階層型相互認証に基づく 緊急災害時に有用な家族間の個人情報共有システム

長谷川友香† 小口 正人†

† お茶の水女子大学 〒 112–8610 東京都文京区大塚 2-1-1
E-mail: †yuka@ogl.is.ocha.ac.jp, ††oguchi@computer.org

あらまし 緊急災害時に家族間で個人情報を共有できれば安否の確認や捜索などの面で大いに助けになる．しかし、平常時から当人の許可なしに個人情報を閲覧できるのはプライバシの観点から問題である．本研究では階層型相互認証機構を備えた家族間における個人情報共有システムを構築し，緊急時のみ当人の許可がなくても情報を閲覧できる認証の仕組みを提案する．

キーワード 階層型認証，災害，家族，個人情報，ライフログ，Google App Engine，Android

A Domestic Personal Information Sharing System useful in case of Emergency based on a Hierarchical Mutual Authentication

Yuka HASEGAWA† and Masato OGUCHI†

† Ochanomizu University
2-1-1 Otsuka, Bunkyo, Tokyo, 112–8610 Japan
E-mail: †yuka@ogl.is.ocha.ac.jp, ††oguchi@computer.org

1. はじめに

東日本大震災のような災害発生時に，真っ先に心配するのは家族の安否であろう．しかし，そのような状況では電話回線の混雑などにより家族と直接連絡を取ることは困難である．その場合，災害発生時に家族がどこで何をしていたかということを知ることができれば最適な安否の確認方法の決定や，捜索する場所の特定の上で役立つ．

近年はスマートフォンを始めとした携帯型情報機器の進歩が目覚ましく，ネットワークとストレージ技術の発展も相まって，個人の日常生活の情報，いわゆるライフログを取得し記録することは容易になってきている．例えばそのような機器のほとんどには GPS 機能が備えられており，特にスマートフォンでは比較的簡単なプログラミングにより移動履歴をログとして端末自身やサーバに記録することが可能である．

災害発生時に移動履歴情報を得ることができれば大変有用であるといえる．このような移動履歴に加え，スケジュールやメールなどの情報も家族がどこで何をしていたかを知る重要な手がかりとなる．

しかしながらこのような個人情報を平常時から許可なしに見られる状態にあるということには，家族間とはいえ抵抗があると思われる．さらに，大規模な災害時ではなくても，ある家族と連絡が取れない場合などにスケジュールだけ閲覧したいというように緊急の度合いに合わせた情報の開示制御が実現できると望ましい．

そこで本研究では平常時は個人情報を蓄積し，緊急時には閲覧を可能とするシステムを構築する．認証方法としては，階層型相互認証を提案するが，これは情報を開示するかしないかという認証ではなく，どの情報まで開示するかという緊急の度合いに合わせた制御を実現するものである．

提案手法に基づき，本研究ではクラウドとスマートフォンを用いた実装を行った．具体的には Google App Engine と Android 端末を用いて，階層型相互認証に基づいて移動履歴やスケジュールなど個人情報を家族間で適切に共有するシステムを構築した．

2. 緊急時に有用なシステムの要件と位置づけ

緊急時に有用な個人情報共有システムの要件としては，一つには緊急時のみ情報の閲覧が可能であるという点，もう一つには緊急度に合わせた制御を行えるという点が挙げられる．これ

ら要件を満たすために，何らかの情報により緊急時であるという判断ができればそれに越したことはない．しかし，緊急時であるという判断を外部の情報から下すことは困難である．また緊急時の状況は個々の家族により異なる．災害などの場合は公的機関が発する警報やニュースサイトなどの記事を頼りにシステムが判断すればよいが，家族と連絡がつかないなど家族単位での緊急時は判断できない．

そこで，緊急時であると判断する代わりに，他のユーザの情報を閲覧しようとするユーザにはペナルティが課されるようにして平常時に情報閲覧をすることへの抑止力とするという考え方ができる．その考えをもとに本研究で採用した認証方法が階層型相互認証である．詳しい認証の考え方は次章で述べる．

本研究で構築したシステムは家族間で使用されることを前提している．そのため，通常の認証とは異なり，緊急時には自分が設定したレベル以上の情報を相手に提供するということを，あらあじめ互いに合意しているものとする．また，このシステムを使用する目的は緊急時に家族を捜すため，自分を捜してもらうためであり，悪意のない利用を想定している．悪意のある利用とは，例えば自分の個人情報としては見られても問題のないダミーの情報を登録しておき，相手の情報を取得しようとするといった利用である．このシステムを通常の情報共有システムとして用いる場合はこのような悪意のある利用が頻発するであろうということが考えられるが，家族間のシステムではこのようなことを行うと緊急時に自分の正しい居場所を家族に伝えられなくなり，自分にとっても相手にとっても非常にデメリットが大きいため，ダミーの情報を登録するということは考えないものとしている．

3. 階層型相互認証

本研究における階層型相互認証とは，ユーザ各々が認証レベルを保持し，その認証レベルが同じユーザどうしは互いに相手の情報を閲覧可能なものとし，認証レベルが異なる場合には互いに閲覧不可とする認証方法である．本研究では認証レベルを一人のユーザが一つ保持するものではなく，各ユーザが他のユーザに対して一つずつ保持するものとした．例えば，ユーザ A はユーザ B に対して一つの認証レベルを保持し，ユーザ C に対しても一つの認証レベルを保持する．さらにユーザ B はユーザ A と C に対して保持し，ユーザ C はユーザ A とユーザ B に対して保持するという具合である．これらの認証レベルは互いに独立しており，例えばユーザ A がユーザ B に対する認証レベルを変更したとしてもユーザ C に対する認証レベルには影響しない．

この認証レベルに個人情報を割り当てる．本研究の実装では一例として，図 1 にあるようにレベル 0 は閲覧不可，レベル 1，2，3 はそれぞれスケジュール，移動履歴，メールまで閲覧可能という割り当てを行った．

自分の認証レベルは基本的に自由に設定可能であり，それによって相手に対して公開する自分の情報を制御することができる．例えばユーザ A がユーザ B に対する認証レベルを 1 としており，ユーザ B がユーザ A に対する認証レベルを 1 とした

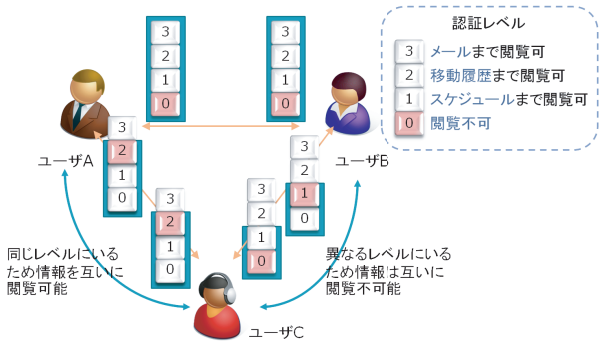


図 1　階層型相互認証の概要図

ならユーザ A と B は互いに相手のスケジュール情報を閲覧できる．ユーザ B がユーザ A にスケジュールを公開したくない場合にはユーザ A に対する認証レベルを 0 とすることで互いに何の情報も閲覧できない状態になる．

このように平常時は情報公開の制御は当人が行えば良いが，緊急時にはこの方法は現実的ではない．あるユーザが無事か確認するために情報を得たいのに，そのユーザの許可がなければ情報を閲覧できないという矛盾が生じてしまう．そこで，緊急時には認証レベルに関して次のような操作を行う．まず，情報を閲覧するユーザ（以下，閲覧ユーザ）が閲覧対象のユーザ（以下，被閲覧ユーザ）の認証レベルを任意のレベルまで上げる．そうすると閲覧ユーザの被閲覧ユーザに対する認証レベルも自動的に同じレベルまで上がり，被閲覧ユーザに対して認証レベル変更を通知するメールが送られる．こうすることで閲覧ユーザと被閲覧ユーザの互いに対する認証レベルが同じになるため，情報の閲覧が可能となる．閲覧ユーザの認証レベルが自動的に上がり，被閲覧ユーザに通知がされることで通常時にこの操作を行うことへの抑止力になると考えられる．

さらに，ユーザ間で上げられる最大の認証レベルが設定可能であり，緊急時に相手の認証レベルを変更するときでもこの最大レベルより上には上げられない仕組みとした．これにより緊急時においてもユーザが希望するレベルのプライバシは保たれる．

この認証方式の特徴として，特権ユーザが存在しないという点が挙げられる．全ての情報を閲覧できてしまう特権ユーザが家族内に存在すると，情報を閲覧されたくないという気持ちから，システム自体が使用されなくなる可能性が高いと考えられるためである．

4. 認証の取り決め

本提案方式では認証の基本的な取り決めとして以下のものを採用している．

基本的な取り決め
- 相手のレベルを上げると自分も同じレベルまで上がる
- 設定した最大値より上には上げられない
- 自分と相手のレベルの低いほうに該当する情報まで互いに閲覧可能

さらにユーザは，相手の認証レベルを変更していない状態，

相手によって自分のレベルを変更された状態，自分が相手のレベルを変更した状態の 3 状態を持ち，その状態に従って可能な操作が決まる．各ユーザから見た状態遷移の様子を図 2 に示す．

通常は「変更なし」状態におり，自分のレベルを変更しても状態は遷移しない．「変更なし」状態から遷移が起こるのは自分が相手のレベルを上げた場合か，相手が自分のレベルを上げた場合，つまり自分以外の認証レベルの操作を行ったときである．

まず，自分が相手のレベルを上げた場合は「自分が相手のレベルを変更」状態に遷移する．この状態と「変更なし」状態の相違点は，自分のレベルを下げることができない点である．ここで相手のレベルを上げておいて自分のレベルを下げることができると，相手だけを高いレベルにとどまらせることになり，相手が不利な状況を作り出すことが可能になるためである．相手がリセット操作を行うことにより，「変更なし」状態に遷移し，再び自分のレベルは自分で自由に設定できるようになる．

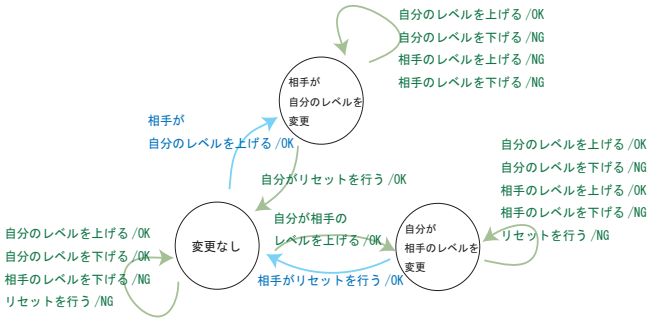


図 2　状態遷移図

相手が自分のレベルを上げた場合は「相手が自分のレベルを変更」状態に遷移する．この状態と「変更なし」状態の相違点は自分のレベルを下げられない点，相手のレベルを上げられない点，リセット操作を行える点である．相手にレベルを上げられた場合は自分のレベルは下げることができず，リセット操作を行うことになる．リセット動作を行うことで「変更なし」状態に遷移する．

「自分が相手のレベルを変更」状態で相手のレベルを上げられるのに，「相手が自分のレベルを変更」状態では相手のレベルは上げられない．これは，「自分が相手のレベルを変更」状態では，ある情報まで閲覧したが有用な情報が得られなかった場合にもう一段階レベルを上げたいということが考えられるが，「相手が自分のレベルを変更」状態では，それからまた相手のレベルを上げたいという状況になってもリセット操作をしてから相手のレベルを上げることが可能だからである．

このように，レベルを変更されたユーザだけがリセット操作を行うことができる．この遷移図は自分から見た場合であり，相手側の制御を考える場合には相手と自分を読みかえればよい．

状態に対応する取り決めをまとめると以下のようになる．

全状態で共通の取り決め
- 自分のレベルを上げられる
- 相手のレベルを下げられない

「変更なし」状態での取り決め
- 自分のレベルを下げられる
- 相手のレベルを上げられる
- リセット操作が行えない

「自分が相手のレベルを変更」状態での取り決め
- 自分のレベルを下げられない
- 相手のレベルを上げられる
- リセット操作が行えない

「相手が自分のレベルを変更」状態での取り決め
- 自分のレベルを下げられない
- 相手のレベルを上げられない
- リセット操作が行える

5.　開 発 環 境

5.1　システム構成

本研究では Google App Engine と Android 端末を用いてシステムの構築を行った．システムの構成図を図 3 に示す．ユーザから情報閲覧のリクエストを受け取ると，アクセス制御部はユーザのアカウント情報と認証レベルを参照し，該当する情報にアクセスしてデータを返す．認証レベルの変更を行うときは該当ユーザに対してメールで通知を行う．

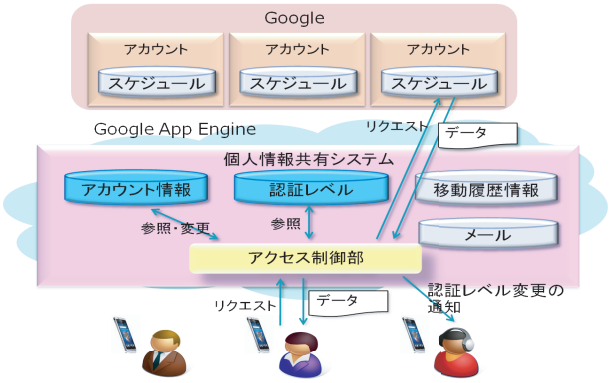


図 3　システム構成

Google App Engine 上に個人情報共有システムのサーバ側を作成した．ユーザアカウント情報やユーザ相互の認証レベル，移動履歴，メールなどの情報を保持し，階層型認証の制御を行う．ユーザはシステムが提供する Web ページにアクセスすることで情報の閲覧や認証レベルの変更などが可能である．また，Android 端末は情報の閲覧と後述する移動履歴の取得に用いた．

5.2　Google App Engine

Google App Engine は Google 社が提供するクラウドサービスで，PaaS 形式のアプリケーションプラットフォームである．Python や Java，Go 言語を使用しての開発が可能であり，本研究では Java を用いて開発した．開発環境は Eclipse を用いた．

Java で開発する場合は Web サイトの構築に Java Servlet と

呼ばれる技術を用いる．Java Servlet とはサーバ上で Web ページなどを動的に生成したりデータ処理を行うために Java で作成されたプログラムのことである．本研究では加えて，動的に Web ページを生成する技術である JSP（Java Server Pages）を用いた．

データの保存にはキーバリューストア型のデータベース管理システム「Big Table」をベースとしたデータストアと呼ばれるサービスを用いる。データストアにアクセスする方法としては標準で JDO（Java Data Objects），JPA（Java Persistent API），Low level API が提供されているが，本研究では推奨されている JDO を用いた．JDO では Java で保存したいクラスを定義し，そのクラスからオブジェクトを作成し，そのオブジェクトをデータストアに保存することになる．データストアに保存されたオブジェクトはエンティティと呼ばれる．

5.3 Android

Android は，Google 社を中心とするモバイル機器の共通プラットフォームを推進する組織「OHA（Open Handset Alliance）」によって発表された携帯情報端末向けのプラットフォームである．Google からアプリケーション開発用に Android SDK が提供されており，Eclipse を用いて開発が可能である．

6. 個人情報へのアクセス

本研究の実装で扱う個人情報はスケジュール，移動履歴とメールの 3 種類である．通常であれば個人情報へのアクセスは当人しか行えないはずであるが，情報共有のためにはシステムからアクセスする手法が必要となる．それぞれの情報へのアクセスの実装方法を以下に示す．

6.1 スケジュール

スケジュールは図 3 に示したように，Google カレンダーの情報を用いる．Google カレンダーの情報を取得するために Google Data API を用いた．これは REST 形式でカレンダーやマップなどの Google のリソースを操作できる API である．Google カレンダーは一般に公開するカレンダー，特定のユーザと共有するカレンダー，ユーザ本人のみで使用するカレンダーに分けられる．一般に公開するカレンダーの情報であれば誰でも取得できるが，特定のユーザと共有しているカレンダーとユーザ本人のみで使用するカレンダーの情報はアクセス権限がないと取得できない．そのため，個人情報共有システムにこの権限を与える必要がある．

本研究ではこれを OAuth 認証を用いて実現した．OAuth 認証とは，あらかじめ信頼関係を構築したサービス間でユーザの同意のもとにセキュアにユーザの権限を受け渡しする「認可情報の委譲」のための仕様である．OAuth 認証の概念図を図 4 に示す．

まずユーザが個人情報共有システムに対して，Google からカレンダーへのアクセス権限を取得するように指示する（1）．個人情報共有システムは Google にアクセスし，リクエストトークンを取得する（2）．個人情報共有システムはユーザを Google にリダイレクトする（3）．ユーザは Google のページ上でカレンダーへのアクセス権限をシステムに委譲する（4）．Google

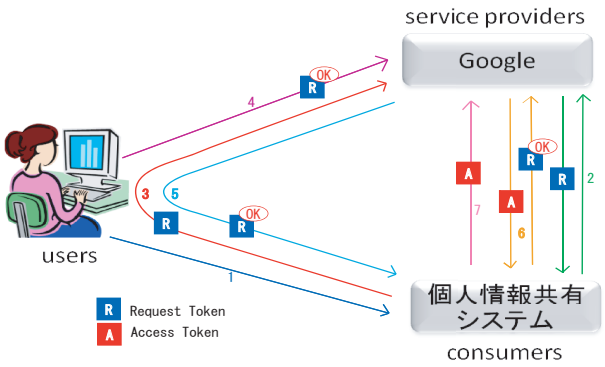


図 4　OAuth 認証の概念図

はシステムにユーザをリダイレクトする（5）．このとき，認可済みのリクエストトークンが渡される．システムは認可済みのリクエストトークンを用いて Google にアクセスし，アクセストークンを取得する（6）．次回からはアクセストークンを用いることでユーザを介することなくカレンダー情報にアクセスできる（7）．

以上のような手順でシステムの初回利用時にユーザは OAuth 認証を行いシステムにカレンダーへのアクセス権限を委譲する．アクセストークンをシステムが保持しておくことで必要が生じた場合にそれを用いてシステムは Google カレンダーにアクセスし，情報を取得できる．

6.2 移動履歴

移動履歴は各ユーザが持つ Android 端末上で動くアプリケーションにより取得する．サービスという技術を用いて，アクティブなアプリケーションとは関係なく一定時間ごとに GPS で位置情報を取得する．位置情報を取得すると Google App Engine 上で動く個人情報共有システムに緯度，経度情報とユーザ ID が送信される．この送信は，URL にそれらの情報をパラメタとして付与し POST することで行う．システムはそれを受け取ると更新日時を加えてデータストアに保存する．移動履歴の取得のリクエストがあると，リクエストの際にパラメタとして付与されたユーザ ID や取得したいデータの期間から検索し，条件にあうデータを軽量のデータ交換フォーマットである JSON(JavaScript Object Notation) 形式で返す．

この位置情報の保存と取得動作はシステム上に WebAPI を用意することで実現した．

6.3 メール

メールは個人情報共有システムに自動転送し，蓄積することでアクセスを可能にした．本研究ではメールサービスとして Gmail からの転送を想定して実装を行った．Gmail はメールの自動転送機能を提供しているが，その設定の際には転送先に Google からメールが送られ，そのメールにあるリンクをクリックして転送を確認するか，メール中の確認コードを転送元の設定画面に入力する必要がある．本システムでは後者を選択し，転送確認のメールを受信したら該当ユーザが閲覧するページで確認コードを表示し，ユーザがそのコードを Gmail の転送設定画面で入力することとした．さらに，Google App Engine

にメールを蓄積する際，メールをそのまま保存することはできないため，メール保存用のクラスを作成し，受信したメールから必要な部分を取り出して保存した．

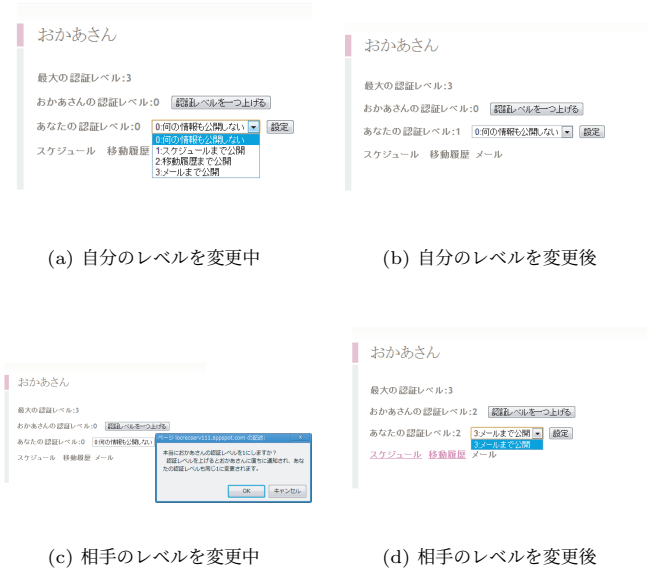
7. 階層型相互認証機構の実装

本章では階層型相互認証機構の実装方法について説明する．まず，データストアに保存するデータのクラスとして AuthGroup というクラスを作成した．このクラスのメンバは，ユーザ 1 の ID，ユーザ 2 の ID，ユーザ 1 の現在の認証レベル，ユーザ 2 の現在の認証レベル，ユーザ間の認証レベルの上限値，ユーザの状態である．ユーザ 1 とユーザ 2 は順不同である．ユーザの中から二人を選ぶ組み合わせとなり，システムのユーザの総数を n とすると ${}_nC_2$ 個のオブジェクトがこのクラスから作成され，データストアにエンティティとして保存される．新規ユーザが登録されるたびに新規ユーザと既存ユーザそれぞれを対とするエンティティを作成する．

図 5　登録済みユーザ一覧画面

他のユーザの情報閲覧や認証レベルの変更などの操作は図 5 に示す「登録済みユーザ一覧」ページから行う．ここで示したのは PC サイトの画面であるが，Android 端末からもスマートフォン用サイトで同じ画面を見ることができる．このページではログインして操作を行っているユーザ（以下，操作ユーザ）を含む AuthGroup のエンティティを探し，その情報を対になっているユーザ（以下，相手ユーザ）ごとに一覧にして表示する．操作ユーザは各相手ユーザとの認証レベルを確認したり，閲覧が許されている情報に関してリンクをクリックすることで閲覧画面に遷移できる．認証レベルの変更もこのページから行う．ここで，4 章で述べたようにユーザの状態によって行える処理が異なるため，画面には実行可能な処理のみ表示する．

まず，相手ユーザに対する自分の認証レベルの変更は図 6(a) のようにリストボックスから選んで行う．任意のレベルを選んで設定ボタンを押すとシステムは該当するエンティティの操作ユーザの認証レベルを書き換え，「登録済みユーザ一覧」ページを再読み込みし変更が反映された状態で表示する．これが図 6(b) であり，認証の取り決めより相手と自分の低いほうのレベルに該当する情報まで互いに閲覧可能であるため，低いほうのレベルは 0 で両ユーザとも何の情報も閲覧できない状態である．最大の認証レベルより上のレベルはリストボックスに選択肢として表示しないようにし，選択されることがないようにしている．



(a) 自分のレベルを変更中　　(b) 自分のレベルを変更後

(c) 相手のレベルを変更中　　(d) 相手のレベルを変更後

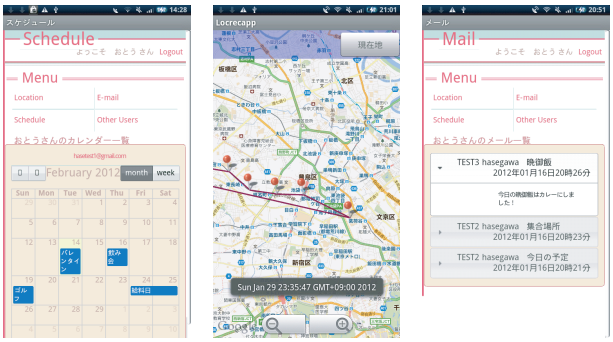
図 6　レベル変更画面

次に，相手ユーザの認証レベルの変更について説明する．この操作は「認証レベルを一つ上げる」というボタンで行い，一段階ずつレベルを上げる仕様となっている．加えて相手ユーザのレベルは上げられても下げられない仕様になっており，これは平常時に気軽に相手ユーザの認証レベル変更を行ってしまうことを防ぐためである．このボタンが押されると該当エンティティの相手ユーザの認証レベルを一つ上げ，操作ユーザの認証レベルも同じレベルに変更する．ボタンを押したときの画面を図 6(c) に示した．また，登録してある相手ユーザのメールアドレス宛てに認証レベル変更の通知を送る．このメールには認証レベルを変更したユーザの ID と現在の認証レベル，閲覧される状態にある情報が記載される．加えて，この変更が不適切である場合には個人情報共有システムにログインして認証レベルを変更し直すようにと通知し，URL を記載する．「登録済みユーザ一覧」ページを再読み込みし，変更を反映したページを表示する．このとき，閲覧できる相手ユーザの情報が一つ増えていることになる．この手順で相手のレベルを 2 まで上げた時の画面が図 6(d) である．相手のレベルを変更しているので，自分のレベルは下げることができない．相手ユーザの認証レベル変更の場合は，相手ユーザが上限の認証レベルにいる時に「認証レベルを一つ上げる」のボタンを無効にすることでそれ以上は上げられないようにしている．

8. 情報閲覧画面の実装

Android アプリケーションからの個人情報の閲覧画面は図 7 のようになっている．PC サイトからも同様に閲覧が可能である．

スケジュールについては，Google Data API を通して取得で



(a) スケジュール　(b) 移動履歴　(c) メール

図 7　Android アプリケーション使用画面

きるデータは CalendarEventEntry というクラスなので，これを JSON 形式に変換し，jQuery のプラグインである FullCalendar に渡すことで表示した．jQuery は JavaScript のライブラリの一つである．

移動履歴については Google Map を用いて表示を行った．Web サイトからの閲覧には Google Maps API を，Android アプリケーションからの閲覧には Android Maps API を用いた．移動したルートを線で表示するために Google Maps API を用いた移動履歴の表示には JavaScript を用い，Android Maps API ではオーバレイという機能を用いた．Android アプリケーションでは取得した位置にピンを表示し，このピンをタップするとその位置情報を取得した日時が表示される仕様とした．

メールについては，jQuery UI のアコーディオンを用いて表示した．

9.　まとめと今後の課題

緊急時に家族間で個人情報を共有するためのシステムを Google App Engine と Android 端末を用いて構築した．プライバシの問題を考慮し，緊急時のみ当人の許可がなくても情報の閲覧が可能にするための階層型相互認証を実装した．また，システムから個人情報にアクセスするための情報に合わせたアクセス手法を検討し，実装した．

今後の課題としては，まずシステムにユーザ間の情報に関する感じ方の違いを反映させることが挙げられる．これは，同じ情報の種類を公開し合っている状態であってもユーザによってその情報の重要性が違うのではないかということである．例えば父親が自分のメールを娘に見られる場合と娘が自分のメールを父親に見られる場合とではそれぞれの感じ方が違うという状況が想定される．そのため，父親がメールを娘に公開している状態で娘は移動履歴まで父親に公開するなど，公開する情報のバランスのとり方をユーザ間の関係性において検討する必要がある．この点に関しては，図 8 で示したようなユーザ間で対応する認証レベルをずらした階層モデルを採用することである程度ユーザの意向を組み込めると考えられる．

また，このような緊急時の認証に関しては検討課題がいくつか考えられる．まずは相手ユーザの認証レベルの変更が即座に

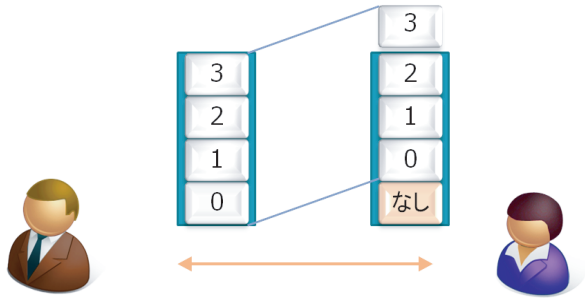


図 8　対応するレベルをずらした階層モデル

反映されることである．もしこのレベル変更が不適切なものであっても相手ユーザ自身が認証レベルを変更し直すまで情報は閲覧される状態にある．この解決策として，相手ユーザにメールが送られた後一定時間待って相手ユーザが何らかのアクションを起こさないときのみ認証レベルを変更するということが考えられる．しかし，一分一秒を争うような緊急の事態であればこの方法は本システムの有用性を欠く可能性があり，また相手ユーザが仕事中や授業中などメールを確認できない状態にあるときはあまり意味をなさない．

さらに，一人のユーザの権限で認証レベルを変更できてよいかという点も問題である．これに対しては，あらかじめ家族で何人のユーザが同意したときのみ認証レベルを変更できるというように決めておくことが考えられる．しかし，災害などで同意を得られる人数がその人数に達しないということもあり得る．

このようにプライバシを高めようとすると新たな問題点が生じてくる．これらの点については採用すべき方法を検討し実装を行っていきたい．

謝　　辞

本研究を進めるにあたり，お茶の水女子大学の小林一郎教授，とめ研究所の宇田川浩氏，日立製作所の谷本幸一氏に大変有用な助言をいただきました．

文　　献

[1] クラウド活用のための Android 業務アプリ開発入門: 出村成和, 日経 BP 社，2011
[2] すっきりわかる Google App Engine for Java クラウドプログラミング: 中田秀基，ソフトバンク クリエイティブ株式会社，2010