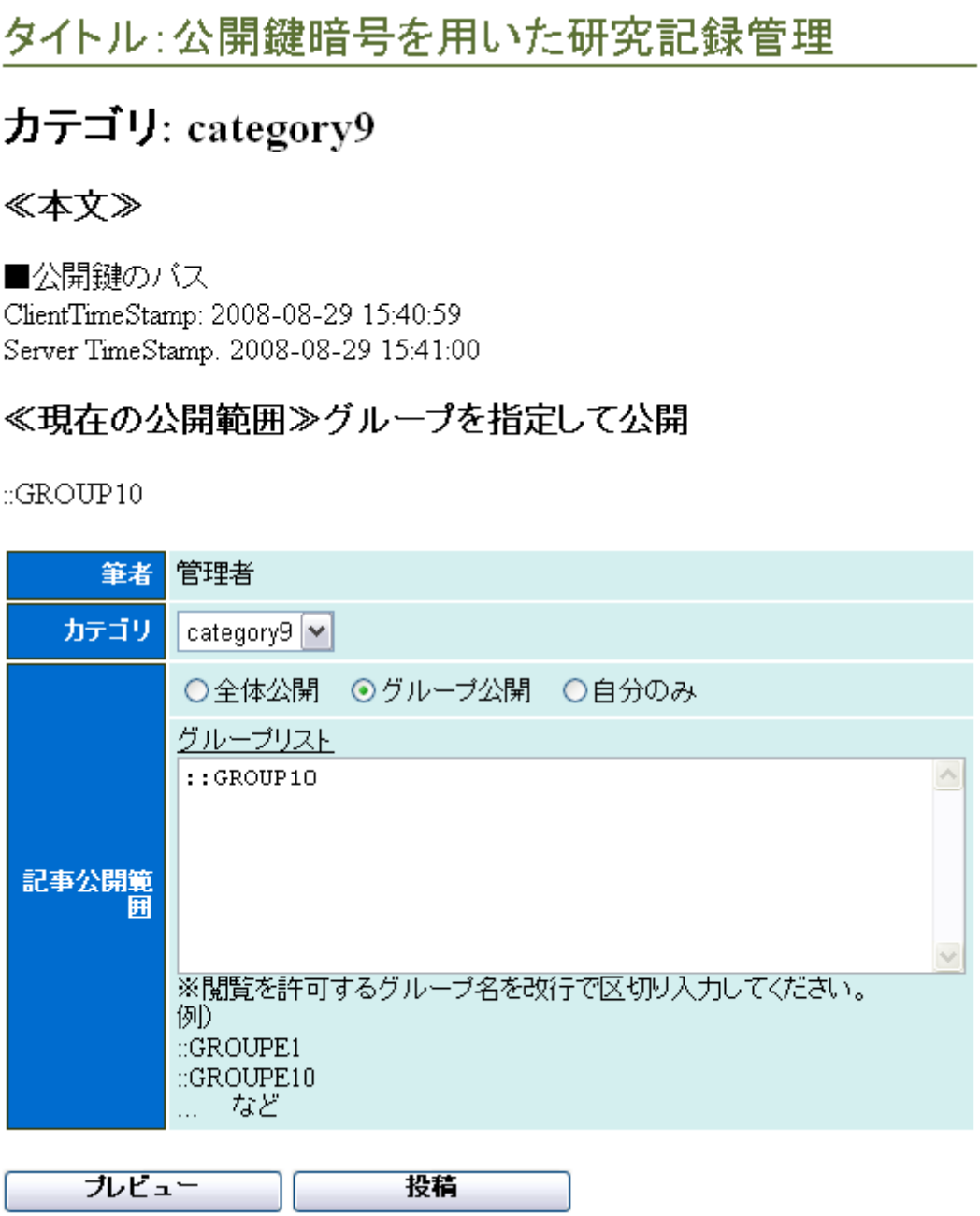


図 22　ブログリスト（公開範囲とカテゴリの修正）

公開鍵暗号を用いた研究記録管理

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

- -----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

昨今，研究成果や論文の捏造の問題が社会問題となったことは記憶に新しい。これらの問題の発生を食い止める事は困難であるが，改竄が困難な手法によって研究記録が保存されているならば，少なくとも研究記録の検証に関しては大いに助けになるであろう。
この際，研究記録の真正性と非改竄性の保証をいかに行なうかが課題となる。

ClientTimeStamp: 2008-07-16 15:02:05
- -----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.9 (MingW32)

iEYEARECAAYFAkh9jt0ACgkQFfuywYqu77h vwCg4TbT0j6VmA81itgNc918BVcj
57wAoNdd4g3dX7Wi/pKZpEcFoiZdGI/P
=m2iZ
- -----END PGP SIGNATURE-----

Server TimeStamp. 2008-07-16 15:02:05
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.9 (MingW32)

iEYEARECAAYFAkh9jt0ACgkQNkcjG4aWrL3U1ACeITAQY/cmvgo1/oVYuhRN7h9
4moAoIErpmkVJm9DRuTybs9EbndH217D
=oqwK
-----END PGP SIGNATURE-----

公開範囲

::GROUP2
::GROUP2:GROUP11
::GROUP4:GROUP13
::GROUP2:GROUP11:GROUP15
::GROUP4:GROUP13:GROUP17

2008-07-16 15:02:05 | 管理者 | 簡易検証 | 通常検証 | コメント(0) | category3

コメントを書く

図 23　ブログリスト（詳細ページ）

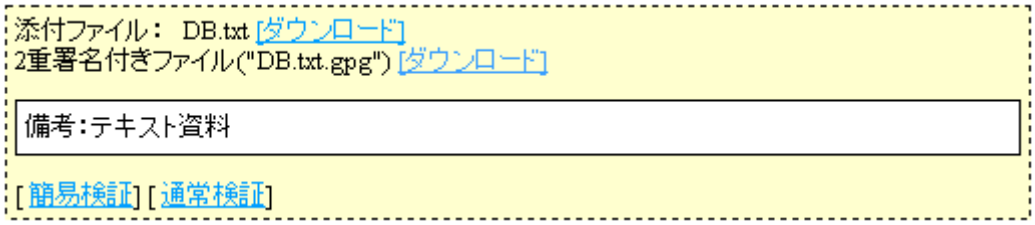


図 24　ブログリスト(添付ファイルのダウンロード)

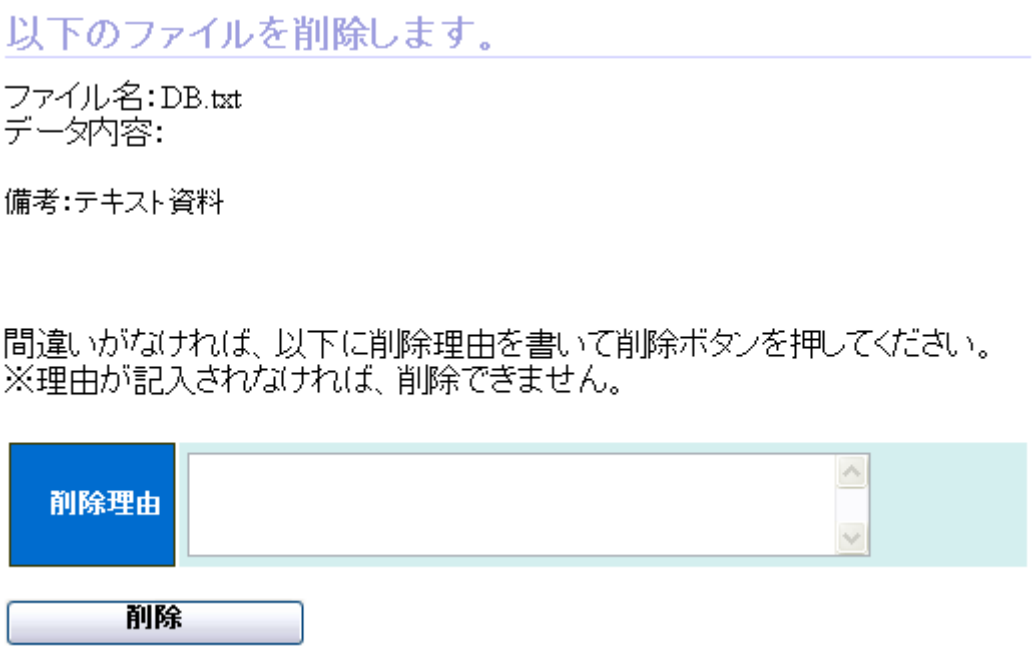


図 25　ブログリスト(添付ファイルの削除フォーム)

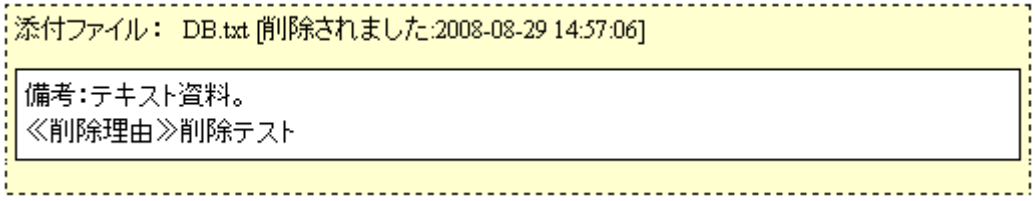


図 26　ブログリスト(添付ファイルの削除後の表示)

2.5.7　アカウントメニュー

「アカウントメニュー」は「ユーザ」および「ゲスト」がアクセスできるコンテンツで，ログインしている利用者自身の情報や投稿データの編集，削除が行えるコンテンツである(図 27)．ただし，「ゲスト」は自身のアカウントデータが閲覧できるだけで，編集や削除作業は一切出来ない．

メニュー項目については表 3 を参照．

2.5.8　管理者メニュー

「管理者メニュー」は「管理者」のみがアクセスできるコンテンツで，システム全体の各データの編集，削除が行えるコンテンツである（図 28）．

ただし，ブログ記事の本文データの編集は，管理者であっても出来ない．

メニュー項目については「2.5.2 利用者の分類」の表 3 に示す．

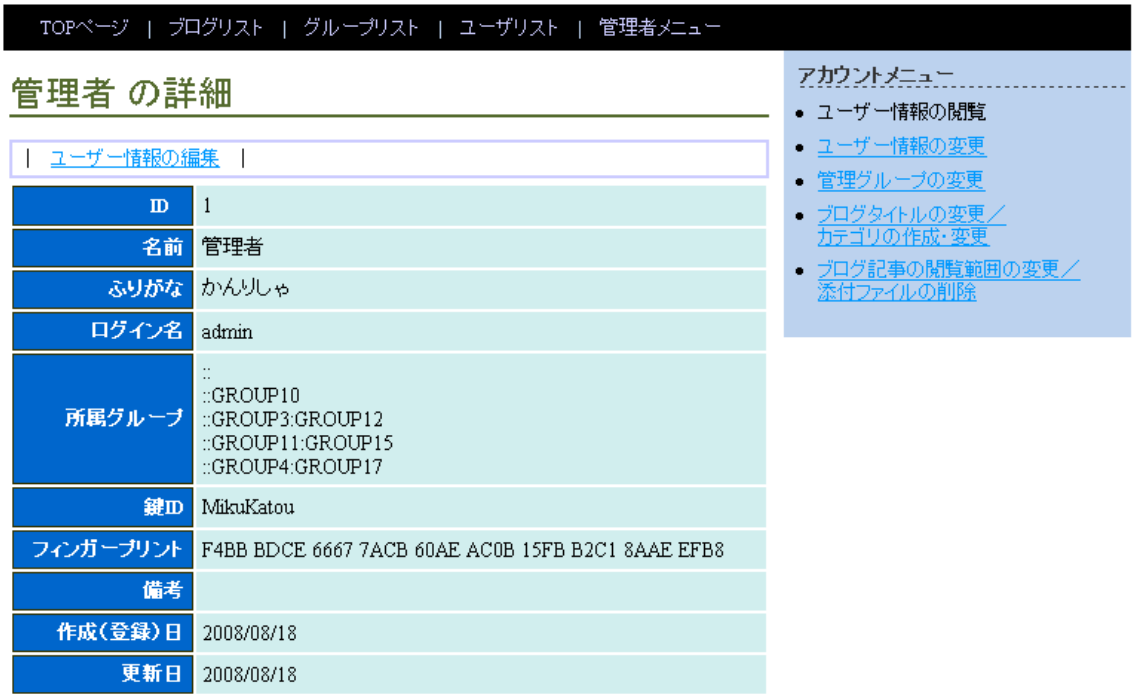


図 27　アカウントメニュー（TOP ページ）

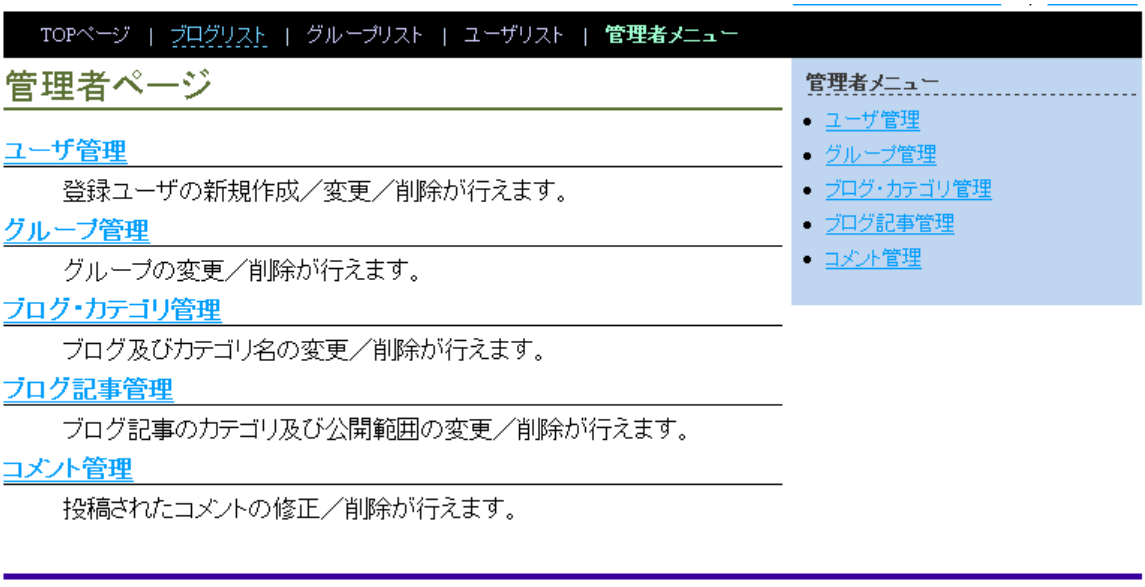


図 28　管理者メニュー（TOP ページ）

2.6 記事毎の公開範囲の指定

前述したように本システムでは記事単位で公開範囲を指定することが出来る.

公開範囲は大きく「全体公開」,「指定グループ」,「自身のみ」の 3 種類に分けて指定することができる.

「全体公開」は, システムの認証(ログイン)を受けなくても閲覧可能な公開範囲である.

「自身のみ」は, 記事を書いたユーザ自身のみが閲覧可能な公開範囲である.

「指定グループ」による公開範囲は, グループを単位として公開範囲を指定する. この時複数のグループを指定することができる. 記述例として「2.5.6 記事の投稿・管理(ブログリスト)」図 21 が挙げられる.

図 21 のように, 改行を区切りとしてテキストエリアにグループ名を入力することで, 公開範囲に含めるグループを指定していく. この時, 指定されたグループを上位に持つ子グループも自動的にその公開範囲に含められる(図 29).

なお, 記事の投稿後にも公開範囲の変更は可能としている. 閲覧範囲は, 記事を書いたユーザ, あるいは管理者のみが変更可能である。「2.5.6 記事の投稿・管理(ブログリスト)」の図 22 に示す.

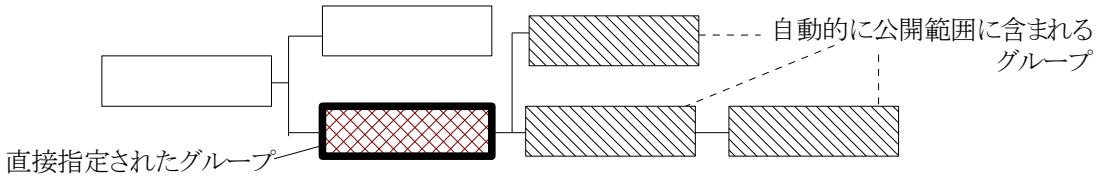


図 29 範囲指定例

3 類似システムとの比較検証

3.1 記事毎の公開範囲の指定

・公開範囲指定のインタフェース

公開範囲を指定する際のインタフェースを Enzin[6], ACS[7], 本システムで比較すると表 5 のようになる.

Enzin は直感的に操作できるインターフェースであり, ACS もクリック操作のみでグループなどの指定が可能であることに対し, 本稿のシステムは若干分かりにくく手間がかかる. これは本稿ではグループがユーザ全体で共通であるため, グループ数が多くなることが予想されるためである. そのため, グループの一覧を表示させるのではなく, テキストエリアで必要最小限の数を文字で入力することで指定するようにした(詳細は「2.5 記事毎の公開範囲の

指定」).

表5 公開範囲の指定方法と変更

	設定方法方法	変更
Enzin	ユーザやグループなどの意味を持たせたアイコンのドラッグ&ドロップによる指定	可
ACS	チェックボックスによる指定	不可
本システム	テキストエリアへのグループ名の記入	可

・公開範囲の設定段階

同様に, Enzin, ACS, 本システムにおける公開範囲の設定の柔軟性を比較すると, 表 6 のようになる.

表 6 に示されるとおり, 投稿後の公開範囲の設定の変更の可否はあるが, いずれもその公開段階での自由度はほぼ同じといえる.

Enzin および ACS は, 一般的なコミュニケーションツールである SNS を基盤とし, 一般のコミュニケーションの支援を目的として作られている. 対して, 本システムではシステムそのものは同種である Blog を基本としたシステムであるが, 研究支援として利用することを前提としている. そのため, ある程度記事を公開する対象や内容・目的が限定され, グループの数は多くなったとしても, グループ間になんらかの関係が存在することが予想される. そこで, 特に実在する組織の階層構造を反映することを想定し, グループを階層的に定義可能とし, グループ間の関連が分かりやすいグループ設定と公開範囲の指定を可能とした.

表6 公開範囲の段階と評価

システム名	公開範囲の段階	グループ管理
Enzin	「自分のみ」,「メンバ」,「グループ(2人以上のメンバの集合)」,「インターネット全体」を自由に組み合わせて実現。	個人
ACS	「自分のみ」,「グループ(一人以上のメンバからなる)」,「一般公開」,「パブリックリリース(一般公開よりもさらに積極的に情報発信)」のいずれかを選択	個人
本システム	「自分のみ」,「グループ(一人以上のユーザからなる)」,「全体公開」, グループの階層化が可能	全体

3.2 比較:真正性と非改ざん性の保持

現在, 真正性と非改竄性を保障する既存の技術として, 電子文書署名サービス(タイムスタンプサービス)がある.

電子文書署名サービスは, 一般的に作成した電子データのハッシュ値をサービスを提供するサーバに送り, 受信したサーバがその受信日時を記録して署名を行うものである.

本項では本人が書いたという真正性の確保も含めた比較のため, サーバには電子文書などと共に, 作成者本人の署名も含めて送ったと想定し, 比較検証を行う.

・非改竄性の保証の強度と検証効率

電子文書署名サービスは，文書作成者とは関連のない第三者である企業が，作成された文書から得られるハッシュ値に基づいて非改竄性の保証を行っており，サービスによっては新聞でスーパーハッシュ値を公開しているものもある．認証を受けた企業がサービスを提供するため，サーバへの進入や，文書作成者と企業の秘密鍵を入手することは非常に困難であり，強度は非常に高いと予想される．

また検証は，必要とされた時にサービス提供者に依頼することでその回答を得られる．

一方本システムも，作成者と，作成者とは別の第三者に対して署名によってその非改竄性の保証を行っているが，現段階では現実的に完全な第三者による検印サーバの管理の依頼は難しいと考えられるため，電子文書署名サービスに対して強度は劣る．ただし，ラボノートの運用，証拠能力から，十分実用に足る能力を持っていると考える．強度を上げるためには，ユーザおよびシステムの秘密鍵の管理法の改善が課題となる．

検証は，基本的に記事の閲覧ページから1クリックで行うことが可能となっている．

・導入・利用のコスト

電子文書署名サービスは，e-文書法(2005)制定以需要が高まり各社で手軽なサービスが提供されるようになっている．なかには，契約会社がデータの受け取りからタイムスタンプまでの作業をシステム化して月額で契約するといったサービスもあるが大掛かりである．また基本的に一文書あたり，あるいは一回当たり10～300円程度の課金であることが多く手軽に利用できるとは言い難い．

本システムも導入までに技術的・金銭的なコストを必要とするが，全て企業に一任するよりは金銭的なコストは低く，また一度設置した後は，署名から検証までの煩雑な作業は全てシステム側がバックグラウンドで行える．また現在広く利用されているブログのシステムを基本としているため，操作方法は通常のブログの記事投稿の作業と同様で済み，利用者は署名のための知識を意識せず一定の真正性が保障された電子記録の保存作業が行える．

4 今後の課題

4.1 セキュリティに関して

本システムの大きな課題として，まずセキュリティの問題がある．

真正性の保証については，ユーザの秘密鍵を IC カードや USB メモリなどの外部記憶メディアに保管し，記事を投稿するときにのみにそれを参照することで解決できよう．また，本システムにおいて，成りすましが行われる可能性もある．しかし，ユーザの秘密鍵を IC カードや USB メモリなどの外部記憶メディアに保管することにより，成りすまして記事を投稿することはできなくなる．

非改竄性の証明の強度を高めるためには，リンキングやヒステリシス署名，履歴交差などの技術の導入の検討も必要であろう[3，10]．関連して，現在 GnuPG に依存したシステムである為，多様な署名方式に対応可能とすることによっても利便性と共に非改竄性の向上が期待できる[11]．

また電子署名の長期利用に関しては，秘密鍵の危殆化などの問題もあり，評価・検討が必要である[12，13]．同様に一定期間ごとの検証記録を保存することによって，どの時点まで真正性が保持されていたかを証拠として残していくことも今後必要となってくるだろう[14]．

現状では，検印サーバでの署名のために，記事やデータそのものを通信している．そのため情報漏えいの危険性がある．現在は，SSL での通信を行うことで対処しているが，情報漏えいの対策として十分とは言いがたい．記事やデータのハッシュ値のみの通信に限ることも含めて今後検討を要する．

また，公開鍵の正当性の確保のため，信用の輪（Web of Trust）も含めた PKI の利用の検討も必要であろう．

4.2 ユーザインタフェースに関して

柔軟な公開範囲の設定については，3.1 節でも取り上げたとおり概ね既存のものに近い自由度となっている．しかし，ログインしている利用者に対しての公開範囲の指定は現状十分に出来るが，ログインしていない一般に対しての公開範囲は全体公開しか存在しない．今後システム利用者と一般を区別せず，任意の記事に対して特定のパスワードを入力することで閲覧を可能とする公開範囲というものも考えられる．

また本システムではグループの階層化を取り入れたが，既存のグループの上位に新たに階層を作成することはできない．木構造の表現を可能とする acts_as_tree という特殊なカラムの命名規約（MagicFieldNames）が Ruby on Rails に存在するので，それを利用したグループの作成も考えられる．また，異なる上位グループを持つグループ同士であっても，そのグループの構成員の間には何らかの関連がある場合も現実にはある．そのような縦方向以外での関連を持たせる実装も含め，今後さらに自由度の高いグループ管理を可能としていくことも課題となる．

交流の支援という観点からは，本システム，または本システムと同種のシステムが複数稼動した場合，ゲストが各ユーザの Blog を閲覧するたびに，個々にログインを要求されるのは煩雑である．そこで，本システムあるいは同種のシステム間において，認証の Delegate 機能も

必要であろう. また RSS の配信やトラックバックなど一般の Blog に存在する機能についても, 閲覧範囲が記事単位で可変であるため難しいが, 今後必要に応じて検討を要する. 同じく一般の Blog システム・サービスによくある, 他システムからのデータのインポート, また逆に他システムへのエクスポート機能の実装も考えられる.

研究支援という観点からは, 記事中での画像や数式や表・グラフ, 化学式への対応も不可欠である. このような問題については, 後藤らによるシステムや[15], Jipsen の仕事[16]が利用/応用可能であろう. また漢字における異体字や国字に対しての対応も検討していきたい.

その他, 記事へのグループ指定を筆頭としたやや煩雑な作業に対して, Ajax 技術なども用いたユーザインタフェースの改善も今後の課題である.

4.3 その他

現在, 文書サーバと検印サーバは一対一で用意することを前提としているが, 今後複数の文書サーバに対し, 共通の検印サーバを利用することも考えられる. その際に, 現在のスクリプトでは別々の秘密鍵で署名を行わせることや, また署名部のコピーを異なるサーバ別にフォルダを分けて保存することは出来ないため, 改良の必要がある.

記録の保存ということを考えると, 現在のデータベースをベースとした記事管理はサーバに負担をかけてしまうと考えられる. そのため, DBMS は必要だが, それに全て埋め込むのではない, 記事やユーザ情報の保存・管理方法が課題として上げられる.

また, 現在基盤となるシステムを Ruby on Rails1.2.6 で作成しているが, より柔軟な機能作成や今後蓄積されるデータの柔軟な活用を行っていくことを考慮すると, HTML だけではなく XML や JSON などの複数のフォーマットでの出力も可能な Ruby on Rails2.0 への移行が望ましいと考えられる.

そして, ユーザインタフェースとも関連するが, 現在グループはシステムの利用者全体で共有し, ログインしている利用者であれば誰でも閲覧が可能で, ユーザであれば任意のグループに所属できるようになっている. しかし, 現実にはグループの存在やそこに所属するメンバーを秘匿しておきたい場合も考えられる. それに関連して, ユーザの情報も現時点ではログインしている利用者であれば, 誰でも閲覧が可能となっている. 昨今個人情報の保護が問題とされる機会も多く, 一般の SNS でも特定の知人のみにしか自分のユーザ情報を開示しない機能を持つものは多い[*10]. それらの問題も含め, 今後情報の開示および管理の権限を誰にどの程度まで認めるのか検討を要する.

*10mixi（http://mixi.jp/）,CURURU（http://www.cururu.jp/）など

5 終わりに

本研究では，研究記録などの管理を主目的とし，公開範囲を柔軟に制御可能かつ，記録の真正性および非改竄性を可能な限り保証し，検証の機能を持つシステムである arχves の構築を試みた．結果として，先に上げたとおりセキュリティやユーザインタフェース，またそのほかの面でも多くの課題はあるものの，一定の機能を持つシステムを実現できた．また，本システムにより，安価に，DVCS でも TAP でもない，電子書名つき電子文書管理の第三のあり方を実現できた．

今後は，課題でも取り上げた本システムの問題点の対応や改良，そして蓄積されるデータの知的処理に関しても研究を進めたい．

参考文献

[1]原田　篤史, 西垣　正勝, 曽我　正和, 田窪　昭夫,「ライトワンス文書管理システム」, 情報処理学会論文誌 Vol.44, no.8, pp.2093-2105, 2003

[2]陳　明強, 吉川　正俊,「時刻認証付き XML 文書のデータベースによる管理について」, 電子情報通信学会第 16 回データ工学ワークシップ（DEWS2005）, 5A-i10, 2005

[3]宇根　正志, 松本　勉,「可用性および安全性の観点からみた各タイムスタンプ方式間の関係」, 情報処理学会論文誌, vol.43, no.8, pp.2644-2658, 2002

[4]Toshiyuki Masui, Satoru Takabayashi, "Instant Group Communication with QuickML", Proc. ACM Conference on Supporting Group Work(Group '03), pp268-273, 2003

[5]江渡　浩一郎, 高林　哲, 増井 俊之,「quikWeb:メーリングリストと Wiki を統合したコミュニケーション・システム」, 情報処理学会研究報告, 2004-HI-111, pp. 5-11, 2004.

[6] 永田周一, 安村通晃,「Enzin:情報の公開範囲を手軽に変更できるコミュニケーションツール」, 情報処理学会論文誌 Vol.48, no.3, pp.1134-1143, 2007

[7]高井一輝, 河口信夫,「ACS:多様な人間関係を表現可能なソーシャルネットワーキングシステム」, 情報処理学会論文誌, vol. 48, no. 7, pp. 2328-2339, 2007.

[8]高田　良宏, 笠原　禎也, 毛利　信浩, 松平　拓也,「多様なアクセス制限に対応した自然科学データベースシステムの開発」, 学術情報処理研究, no.11, pp.50-59, 2007

[9]C.Adams, P.Dain, D.Pinkas, R.Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol(RFC3161)", ftp://ftp.rfc-editor.org/in-notes/rfc3161.txt

[10]洲崎　誠一, 松本　勉,「電子署名アリバイ実現機構―ヒステリシス署名と履歴交差」, 情報処理学会論文誌, vol.43, no.8, pp2381-2393, 2002

[11]山田　竜也, 宮地　充子, 双紙　正和,「オープンネットワークにおける安全な暗号方式の更新に関する考察」, 情報処理学会論文誌　Vol.4, No.8, pp 2102-2109, 2000

[12]宮崎　邦彦, 吉浦　裕, 岩村　充, 松本　勉, 佐々木　良一,「第三者機関への依存度に基づく長期利用向け電子署名技術評価手法の提案」, 情報処理学会論文, vol.44, no.8, pp1955-1969, 2003

[13]小森　旭, 花岡　悟一郎, 松浦　幹太, 須藤　修,「署名鍵漏洩問題における電子証拠生成技術について」, 電子情報通信学会「暗号と情報セキュリティシンポジウム」予行集, pp.983-988, 2003

[14] 電子商取引推進協議会, 認証・公証 WG,「電子署名文書長期保存に関するガイドライン」, 2002

[15]後藤　洋信, 坂本　雅洋, 江見　圭司,「数式表示可能なウェブ上でのコミュニケーションシステムの構築」, 情報処理学会研究報告 2008-CE-94, pp1-8, 2008.

[16]Peter Jipsen, "ASCII MathML", http://www1.chapman.edu/~jipsen/asciimath.html

[17]黒田　努, 佐藤　和人　共著, 株式会社オイアクス　監修,「基礎 Ruby on Rails」, インプレスジャパン

付録 A　Ruby on Rails の導入方法メモ

1.インストール
(1) Ruby のインストーラを入手する.
　http://rubyinstaller.rubyforge.org（RubyInstaller：One-Click Ruby Installer for Windows）
ここで ruby186-26.exe（数字部分は任意のバージョン）をダウンロード.

(2)その後ダウンロードしたファイルをクリックし「実行」⇒「全てのコンポーネント」を選択して，インストールされるのを待つ.
このとき，Gem も一緒にインストールされているはずなので，コマンドプロンプトで"gem -v"を入力し数字（バージョン）が出力されることを確認する.

(3)Rails のインストール
Gem がインストールされていれば，コマンドプロンプトで下記のコマンドで最新の rails がインストールされる.
"gem install rails --include-dependencies"

＊特定のバージョンを指定してインストールする場合は"-v"または"--version"を付加して行う.
コマンド例：
gem install rails --version [バージョン] --include-dependencies
gem install rails -y -v [バージョン] --include-dependencies
(-v，--version：バージョンの指定，--include-dependencies：依存するライブラリも一緒にインストール，-y：インストール時の確認に全て"Yes"で)

2.Rails プロジェクトの作成メモ
・バージョンを指定してプロジェクトを作成
rails [プロジェクト名] _[バージョン]_

・Rails2.0 以降で DB を MySQL に指定して作成する
rails [プロジェクト名] -d mysql
（"-d"は"--database"でも可）

・利用しているプラグイン
・ssl_required[1]

・プラグインインストール
ruby script/plugin install [プラグインの URL]

3.Gem のコマンド
・gem の環境を確認
gem env

・インストールされてる rails のバージョンの確認
gem list rails

1　http と https の切り替え制御（http://svn.rubyonrails.org/rails/plugins/ssl_requirement/）

標準でインストールされていないが，利用しているライブラリ
・win32-open3 [*12]
・mongrel [*13]

インストール方法：
Gem がインストール済みであれば rails のインストール同様コマンドプロンプトで"gem install [ライブラリ名]"

付録 B Apatch の設定（CA ルート証明書作成含む）

○Mongrel＋Apatch（＋SSL）の設定
1.Apatch2.2 以降をインストールする

2.Apache の設定ファイル（httpd.conf）の設定を行う
1)必要なモジュールをロードするために，以下のコマンドのコメントアウトをはずす
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule rewrite_module modules/mod_rewrite.so
LoadModule ssl_module modules/mod_ssl.so
Include conf/extra/httpd-ssl.conf

2)バーチャルホストの設定
<VirtualHost *:80>
ServerName [プロジェクト名]
DocumentRoot [アプリケーションの Public ディレクトリ]
ErrorLog [エラーログの出力場所]

<Directory "[プロジェクトのパス]\public">
Options FollowSymLinks
AllowOverride all
Order deny, allow
Deny from all
Allow from 127.0.0.1 192.168.0.0
</Directory>

3)フォアワードプロシキの Off
ProxyRequests Off

4)静的コンテンツを Apache 側で処理するように記述
ProxyPass /stylesheets/ !
ProxyPass /images/ !
ProxyPass /javascripts/ !

*12 標準入力・標準出力・標準エラー出力にパイプをつなぐ/GPG の出力取得に利用（http://rubyforge.org/frs/?group_id=85）
*13 Web サーバ/Ruby on Rails の稼動に利用（http://mongrel.rubyforge.org/）

ProxyPass /*.html !
ProxyPass /favicon.ico !

5)静的コンテンツのパターンにマッチしなかったリクエストを Mongrel に転送するように指示する
ProxyPass（転送の基点）（転送先アプリケーションサーバの URL）

6)バーチャルホストの設定を閉じる
</VirtualHost>

3.Apache の SSL 設定ファイル（httpd-ssl.conf）の設定を行う
1)SSL の設定を以下のように書き換える
<VirtualHost サーバの IP アドレス:443>
SSLCertificateFile "※デジタル証明書パス"
SSLCertificateKeyFile "※秘密鍵パス"
※デジタル証明書と秘密鍵の作成方法は後述

2)バーチャルホストの設定
2.の 2)～5)の記述を同様に追加する.

4.Mongrel を起動して動作確認を行う.
＞mongrel_rails start -e development -p 3000
（Mongrel がインストールされている状態であれば”ruby script/server”でも上記と同様に実行可能）

○デジタル証明書と秘密鍵の作成方法
1. Web サーバの秘密鍵の生成（鍵生成ディレクトリに移動)
[/Apache の conf 以下:C:\Program Files\Apache Software Foundation\Apache2.2\conf\key]
＞openssl genrsa -des3 1024 > server.key
genrsa RSA 形式の秘密鍵の作成.
-des3 des3 方式でファイルを暗号化.
1024 1024 バイトの鍵を生成.

2.Web サーバの公開鍵の作成
＞openssl req -config ..\openSSL.cnf -new -key server.key > server.csr
req： CSR ファイルの作成します.
-new：新規に CSR ファイルを作成.
-key 鍵ファイル名 ： 入力する秘密鍵のファイル名を指定（パスフレーズ付きの鍵の場合は,パスフレーズが必要）
-config openssl の cnf ファイル ： openssl.cnf のパスを指定.

3.デジタル証明書の作成
＞openssl x509 -in server.csr -days 365 -req -signkey server.key > server.crt
x509：X.509 形式のデジタル証明書の作成します.
-in：CSR ファイルパスを指定します.
-days: 日数　証明書の有効期限を指定します.
-req：入力ファイルが CSR ファイルであることを指定します.
-signkey：自己証明書生成時に使用するオプション. 秘密鍵のパスを指定します.

備考:秘密鍵のパスフレーズの解除方法
（server.key⇒server.key_bk（ファイル名は任意）に変更後）
＞openssl rsa -in server.key_bk > server.key
rsa：RSA 方式の鍵作成.
-in：CSR ファイルパスを指定.

○その他メモ
・Apache のルートを Rails のプロジェクトの public フォルダに指定した場合（上記設定）のブラウザから「arχves」の TOP ページへのアクセス方法は、”[ルート URL]/main”となる。
例:http://localhost/main

※[ ]内の文字は全て環境に応じて任意の文字・数値を入れてください。

参考文献
高橋 征義 , 諸橋 恭介,「Rails レシピブック 183 の技」, ソフトバンククリエイティブ, 2008/5/31
[Think IT] 第 7 回:Apache+SSL 環境を構築しよう！
（http://www.thinkit.co.jp/free/article/0706/3/7/）

付録 C　GnuPG のコマンド

書式: gpg [オプション] [ファイル]

署名，検査，暗号化や復号，既定の操作は，入力データに依存

コマンド:

-s, --sign [ファイル]	署名を作成
--clearsign [ファイル]	クリア署名を作成。変換後のファイル名は"『元ファイル名』.asc"
-b, --detach-sign	分離署名を作成
-e, --encrypt	データを暗号化
-c, --symmetric	暗号化には対称暗号法のみを使用
-d, --decrypt	データを復号 (既定)変換後の出力結果は標準出力に出るため、リダイレクト(">『ファイル名』")で指定。
--verify	署名を検証
--list-keys	鍵の一覧
--list-sigs	鍵と署名の一覧
--check-sigs	鍵署名の検査と一覧
--fingerprint	鍵と指紋の一覧
-K, --list-secret-keys	秘密鍵の一覧
--gen-key	新しい鍵対を生成
--delete-keys	公開鍵輪から鍵群を削除
--delete-secret-keys	秘密鍵輪から鍵群を削除
--sign-key	鍵に署名
--lsign-key	鍵へ内部的に署名
--edit-key	鍵への署名や編集
--gen-revoke	失効証明書を生成
--export	鍵を書き出す
--send-keys	鍵サーバーに鍵を書き出す
--recv-keys	鍵サーバーから鍵を読み込む
--search-keys	鍵サーバーの鍵を検索する
--refresh-keys	鍵サーバーから鍵を全部更新する
--import	鍵の読込み/併合
--card-status	カード状態を表示
--card-edit	カードのデータを変更
--change-pin	カードのPINを変更
--update-trustdb	信用データベースを更新
--print-md アルゴリズム [ファイル]	メッセージ要約を表示

オプション:

-a, --armor	ASCII形式の包装を作成
-r, --recipient 名前	「名前」用に暗号化
-u, --local-user	署名や復号にこのユーザーidを使用
-z N	圧縮レベルをNに設定(0は非圧縮)
--textmode	正準テキスト・モードを使用
-o, --output	出力ファイルとして使用
-v, --verbose	冗長
-n, --dry-run	無変更
-i, --interactive	上書き前に確認
--openpgp	厳密なOpenPGPの振舞を採用
--pgp2	PGP 2.x互換のメッセージを生成

Help に載っている以外で利用しているコマンド･オプション　利用例など

コマンド:

--version	ヴァージョンの確認
--enarmor [ファイル]	ファイルをアスキーコード（文字列）に変換（エンコード）する。変換後のファイル名は"『元ファイル名』.asc"
--dearmor [ファイル]	ファイル文字列をデコードする。変換後のファイル名は"『元ファイル名』.gpg"

○利用例
■署名コマンド
gpg -u [鍵ID] --passphrase [パスフレーズ] --clearsign [処理対象ファイルのパス]
■復号コマンド
gpg -u [鍵ID] --passphrase [パスフレーズ] --decrypt [処理対象ファイルのパス] > [出力ファイル名]
■署名の検証コマンド
gpg -u [鍵ID] --passphrase [パスフレーズ] --verify [処理対象ファイルのパス]

付録 D　データベース（MySQL）の構成･テーブルリスト･設定など

【設定】

エンコード：UTF-8

ストレージエンジン：InnoDB

メモ：

- コマンドプロンプトでの DB 確認を行う際，UTF-8 であるため，日本語は文字化けを起こす．コマンドプロンプトでも日本語出力をする際は"chcp 65001”(UTF-8 へのコードチェンジ)を行う．
- 通常の設定では，いくら Rails 側で UTF-8 で設定しても表示エラーが起こる場合がある．その場合は，my.cnf ファイル（MySQL の設定ファイル）の[Client]以下に"default-character-set=utf8”以外にも，"character-set-server=utf8”（文字コードを UTF-8 に指定）と"skip-character-set-client-handshake”（Client による自動文字コードセットをスキップする）を記述しておく．

※テーブルリスト･データベースの構成については次頁以降。

表1

テーブルリスト　08/0714

blog_kijis	型	説明
blog_id*	integer	ブログID（外部キー）
heading	text	記事タイトル
body	text	本文
category_id*	integer	カテゴリID（外部キー）
scope	integer	閲覧許可範囲
created_at	datetime	投稿日時

blogs	型	説明
user_id*	integer	ユーザーID(外部キー）
title	string	タイトル
created_at	datetime	作成日時
updated_at	datetime	更新日時

categories	型	説明
blog_id*	integer	ブログID（外部キー）
title	string	カテゴリ名

comments	型	説明
blog_kiji_id*	integer	ブログ記事ID（外部キー）
author	string	書いた人
body	text	コメント本文
hashed_password	string	パスワード
salt	string	パスワードソルト
created_at	datetime	投稿日時

groups	型	説明
title	text	グループ名
furigana	text	ふりがな
kaisou	integer	階層（親を0、下層ほど加算）
parent	integer	直近の親のID
user_id*	integer	作成者ID（外部キー）
remark	text	備考（グループの説明）
created_at	datetime	作成（登録）日

materials	型	説明
file_name	string	ファイル名
content_type	string	Content type
path	string	ファイルパス
remark	text	備考
created_at	datetime	作成日時
blog_kiji_id*	integer	ブログ記事ID（外部キー）
del	boolean	削除フラグ
deleted_at	datetime	削除日時

trackbacks	型	説明
title	string	記事タイトル
url	string	記事URL
excerpt	text	記事の要約
blog_name	string	ブログサイト名
blog_kiji_id*	integer	（トラックバックを受けた）記事ID
date	datetime	受信日時

users	型	説明
name	string	利用者名
furigana	string	ふりがな
login_name	string	ログイン名
hashed_password	string	パスワード
salt	string	パスワードソルト
administrator	boolean	管理者フラグ
guest	boolean	ゲストフラグ
key_name	string	鍵の名前
passphrase	string	ユーザ鍵のパスフレーズ
finger_print	string	フィンガープリント
remark	text	備考
created_at	datetime	作成（登録）日
updated_at	datetime	更新日

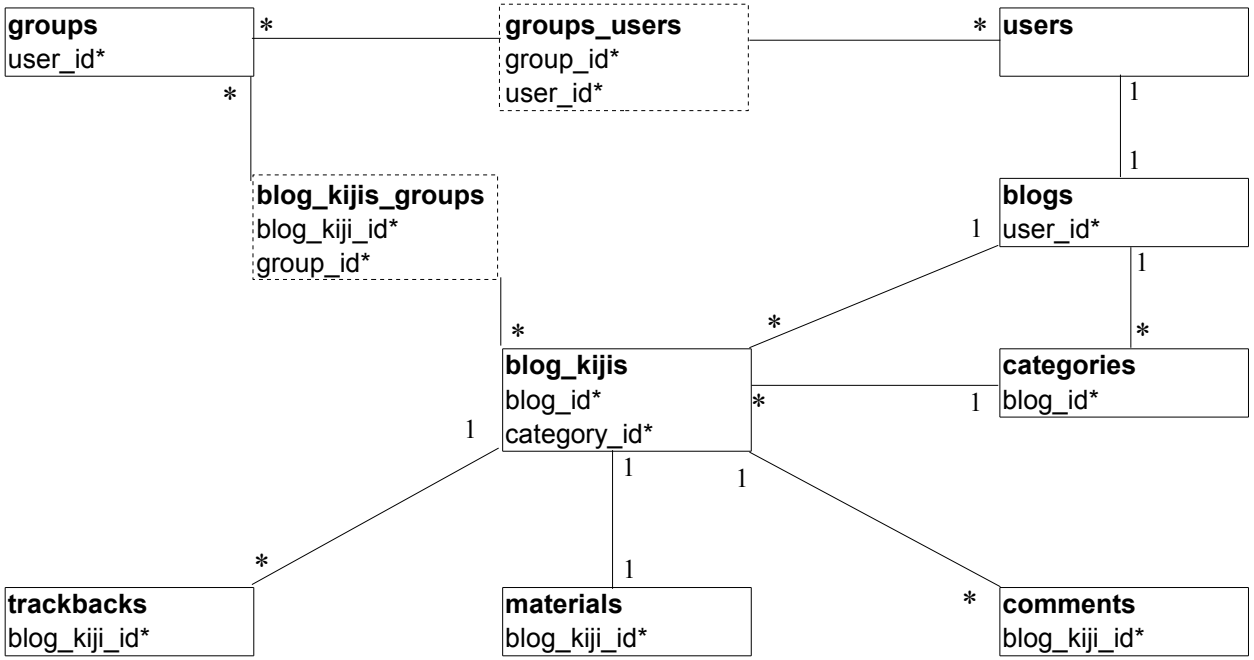
blog_kijis_groups

blog_kiji_id	integer
group_id	integer

080714

データベーステーブル関連表

・太文字はテーブル名
・通常文字は外部キー
・点線は結合テーブル（多対多の関連付け用）



付録 E　Controller 別 Action リスト

■Controller の構成

/（ルート:/app/controllers）
└_application.rb
└_blog_kijis_controller.rb
└_comments_controller.rb
└_groups_controller.rb
└_login_controller.rb
└_main_controller.rb
└_（sign_controller.rb[*14]）
└_users_controller.rb

/account
　└ blog_category_controller.rb
　└ blog_kijis_controller.rb
　└ groups_controller.rb
　└ main_controller.rb

/admin
　└ base.rb
　└ blog_category_controller.rb
　└ blog_kijis_controller.rb
　└ comments_controller.rb
　└ groups_controller.rb
　└ main_controller.rb
　└ users_controller.rb

- モジュール名はどの Controller でも共通で”Common”です。
- 同名の controller は同じデータを取り扱っているメソッドで、モジュールを共有しています。
- 名称が異なっていますが、account/main_controller は他の users_controller 同じデータを取り扱っている為、モジュールでメソッドを共有利用しています。
- 作業内容の文頭に『★』がついている Action はサブルーチンです。
- Controller 名の”controller.rb”は省略して記述しています。
- 「trackback」「send_tb」は、それらしいものを作成してますが、実装はしてません。

*14　検印サーバの Controller

Controller別Action一覧（1）

Controller	mod	Action名	引数	作業内容	呼び出しﾒｿｯﾄﾞ	実行条件	View有無
Application		rescure_action_in_public	-	#例外処理（存在しないページに404NotFound表示）	-	-	-
		block_non_members	-	#ログインしていないユーザーをはじく	-	-	-
		file_exist_check	filename	#同名ファイルの存在チェック	-	-	-
		author_check	-	#編集者と投稿者の一致チェック	-	-	-
		block_guests	-	#ゲストユーザーをはじく	-	-	-
		block_by_group	-	#グループが一致しないユーザを弾く	-	-	-
		[private]					
		resume_session	-	#セッション情報がないと判定したらログアウトさせる	-	-	-
BlogKijis		new		#記事の新規作成	-	-	○
		create		#プレビュー表示、新しいブログ記事の作成	file_upload	添付ﾌｧｲﾙが存在する	○
					set_group	ｸﾞﾙｰﾌﾟ範囲指定時	
					gpg_sign	無し	
					send_kiji	無し	
		file_upload	data_file	#★添付ファイルデータの変換・署名処理	-	-	-
	*	show		#記事全文（続きも含めた）表示	-	-	○
	*	set_group	scope_groups	#★公開範囲に含めるグループの検索	-	-	-
	*	search		#ブログ記事の検索	-	-	○
		gpg_sign	body,author	#★blog記事投稿時にPGPで署名を行う	-	-	-
		send_kiji	kiji, data_id, data	#★署名サーバへのデータ送受信/二重署名（記事データ，添付データ，添付ファイルデータ）	-	-	-
		index		#トップページ：最新の記事3件	-	-	○
		title_list		#トップページ：最新の記事20件	-	-	○
		send_tb※		#トラックバックの送信	-	-	-
		trackback ※		#トラックバックの受信	-	-	-
		[private]					
		prepare_authors		#ブログを持ってるユーザーデータの取得	-	-	-
Comment	*	new	−	#コメントの新規作成			○

modマーク説明
* ：モジュール化されてるAction　○：/同名のコントローラから呼び出し　◎：Account/同名のコントローラから呼び出し

Controller別Action一覧（2）

Controller	mod	Action名	引数	作業内容	呼び出しﾒｿｯﾄﾞ	実行条件	View有無
Comment	*	create	−	#コメントの投稿（登録）	−	-	−
	*	edit	−	#編集	−	-	○
	*	update	−	#データの更新	−	-	−
Groups	*	index	−	#グループ一覧	−	-	○
	*	group_set		#同じルートを持つグループの表示	−	-	○
	*	search	−	#グループ検索	−	-	○
	*	show	−	#詳細表示	−	-	○
	*	id_list	−	#IDやグループ名の一覧表示用	−	-	○
	*	new	−	#新規登録	−	-	○
	*	create	−	#新規登録→作成	−	-	−
		[private]					
	*	prevew_group	−	#サイドバー用の処理(ルートグループ抽出)	−	-	−
Login		login	−	#ログイン	−	-	−
		logout	−	#ログアウト	−	-	−
Main		index	-	#TOPページ	-	-	○
		blog_index	-	#blog記事の一覧閲覧(※一般用)	-	-	○
		blogs_list	-	#ブログ保有者の一覧表	-	-	○
		get_archive	-	#二重署名済みの添付ファイルをダウンロード	-	-	-
		get_verify_archive	-	#復号済み添付ファイルのダウンロード	-	-	-
		key_list	-	#公開鍵のリスト	-	-	○
		get_key	-	#公開鍵ファイルのダウンロード（各ユーザ）	-	-	-
		get_server_key	-	#公開鍵ファイルのダウンロード（サーバ）	-	-	-
		kiji_verify	-	#記事の簡易検証	kiji_server_verify	通常検証時	○
		kiji_server_verify	mode,id	#★記事の通常検証	-	-	-
		how_to	-	#手作業での検証手順ページ	-	-	○
		[private]					
		file_delete	−	#★一時ファイルの削除	-	-	-
		separate_user_blog		#ログイン後のブログページへの自動遷移			

modマーク説明
* ：モジュール化されてるAction　○：/同名のコントローラから呼び出し　◎：Account/同名のコントローラから呼び出し

Controller別Action一覧(3)

Controller	mod	Action名	引数	作業内容	呼び出しﾒｿｯﾄﾞ	実行条件	View有無
Sign※		get_kiji	−		server_sign	無し	-
				#データ受信処理(記事ファイル処理のみ)	server_sign(2)	添付ファイルがあると	-
		server_sign	data, data_id ,mode	サーバ署名(署名するデータ, ID,データ種類)	-	-	-
		verify_file	−	#通常検証時のサーバ側で保存している署名部ファイルのデータ呼び出し(書き出し)	read_sign	無し	-
		read_sign	mode,data_id	保存ファイルの読み込み	-	-	-
※SignControllerは署名サーバ側のプログラム							
Users	*	index	−	#氏名による一覧	-	-	○
	*	search	−	#ユーザー検索	-	-	○
	*	show	−	#詳細表示	-	-	○
		prepare_groups	−	#グループデータの収集	-	-	-
Account/blog_ category		index	−	#TOPページ	-	-	○
	*	b_edit	−	#ブログタイトルの編集	-	-	○
	*	b_update	−	#データの更新	-	-	-
	*	c_new	−	#カテゴリの新規作成	-	-	○
	*	c_create	−	#新規登録→作成	-	-	-
	*	c_edit	−	#カテゴリの変更	-	-	○
	*	c_update	−	##カテゴリの更新	-	-	-
	*	c_delete	−	#カテゴリの削除	-	-	-
Account/blog_ kijis	*	preview_text	scope	#★公開範囲の文字置き換え	preview_text	無し	-
	*	scope_edit	−	#ブログ公開範囲の変更	-	-	○
	*	scope_update	-	##ブログ公開範囲の変更	preview_text	[プレビュー]実行時	-
					set_group	グループ範囲指定時	
	*	material_destroy	−	#添付ファイル削除	-	-	○
	*	material_delete	−	##ファイル削除処理	-	-	-

modマーク説明
* :モジュール化されてるAction　○:/同名のコントローラから呼び出し　◎:Account/同名のコントローラから呼び出し

Controller別Action一覧（4）

Controller	mod	Action名	引数	作業内容	呼び出しﾒｿｯﾄﾞ	実行条件	View有無
Account/blog_kijis		index	−	#ブログ記事の一覧	-	-	○
	○	show	−	#記事全文（続きも含めた）表示	-	-	-
	○	set_group	scope_groups	#★公開範囲に含めるグループの検索	-	-	-
	○	search		#ブログ記事の検索	-	-	○
Account/groups		index		#管理グループ一覧	-	-	○
	*	simple_name	name,kana	#★グループ名加工（取り出し）	-	-	-
	*	edit	−	#管理グループ情報の編集フォーム	simple_name	無し	○
	*	update	−	#管理グループ情報の変更	-	-	-
	*	change_users	−	#グループに属するユーザの変更	-	-	-
	*	delete	−	#グループの削除	simple_name	無し	-
					delete_edit	無し	-
	*	delete_edit	id, p_id, name, kana	#★グループ削除時の関連グループ情報の修正・保存（ID,親ID,（置換する）名前,フリ仮名）	-	-	-
	*	choose_users	−	#グループに属する会員の選択	-	-	○
	○	group_set	−	#同じルートを持つグループの表示	−	-	-
	○	search	−	#グループ検索	−	-	○
	○	show	−	#詳細表示	−	-	-
	○	id_list	−	#IDやグループ名の一覧表示用	−	-	-
	○	new	−	#新規登録	−	-	-
	○	create	−	#新規登録→作成	−	-	−
		[private]				-	
	○	prevew_group	−	#サイドバー用の処理(ルートグループ抽出)	−	-	−
Account/main		index	−	#アカウント情報の表示	-	-	○
	*	edit	−	#アカウント情報の編集	-	-	○
	*	update	−	#アカウント情報の変更	upload_key	鍵アップロード時	-
	*	upload_key	key_data	#★公開鍵のアップロード(アップロード鍵のデータ)	-	-	-

modマーク説明
* :モジュール化されてるAction　○:/同名のコントローラから呼び出し　◎:Account/同名のコントローラから呼び出し

Controller別Action一覧(5)

Controller	mod	Action名	引数	作業内容	呼び出しﾒｿｯﾄﾞ	実行条件	View有無
Admin/Base		[private]					
		block_non_administrators	-	#管理者以外をはじく	-	-	-
Admin/blog_category		index	-	#トップページ	-	-	○
	◎	b_edit	–	#ブログタイトルの編集	-	-	○
	◎	b_update	–	#データの更新	-	-	-
	◎	c_new	–	#カテゴリの新規作成	-	-	○
	◎	c_create	–	#新規登録→作成	-	-	-
	◎	c_edit	–	#カテゴリの変更	-	-	○
	◎	c_update	–	##カテゴリの更新	-	-	-
	◎	c_delete	–	#カテゴリの削除	-	-	-
Admin/blogKijis		index	-	#トップページ:最新の記事10件	-	-	○
		prepare_authors	-	#ブログを持っているユーザ(サイドバー表示用)	-	-	-
		delete	-	#記事の削除	-	-	-
	◎	preview_text	scope	#★公開範囲の文字置き換え	preview_text	無し	-
	◎	scope_edit		#ブログ公開範囲の変更	-	-	○
	◎	scope_update		##ブログ公開範囲の変更	preview_text set_group	[プレビュー]実行時 グループ範囲指定時	-
	◎	material_destroy		#添付ファイル削除	-	-	○
	◎	material_delete		##ファイル削除処理	-	-	-
	○	show		#記事全文(続きも含めた)表示	-	-	-
	○	set_group	scope_groups	#★公開範囲に含めるグループの検索	-	-	-
	○	search		#ブログ記事の検索	-	-	○

modマーク説明
* :モジュール化されてるAction　○:/同名のコントローラから呼び出し　◎:Account/同名のコントローラから呼び出し

Controller別Action一覧(6)

Controller	mod	Action名	引数	作業内容	呼び出しﾒｿｯﾄﾞ	実行条件	View有無
Admin/Comment		index	-	#コメント一覧	-	-	○
	○	new	−	#コメントの新規作成			○
	○	create	−	#コメントの投稿(登録)	−	-	−
	○	edit	−	#編集	−	-	○
	○	update	−	#データの更新	−	-	−
		delete	-	#削除	-	-	○
		search	-	#コメントの検索	-	-	○
Admin/Groups	○	index	−	#グループ一覧	−	-	○
	○	group_set	−	#同じルートを持つグループの表示	−	-	○
	○	search	−	#グループ検索	−	-	○
	○	show	−	#詳細表示	−	-	○
	○	id_list	−	#IDやグループ名の一覧表示用	−	-	○
	○	new	−	#新規登録	−	-	○
	○	create	−	#新規登録→作成	−	-	−
	◎	simple_name	name,kana	#★グループ名加工(取り出し)	-	-	-
	◎	edit	−	#管理グループ情報の編集フォーム	simple_name	無し	○
	◎	update	−	#管理グループ情報の変更	-	-	-
	◎	change_users	−	#グループに属するユーザの変更	-	-	-
	◎	delete		#グループの削除	simple_name	無し	-
					delete_edit	無し	-
	◎	delete_edit	id, p_id, name, kana	#★グループ削除時の関連グループ情報の修正・保存(ID,親ID,(置換する)名前,フリ仮名)	-	-	-
	◎	choose_users	−	#グループに属する会員の選択	-	-	○
		[private]					
	○	prevew_group	−	#サイドバー用の処理(ルートグループ抽出)	−	-	−
Admin/main		index	−	#トップページ	-	-	○

modマーク説明
* :モジュール化されてるAction　○:/同名のコントローラから呼び出し　◎:Account/同名のコントローラから呼び出し

Controller別Action一覧（7）

Controller	mod	Action名	引数	作業内容	呼び出しﾒｿｯﾄﾞ	実行条件	View有無
Admin/users		new		#新規登録	upload_key	鍵アップロード時	○
		create		#新規登録→作成			-
		upload_key	key_data	#公開鍵のアップロード(アップロード鍵のデータ)			-
	○	index	−	#氏名による一覧	-	-	○
	○	search	−	#ユーザー検索	-	-	○
	○	show	−	#詳細表示	-	-	○
	◎	edit		#アカウント情報の編集	-	-	○
	◎	update		#アカウント情報の変更	upload_key	鍵アップロード時	-
	◎	upload_key	key_data	#★公開鍵のアップロード(アップロード鍵のデータ)	-	-	-

modマーク説明
* :モジュール化されてるAction　○:/同名のコントローラから呼び出し　◎:Account/同名のコントローラから呼び出し

Controller別フィルター覧（1）

Controller	フィルター	Action名	作業内容	適用範囲(Action名)
Application	before	resume_session	#セッション情報がないと判定したらログアウトさせる	全てのController,Action
	before	block_non_members	#ログインしていないユーザをはじく	show, search,get_kiji以外
	before	block_guests	#ゲストユーザへの制限	new, create
	before	block_by_group	#閲覧許可範囲にないユーザを弾く	show
BlogKijis	before	prepare_authors	#ブログを持ってるユーザデータの取得	全体
	before	block_non_members	#ログインしていないユーザをはじく	全体
	before	block_guests	#ゲストユーザへの制限	new, edit, delete
Groups	before	prevew_group	#サイドバー用の処理	全体
	after	file_delete	#検証時の一時ファイル削除	kiji_verify
Main	before	separate_user_blog	#ログイン後のブログページへの自動遷移	blog_index
	before	block_non_members	#ログインしていないユーザをはじく	全体
	before	prepare_groups	#サイドバー表示用	全体
Users	before	block_guests	#ゲストユーザへの制限	edit, update
	before	block_non_members	#ログインしていないユーザをはじく	全体
Account/BlogCategory	before	block_guests	#ゲストユーザへの制限	全体
	before	block_non_members	#ログインしていないユーザをはじく	全体
	before	block_guests	#ゲストユーザへの制限	全体
Account/BlogKijis	before	author_check	#編集者と投稿者の一致チェック	scope_edit, scope_update
	before	block_non_members	#ログインしていないユーザをはじく	全体
Account/Groups	before	block_guests	#ゲストユーザへの制限	全体
Account/Main	before	block_non_members	#ログインしていないユーザをはじく	全体
	before	block_guests	#ゲストユーザへの制限	index以外

modマーク説明
* ：モジュール化されてるAction　〇：/同名のコントローラから呼び出し　◎：Account/同名のコントローラから呼び出し

Controller別フィルター覧（2）

Controller	フィルター	Action名	作業内容	適用範囲(Action名)
Admin/Base	before	block_non_administrators	#管理者以外をはじく	Admin以下のController全体
Admin/logKijis	before	prepare_authors	#ブログを持ってるユーザデータの取得	
Admin/Group	before	prevew_group	#サイドバー用の処理	全体

modマーク説明
* :モジュール化されてるAction　〇:/同名のコントローラから呼び出し　◎:Account/同名のコントローラから呼び出し

付録 F　View テンプレート対応一覧

■テンプレート一覧

/(ルート:/app/view)

layouts	application.rhtml

shared	404.rhtml
	_account.rhtml
	_login_form.rhtml
	_menu_bar.rhtml
	_side_bar.rhtml
	_search.rhtml

blog_kijis	create.rhtml
	index.rhtml
	new.rhtml
	search.rhtml
	show.rhtml
	title_list.rhtml
	_errors.rhtml
	_footer.rhtml
	_form.rhtml
	_list_side_bar.rhtml
	_material.rhtml
	_side_bar.rhtml

comments	edit.rhtml
	index.rhtml
	new.rhtml
	_errors.rhtml
	_form.rhtml

groups	edit.rhtml
	id_list.rhtml
	index.rhtml
	new.rhtml
	show.rhtml
	_errors.rhtml
	_form.rhtml
	_side_bar.rhtml

main	blogs_list.rhtml
	blog_index.rhtml
	how_to.rhtml
	index.rhtml
	key_list.rhtml
	kiji_verify.rhtml
	_footer.rhtml
	_side_bar.rhtml

users	index.rhtml
	show.rhtml
	_errors.rhtml
	_form.rhtml
	_side_bar.rhtml

/account

　　|_ _acc_menu.rhtml
　　|_ _side_bar.rhtml

blog_category	b_edit.rhtml
	c_edit.rhtml
	c_new.rhtml
	index.rhtml
	_b_errors.rhtml
	_c_errors.rhtml
	_c_form.rhtml

blog_kijis	index.rhtml
	material_destroy.rhtml
	scope_edit.rhtml
	_side_bar

groups	choose_users.rhtml
	edit.rhtml
	index.rhtml
	_errors.rhtml
	_form.rhtml
	_side_bar.rhtml

main	edit.rhtml
	index.rhtml
	_errors.rhtml
	_form.rhtml

/admin

blog_category	index.rhtml
	_side_bar

blog_kijis	index.rhtml
	material_destroy.rhtml
	scope_edit.rhtml
	search.rhtml
	show.rhtml
	_errors.rhtml
	_form.rhtml
	_material.rhtml
	_side_bar.rhtml

comments	edit.rhtml
	index.rhtml
	new.rhtml
	_errors.rhtml
	_form.rhtml
	_side_bar.rhtml

groups	choose_users.rhtml
	edit.rhtml
	index.rhtml
	_errors.rhtml
	_form.rhtml
	_side_bar.rhtml

main	index.rhtml
	_side_bar.rhtml

users	edit.rhtml
	index.rhtml
	new.rhtml
	show.rhtml
	_errors.rhtml
	_form.rhtml
	_side_bar.rhtml

- “layout”フォルダにはメインフレーム（ページ構成の定義）が，”share”フォルダはコントローラ関係なく共有するファイルがまとめられています．
- ”_（アンダーバー）”で始まっているファイル名は全て部分テンプレートです．
- 基本的に Action 名と対応したファイル名となっています．
- フォルダの構成は，Controllers と同じです．（ただし，View がまったく存在しない Controller もあります．）

Controller別view対応表（1）

Controller	Action名	表示項目	メインページ			サイドバー		
			パス	部分ﾃﾝﾌﾟﾚｰﾄ	表示条件	パス	部分ﾃﾝﾌﾟﾚｰﾄ	表示条件
BlogKijis	new，create	#記事の新規作成	new	form	無し	side_bar	shared/search	無し
		#プレビュー表示	create	form	無し	side_bar	shared/search	無し
		#投稿時の不備		errors	入力項目に不備がある時			
			new	form	無し			
	show	#記事全文（続きも含めた）表示		material	添付ファイルがあるとき	side_bar	shared/login_form	未ﾛｸﾞｲﾝ時
			show	fotter	無し		shared/search	無し
	search	#ブログ記事の検索	search	fotter	無し	side_bar	shared/search	無し
	index	#トップページ：最新の記事3件	index	fotter	無し	〃	〃	〃
	title_list	#トップページ：最新の記事20件	title_list	-	-	list_side_bar	shared/login_form	未ﾛｸﾞｲﾝ時
Comment	new，create	#コメントの新規作成	new	errors	入力項目に不備がある時	blog_kijis/side_bar	-	-
				form	無し			
	edit	#編集	edit	errors	入力項目に不備がある時	blog_kijis/side_bar	-	-
				form	無し			
Groups	index	#グループ一覧	index	-	-	side_bar	shared/search	無し
	group_set	#同じルートを持つグループの表示	index	-	-	〃	〃	〃
	search	#グループ検索	index	-	-	〃	〃	〃
	show	#詳細表示	show	-	-	〃	〃	〃
	id_list	#IDやグループ名の一覧表示用	id_list	-	-	〃	〃	〃
	new，create	#新規登録	new	errors	入力項目に不備がある時	〃	〃	〃
				form	無し	〃	〃	〃
Main	index	#TOPページ	index	-	-	side_bar	shared/login_form	未ﾛｸﾞｲﾝ時
	blog_index	#blog記事の一覧閲覧(※一般用)	blog_index	fotter	無し	blog_kijis/side_bar	-	-
	blogs_list	#ブログ保有者の一覧表	blogs_list	-	-	blog_kijis/side_bar	-	-
	key_list	#公開鍵のリスト	key_list	-	-	side_bar	shared/login_form	未ﾛｸﾞｲﾝ時
	kiji_verify	#記事の簡易検証	kiji_verify	-	-	side_bar	shared/login_form	未ﾛｸﾞｲﾝ時
	how_to	#手作業での検証手順ページ	how_to	-	-	side_bar	shared/login_form	未ﾛｸﾞｲﾝ時
Users	index	#氏名による一覧	index			side_bar	group_list	無し

Controller別view対応表（2）

Controller	Action名	表示項目	メインページ			サイドバー		
			パス	部分ﾃﾝﾌﾟﾚｰﾄ	表示条件	パス	部分ﾃﾝﾌﾟﾚｰﾄ	表示条件
				-	-		shared/search	無し
	search	#ユーザー検索	index	-	-	〃	〃	〃
	show	#詳細表示	show	-	-	〃	〃	〃
Account/blog_category	index	#TOPページ	index	-	-	account/side_bar	account/acc_menu	無し
	b_edit	#ブログタイトルの編集	b_edit	b_errors	入力項目に不備がある時	account/side_bar	account/acc_menu	無し
	c_new,c_create	#カテゴリの新規作成	c_new	c_errors	入力項目に不備がある時	〃	〃	〃
				c_form	無し			
	c_edit,c_update	#カテゴリの変更	c_edit	c_errors	入力項目に不備がある時	〃	〃	〃
				c_form	無し			
Account/blog_kijis	index	#ブログ記事の一覧	index	-	-	side_bar	account/acc_menu	無し
							shared/search	無し
	search	#ブログ記事の検索	index	-	-	side_bar	account/acc_menu	無し
							shared/search	無し
	scope_edit	#ブログ公開範囲の変更	scope_edit			account/side_bar	account/acc_menu	無し
	material_destroy	#添付ファイル削除	material_destroy			account/side_bar	account/acc_menu	無し
Account/groups	index	#管理グループ一覧	index	-	-	side_bar	account/acc_menu	無し
							shared/search	無し
	search	#グループ検索	index	-	-	side_bar	account/acc_menu	無し
							shared/search	無し
	edit	#管理グループ情報の編集フォーム	edit	errors	入力項目に不備がある時	account/side_bar	account/acc_menu	無し
				form	無し			
Account/main	index	#アカウント情報の表示	index			account/side_bar	account/acc_menu	無し
	edit	#アカウント情報の編集	edit	errors	入力項目に不備がある時	account/side_bar	account/acc_menu	無し
				form	無し			
Admin/blog_category	index	#トップページ	index	-	-	side_bar	admin/main/ad_menu	無し
							shared/search	無し
	b_edit	#ブログタイトルの編集	b_edit	b_errors	入力項目に不備がある時	account/side_bar	account/acc_menu	無し

Controller別view対応表（3）

Controller	Action名	表示項目	メインページ			サイドバー		
			パス	部分ﾃﾝﾌﾟﾚｰﾄ	表示条件	パス	部分ﾃﾝﾌﾟﾚｰﾄ	表示条件
Admin/blog_category	c_new	#カテゴリの新規作成	c_new	c_errors	入力項目に不備がある時	〃	〃	〃
	c_edit	#カテゴリの変更	c_edit	c_errors	入力項目に不備がある時	〃	〃	〃
				c_form	無し			
Admin/blogKijis	index	#トップページ：最新の記事20件	index	-	-	side_bar	admin/main/ad_menu	無し
							shared/search	無し
	scope_edit	#ブログ公開範囲の変更	scope_edit			〃	〃	〃
	material_destroy	#添付ファイル削除	material_destroy			〃	〃	〃
	search	#ブログ記事の検索	search	footer		〃	〃	〃
	show	#記事全文（続きも含めた）表示	show	material	添付ファイルがあるとき	〃	〃	〃
				fotter	無し			
Admin/Comment	index	#コメント一覧	index	-	-	side_bar	admin/main/ad_menu	無し
							shared/search	無し
	new	#コメントの新規作成	new	errors	入力項目に不備がある時	blog_kijis/side_bar	shared/search	無し
				form	無し			
	edit	#編集	edit	errors	入力項目に不備がある時	blog_kijis/side_bar	shared/search	無し
				form	無し			
	search	#コメントの検索	index	-	-	side_bar	admin/main/ad_menu	無し
							shared/search	無し
Admin/Groups	index	#グループ一覧	index	-	-	side_bar	admin/main/ad_menu	無し
							shared/search	無し
	group_set	#同じルートを持つグループの表示	index	-	-	〃	〃	〃
	search	#グループ検索	index	-	-	〃	〃	〃

Controller別view対応表（4）

Controller	Action名	表示項目	メインページ			サイドバー		
			パス	部分ﾃﾝﾌﾟﾚｰﾄ	表示条件	パス	部分ﾃﾝﾌﾟﾚｰﾄ	表示条件
Admin/Groups	edit	#管理グループ情報の編集フォーム	edit	errors	入力項目に不備がある時	〃	〃	〃
				form	無し			
	choose_users	#グループに属する会員の選択	choose_users	-	-	〃	〃	〃
Admin/main	index	#トップページ	index	-	-	side_bar	admin/main/ad_menu	無し
Admin/users	new	#新規登録	new,create	errors	入力項目に不備がある時	side_bar	admin/main/ad_menu	無し
				form	無し		shared/search	無し
	index	#氏名による一覧	index			〃	〃	〃
	search	#ユーザ検索	search			〃	〃	〃
	show	#詳細表示	show			〃	〃	〃
	edit	#ユーザ情報の編集	edit	errors	入力項目に不備がある時	〃	〃	〃
				form	無し			

付録 G Model,Helper 一覧

- Models について
  - 通常のデータアクセスのメソッドと validation（検証）のメソッドとは別表で記載しています。
  - 基本的にどの Model にも”JAPANESE_NAMES”というハッシュにカラム名をキーとして対応する日本語のカラム名を保存した変数が存在しています。

- Helper について
  - “simple_text”はあくまでも見栄えの為に適当に作ったものなので、基本的に改行つきの長文の出力は"simple_format”の方をお勧めします。

■Model でのテーブル間のオプション設定について

【Blogs.rb】

○削除時に、対応するデータを削除⇒Blog_kijis，Categories

○User への外部キーを author に名称変更（通常 user，user_id）

【BlogKijis.rb】

○削除時に、対応するデータを削除⇒Comments，TrackBacks，Materials

検証用メソッドリスト

Model	メソッド名	チェック内容	適用カラム
BlogKiji	validates_presence_of	#空値のチェック	heading, body, scope,
	validates_length_of	#60文字以下かチェック	heading
Category	validates_presence_of	#空値のチェック	title, blog_id
	validates_length_of	#20文字以下かチェック	title
Comment	validates_presence_of	#空値のチェック	author, body
	validates_length_of	#60文字以下かチェック	author
Groups	validates_presence_of	#空値のチェック	title, furigana, user_id
	validates_length_of	#60文字以下かチェック	title
	validates_numericality_of	#整数値かどうかの検証	kaisou, user_id
Users	validates_presence_of	#空値のチェック	name, furigana, login_name
	validates_length_of	#60文字以下かチェック	name, furigana, login_name
	validates_confirmation_of	#パスワードと確認用と一致するかのチェック	password, passphrase
	validates_uniqueness_of	#同じ値が存在しないかのチェック	furigana, login_name

Model Helper一覧（1）

Models	ﾒｿｯﾄﾞ	引数	作業内容	呼び出しﾒｿｯﾄﾞ	実行条件
Blog	japanese_name	key	#keyに対応するカラム名を日本語に読み替え	-	-
	recent_entries	limit, page	#（著者別）最新の記事の取り出し	-	-
BlogKiji	japanese_name	key	#keyに対応するカラム名を日本語に読み替え	-	-
	recent_entries	limit, page	#最新の記事の取り出し	-	-
	set_groups	get_groups	#グループの設定	-	-
Category	japanese_name	key	#keyに対応するカラム名を日本語に読み替え	-	-
	recent_entries	limit, page	#(カテゴリ別)最新の記事の取り出し	-	-
Comment	japanese_name	key	#keyに対応するカラム名を日本語に読み替え	-	-
	authenticate	id, password	#パスワードの認証。Commentオブジェクトを返す	hashed_password	無し
	password=	pw	#パスワードが送られたときの処理	hashed_password	パスワードが空でない時
	hashed_password	password, salt	#ハッシュ値の生成	-	-
Group	japanese_name	key	#keyに対応するカラム名を日本語に読み替え	-	-
	change_users	user_params, user_ids_by_page	#所属会員の変更/※ページ内に表示されたユーザの情報のみ変更	-	-
Material	japanese_name	key	#keyに対応するカラム名を日本語に読み替え	-	-
User	japanese_name	key	#keyに対応するカラム名を日本語に読み替え	-	-
	authenticate	login_name, password	#ログイン認証。Userオブジェクトを返す	hashed_password	無し
	password=	pw	#パスワードが送られたときの処理	hashed_password	パスワードが空でない時
	hashed_password	password, salt	#ハッシュ値の生成	-	-
	change_groups	group_params	#グループの設定	-	-

Model Helper一覧(2)

Helper	ﾒｿｯﾄﾞ	引数	作業内容	呼び出しﾒｿｯﾄﾞ	実行条件
Application	menu_link_to	item	#メニューバーの出力補助	-	-
	simple_text	text	#単純な改行の変換	-	-
	sign_remove	text	#署名部分の除去ヘルパー(blog_kijis/index用)	simple_text	無し
BlogKijis	scope_check	kiji_groups	#閲覧範囲の不一致チェック	-	-
Main	nomal_verify	value	#通常検証時の出力	-	-

付録 H 「arχves」メモ

- 検印サーバ側のプログラムは、DB を一切利用していない為、任意のプロジェクト名で Ruby on Rails を作成後、以下のプログラムファイルのコピーとフォルダの作成で必要な動作が得られます。

  1. /app/controllers/へ、sign_controller.rb をコピーする。
  2. 署名部分のコピーを保存するフォルダを作成しておく。
  /public/sign/kiji　　……記事の署名部のコピー保存フォルダ
  /public/sign/material　……添付ファイルの署名部コピー保存フォルダ

- 添付データや署名部の保存場所や一時ファイル名について
  ■公開鍵ファイルのパス
  ○　/public/keys/key[ユーザ ID].gpg

  ■署名部の出力ファイル名
  　基本:#{path}#{mode}_#{filename}
  1)記事署名部の保存パス
  /public/sign/kiji/kiji_sign[データ ID].txt
  2)添付ファイル署名部の保存パス
  /public/sign/material/material_sign[データ ID].txt

  ■復号済み添付ファイルのダウンロード時
  ○　署名済みデータの保存場所
  ○　public/material/Data[添付データ ID]_[記事 ID]_[アップロードファイル名].txt

  ■ユーザ署名実行時の一時ファイル名
  ○　sign[データ ID].txt

- Controller, View の一覧には記載していませんが, blog_kijiscontroller に検証の改竄の確認の取りやすさのために「edit（編集）」action を実装し, 対応する view も作成しています。実際の稼動の時には削除しておいてください。

- GnuPG の公開鍵のファイル管理（アップロード）は出来ますが、登録作業は全て手動で行っています。また、登録されている鍵の管理は全て GnuPG で行われています。

- 「ブログリスト」の一覧表示時のアイコンは  が添付ファイルあり が添付ファイル削除済み記事を示します。（適当に作ったものなので、分かりにくければ変更して下さい）