同期シナリオを用いてセンシング携帯端末と協調連携するアプリケーションフレームワークの提案

坂本　憲昭[†1]　坂本　一樹[†2]　名生　貴昭[†3]
市川　昌宏[†3]　新井　イスマイル[†4]　西尾　信彦[†5]

近年，複数のセンサが搭載された携帯端末の普及によって，日常のセンサデータを収集する環境が整いつつある．収集したセンサデータをマイニングすることで，ユーザの行動における周期性の発見や未来の行動予測を行う研究や個人の状況に応じたサービスが可能になる．このようなコンテキストアウェアサービスが提案されるに伴って効率的にライフログデータのセンシング及び管理を行うためのフレームワークの研究が行われている．

従来, リアルタイム性を確保した上で，高負荷なマイニングを実現をすることが課題であった．そこで，本研究では課題を解決する手法として同期シナリオを用いてサーバ・クライアント間で同期スケジュールを共有することで協調連携するフレームワークを提案し、プロトタイプの実装を行った．今後は作成したサンプルアプリケーションに提案フレームワークを適用し, 評価する．

Proposal of Application Framework Synchronized with Portable Sensing Devices

Akinori Sakamoto,[†1] Kazuki Sakamoto,[†2]
Takaaki Myoujou,[†3] Masahiro Ichikawa,[†3]
Ismail Arai[†4] and Nobuhiko Nishio[†5]

Recently, sensor-equipped portable devices have become popular. Mining collected sensor log enables various services which adapt to personal situation. As such context-aware services increase,many frameworks are studied to manage sensor logs effectively.

Current approach faces the problem of difficulty in real-time context mining. In this paper, we propose an application framework which synchronizes server-side and mobile framework to address the problem and its prototype is developed.

1. はじめに

近年，複数のセンサ (GPS や加速度センサ) が搭載された携帯端末である iPhone や Android 携帯が普及してきた．これらはネットワークに常時接続が可能であり，各個人が普段から持ち歩くため，リアルタイム性のあるセンシングが可能となっている．したがって総合的な常時センシング携帯端末としての役割が期待できる．

このような常時センシングを用いたコンテキストアウェアサービスの研究も多数行われ[1)]，これまでの大衆向けに提供されていた静的なサービスを、個人の状況に応じた動的なサービスへの移行することも行われつつある．例えば「現在地の周辺にあるイベント情報を提示する」,「友人の居場所を共有してコンタクトをとる」,「形成しているグループに応じてリマインドする」などといったサービスが提案されている[2)3)]．

さらに，リアルタイムなセンシング情報だけではなく，日々の生活の中でセンシング（ライフログセンシング）し, 情報を蓄積した膨大なログデータに対して確率的手法などを用いてマイニングすることで，ユーザの行動における周期性の発見や未来の行動予測を行う研究が数多く提案され[4)]，ライフログを用いたコンテキストアウェアサービスも提案されている．

コンテキストアウェアサービスが提案されるに伴って効率的にライフログデータのセンシングおよび管理を行うためのフレームワークの研究が行われている．Pierre-Charles David らは WildCAT[5)] というセンシング端末上で動作し，センサデータへのアクセスをドメイン化することで，上位のマイニングモジュールから抽象的に取り扱う管理手法を提案している．また，Alois Ferscha らは CASP[6)] という定義された XML フォーマットに従って，クライアントからサーバ側にセンサデータが送られ，サーバ側で管理する手法を提案している．

しかし，WildCAT のようにセンシング端末上で動作するフレームワークではリアルタイ

†1 立命館大学大学院理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University
†2 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology
†3 立命館大学情報理工学部情報システム学科
Department of Computer Science and engineering, Ritsumeikan University
†4 立命館大学総合理工学研究機構
The Research Organization of Science and Engineering, Ritsumeikan University
†5 立命館大学情報理工学部
College of Information Science and Engineering, Ritsumeikan University

ム性の求められるセンサデータとマイニング結果のマッチング処理を行うことは可能だが，高負荷なマイニングを行うことは難しい．また，CASP のようにサーバ上で動作するフレームワークでは高負荷なマイニングを行う可能だが，リアルタイムなマッチングは難しい．

そこで本研究では，記憶容量と計算資源が十分とは言い難いセンシング携帯端末とネットワーク上の豊富な資源を有するサーバ間で同期スケジュールの共有を行い協調連携させ，計算資源が必要な高負荷なマイニングはネットワーク上の端末で行うが，リアルタイム性の求められるセンサデータとマイニング結果のマッチング処理はセンシング携帯端末側で行うフレームワークを提案する．

本論文は，全 5 章で構成される．次の第 2 章では，フレームワークの関連研究の分析を行い，要件を示す．第 3 章では，提案するフレームワークについての設計と，協調連携を実現するための同期シナリオについて述べる．第 4 章では，フレームワークのプロトタイプ実装について述べ，フレームワーク上で動作するいくつかのアプリケーション例を示す．最後に第 5 章にて結論と今後の課題についてまとめる．

2. 関連研究分析と本研究の要件

本節では，前章で述べた関連研究についての分析を行い，それらの問題点を述べたうえで本フレームワークが満たすべき要件を挙げる．

WildCAT

WildCAT はセンシング端末上で動作する拡張可能な Java フレームークであり，ライフログセンシングを行うモジュールの再利用性を高める目的で研究されている．様々な種類のセンサやストレージの取り扱いを想定し，ライフログへのアクセスをドメイン化することで，上位のマイニングモジュールから抽象的に取り扱う管理手法を提案している．ライフログセンシングを行うモジュールはドメイン規則に従ったリクエストを受け付けるインタフェースを実装する必要がある．

CASP

CASP はサーバ上で動作するフレームワークであり，あらかじめ定義された XML フォーマットに従って，クライアントからライフログサーバへデータの送信が行われる．サーバ上では，Context Query API を通じて管理されているライフログデータへアクセスすることができる．また CASP では，センサの出力値をそのままサーバへアップロードするため，

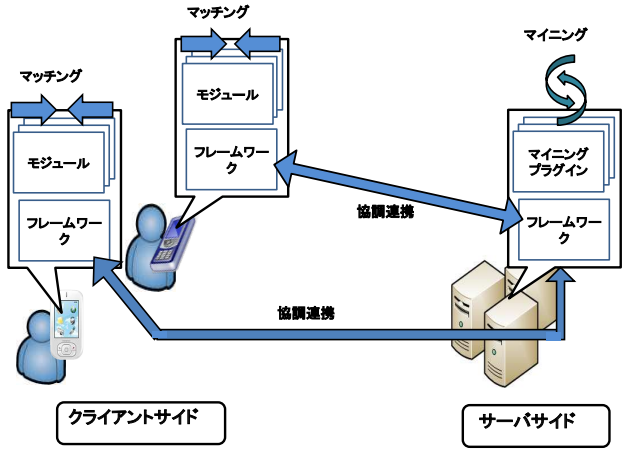

図 1　システム概要図
Fig. 1　Outline Of Framework

ネットワークの帯域を制限してしまうなどの課題があることについても言及されている．

既存フレームワークの持つ課題

WildCAT のようなフレームワークでは，ライフログセンシングを行うプラットフォーム上で，ライフログデータの管理，マイニングが行われる．そのため，リアルタイム性が求められるサービスに有効であるが，プラットフォームの処理能力が期待出来ないため，高負荷なマイニングが行えないといった問題がある．一方の CASP のようなフレームワークにおいては，豊富な計算資源が期待出来るが，ライフログデータがアップロードされるまでの間はマイニングが行えないという問題がある．したがって，本研究ではリアルタイム性を確保した上で，高負荷なマイニングを実現する．

3. 設　　計

前章で挙げた課題を解決するものとして，サーバ・クライアント間で同期スケジュールを共有することで協調連携するフレームワークを提案する．図 **1** に本システムの概要図を示す．本フレームワークはサーバサイドとクライアントサイドのフレームワーク同士が協調連携することでサーバサイドでマイニングされた結果を用いてクライアントサイドでリアルタイムなマッチングを行う．ここで, リアルタイム性を確保したうえで, 高負荷なマイニングを実現させるためには, サーバ側のマイニング結果を適切なタイミングでクライアント側に反映させる必要がある. しかし, サーバクライアント間で常にコネクションを確立させてお

くと消費電力が大きくなってしまう. よって, 本フレームワークでは各クライアントアプリケーション毎のセンシング間隔などの設定を 1 つに纏めた同期シナリオという仕組みを用いてサーバクライアント間の通信を行う. 同期シナリオについては 3.2 節で詳しく述べる.

3.1 クライアントサイドフレームワークとサーバサイドフレームワークの設計

本研究で提案するフレームワークは 2 つの機構から構成される. 1 つ目の機構は, ライフログセンシングとリアルタイムなマッチングを行うクライアントサイドフレームワークである. 2 つ目の機構は, ライフログデータの管理や高負荷なマイニングを行うサーバサイドフレームワークである. クライアントサイドフレームワークは, 各ユーザが保持するセンシング携帯端末上で個別に動作するものであり, フレームワーク上でリアルタイムにマッチングを行うアプリケーションが動作する. また, 一方, サーバサイドフレームワークは, フレームワーク上で高負荷なマイニングを行うモジュールが動作し, サービスが必要とするマイニング結果を提供する. 本研究ではこのモジュールをマイニングモジュールと呼ぶ. クライアントとサーバは後述する同期シナリオを用いて協調連携を行う. 以下でクライアントサイドフレームワークとサーバサイドフレームワークを構成する各モジュールについての詳細を述べる.

3.1.1 クライアントサイドフレームワーク

クライアントサイドフレームワークの概要図を図 **2** に示す.

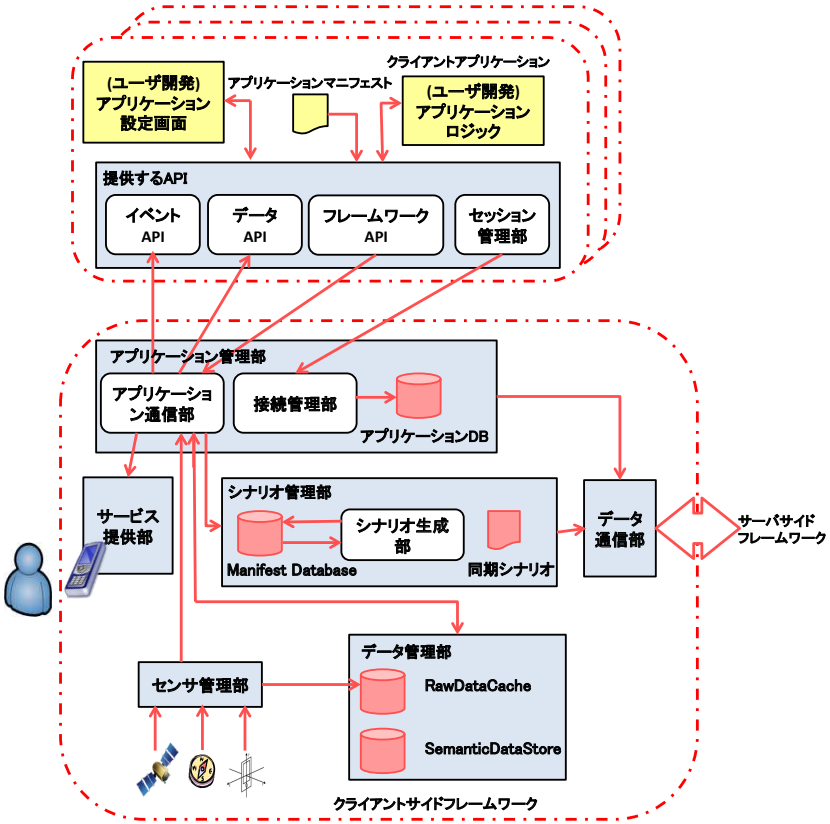

図 **2** クライアントサイドフレームワークの内部設計
Fig. 2 Design details of client-side framework

アプリケーション管理部

アプリケーション管理部では，主にクライアントアプリケーションとクライアントサイドフレームワークのコネクションを管理する．クライアントアプリケーションからフレームワークへの登録・削除要求，ライフログデータへのアクセスを受け付ける．また，リアルタイム性の求められる処理であれば，フレームワーク内で管理されているセンサ管理部からのセンサデータをクライアントアプリケーションへ伝える役割も果たす．

シナリオ管理部

シナリオ管理部は，クライアントアプリケーションの登録によって定義されるアプリケーションマニフェストを読み込む．クライアントアプリケーションの追加・削除があった際に同期シナリオがアップデートされる．アップデートの要求がかかった際には，内部で管理している Manifest Database からアプリケーションマニフェストを読み込み，同期シナリオとして生成する．アプリケーションマニフェストについては 3.1.2 節で詳しく述べる．

センサ管理部

センサ管理部は, 同期シナリオの定義を読み込み，各センサが指定された間隔での観測を開始する．観測結果はデータ管理部に渡される．また，必要に応じてクライアントアプリケーションにリアルタイムのセンサデータを提供する．

データ管理部

データ管理部は，RawDataCache と SemanticDatabase の 2 つに分類される．RawDataCache は，センサ管理部から受け取ったセンサデータを一時保存しておく．保存しておく量についてはシナリオにより定義される．SemanticDatastore には，マイニングモジュールによって作成されたデータが保存される．SemanticDatastore のスキーマは，アプリケーションマニフェストによって定義され，クライアントアプリケーションの導入時に作成される．

データ通信部

データ通信部は,HTTP でサーバサイドフレームワークとの通信を行う．同期シナリオによって定義されたタイミングでライフログデータのアップロードをはじめ，SemanticDatabase のアップデート，サーバ側ライフログデータへの検索クエリなどのリクエストを送信する. その後，各モジュールからのレスポンスをまとめ，クライアントのリクエストに対するレスポンスにデータを載せる方法 (ピギーバック) によってレスポンスを受けとる．

ピギーバックを用いる理由は, キャリアの通信ポリシへの対応として，クライアント/サーバ間の通信に HTTP を用いるためである．HTTP の特徴として, クライアントとサーバの通信は 1 回のやり取りで完結することが挙げられる．よって, 常にコネクションを確立しておかなければならないようなプロトコルと比較し，コネクションの管理が不要であり，軽量に動作するため，電力消費が少ないというメリットがある．

提供する **API**

クライアントアプリケーションに対して，フレームワークを制御するためのいくつかの API を提供する．セッション管理 API は，アプリケーションマニフェストとともに，フレームワークにクライアントアプリケーションを登録し, 管理する機能を有する．イベント受信 API は，フレームワークにてセンシングされたリアルタイムなセンサデータを受け付ける．またデータ参照部では，RawDataCache 等のストレージに対して検索クエリを実装する．フレームワーク制御 API では，同期シナリオに定義されたライフログセンシングの間隔などを，動的に変更することが可能となる．

3.1.2 サーバサイドフレームワーク

サーバサイドフレームワークの概要図を図 **3** に示す．

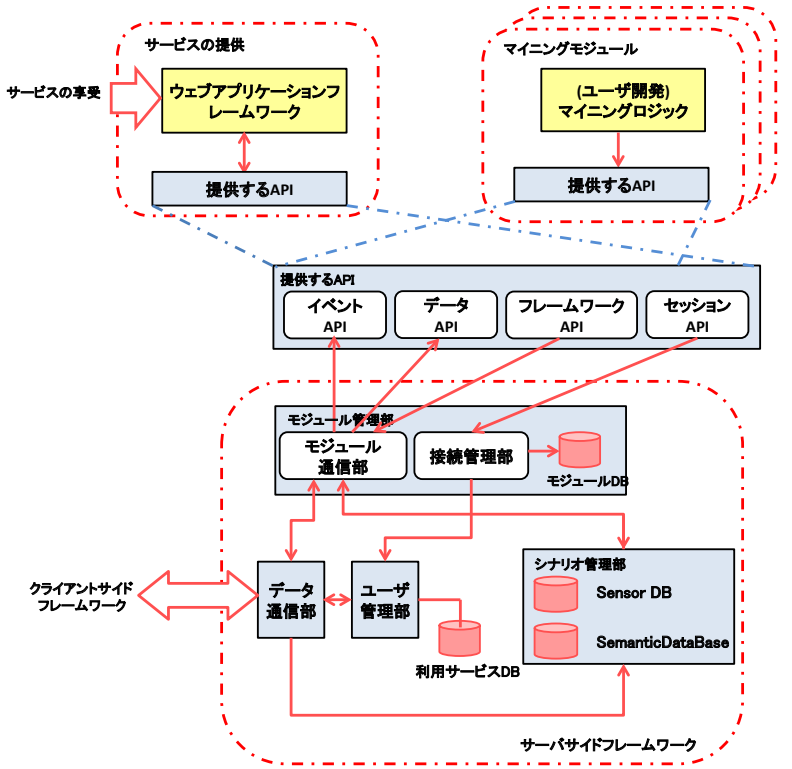

図 **3** サーバサイドフレームワークの内部設計
Fig. 3 Design details of server-side framework

データ通信部

データ通信部は,HTTP でクライアントサイドフレームワークからのクエリを受け付ける．クエリには，ライフログのアップロード，データ検索，同期シナリオの更新の 3 種類が存在する．ライフログのアップロード並びにデータ検索は，データ管理部に渡され，シナリオの更新はユーザ管理部が処理する．また，ピギーバックによるレスポンスでクライアントにメッセージを送信する．

ユーザ管理部

ユーザ管理部は, ユーザの認証情報とユーザが持つデバイスごとに同期シナリオを管理している．また, クライアントサイドにて，新たなクライアントアプリケーションを登録された際に, 同期シナリオの更新が行われる．

データ管理部

データ管理部は, ライフログのアップロード時に，ユーザ認証並びにデバイス認証をかけた上で Sensor DB に格納する．データの検索要求があった場合は，そのクエリを解決し，データ通信部を通じてクライアントサイドに結果を返す．

提供する **API** とサービス

提供する API については，構成をクライアントサイドと同様のものとした．サーバサイドフレームワークでは，提供する API を用いて，高度なマイニングを行うモジュールと，サービスを提供するモジュールを動作させることが可能である．サービス提供モジュールは，Struts[7] や Wicket[8]，Click[9] などの既存のウェブアプリケーションフレームワークを用いての実現を想定している．ウェブアプリケーションフレームワーク上で提供する API を通じてサービスの開発を行い，ユーザは WebKit ベースのモジュールを通じて，サービスを享受する．

アプリケーションマニフェスト

サーバサイドでマイニングされたデータはクライアントサイドで活用される. そのため, クライアントサイドでサーバサイドのマイニング間隔やサーバサイドへの問い合わせ間隔を指定することが望ましい. よって, 提案フレームワークでは，各クライアントアプリケーションが必要とするマイニングモジュールに対してのマイニング間隔や通信のタイミングなどについて定めたアプリケーションマニフェストを用いる. アプリケーションマニフェストを用いることで, アプリケーション開発者は, 下記のような設定を容易に実装することができる. 以降, アプリケーションマニフェストに記述される内容についての詳細を述べる．

- 利用するマイニングモジュールとマイニング間隔の指定
  各クライアントアプリケーションは利用するマイニングモジュールを定義する. また, 利用するマイニングモジュールのマイニング間隔も定義することが可能である．

- 利用するセマンティックテーブルの指定
  各マイニングモジュール毎にマイニングした結果を保持するためのテーブルをセマンティックテーブルと呼ぶ. マイニングモジュールはフレームワーク内で管理されているデータベース (セマンティックデータベース) 内に 1 つ以上のセマンティックテーブルを保持する場合が考えられる. そのため, 各クライアントアプリケーションは利用するセマンティックテーブルを定義する. フレームワークはここで指定された内容に従い, クライアント側に必要なマイニング結果を渡す．

- サーバ側への問い合わせ間隔の指定
  各クライアントアプリケーションがサーバ側へマイニング結果を取得しに行く際の問い合わせ間隔を定義する．

- 利用するセンサとインターバルの指定
  クライアントアプリケーションがサービスを提供する際に利用するセンサ情報について定義する．各センサを取得する際のインターバルについても定義することができる．また，リアルタイムにセンシングした結果の利用をサポートするため，リスナー登録の有無もここで定義する．フレームワークはここで指定された内容に従い，リスナーの登録を行ったプラグインに対してのみ，対象のセンサ情報を通知するよう動作する．

- ローカルキャッシュの設定
  本来ライフログデータは全てサーバサイドで管理されているが，マッチングを行うにあたり,「直近の 10 分のライフログデータを対象とする」や「直近の 100 サンプルを対象とする」といった要求に対応する必要がある．そのため, クライアントサイドのローカルキャッシュ内で管理されるセンサログの有効時間や要素数を定義することができる．

3.2 同期シナリオ

本フレームワークでは, クライアント上のアプリケーションとサーバ上のマイニングモジュールが必要な時に通信を発生させるため同期シナリオを用いる. 同期シナリオは, 各アプリケーションが持つアプリケーションマニフェストの記述から全てのアプリケーションの要求が満たされるように生成した 1 つのマニフェストである．

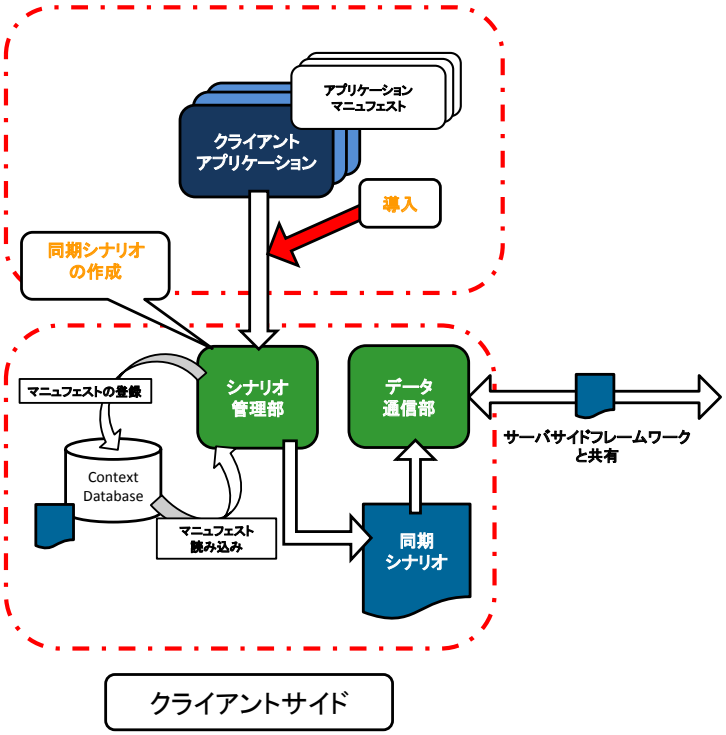

図 4　同期シナリオ生成フロー概要図
Fig. 4　Assembly of application manifests

3.2.1　同期シナリオの生成

同期シナリオ生成までのフローは図 **4** の通りである．

読み込まれたアプリケーションマニフェストは，リレーショナル型の Context Database 内で管理される．このように管理することで, 複数のアプリケーションマニフェストにまたがって同期シナリオを生成する際の高速化を図る目的がある．シナリオ管理部は Context Database よりアプリケーションマニフェストを読み込み，それぞれのマニフェストを考慮し 1 つの同期シナリオを生成する．また，動的にシナリオを変更出来るような機能も提供する．

同期シナリオはアプリケーションがフレームワーク上に導入されたことをトリガとして生成される．同期シナリオの生成に関しては, クライアントサイドフレームワークにおいてセンシング間隔の設定などを変更する同期シナリオの生成までは実装が完了している. 具体的には, センシングの間隔は, 各アプリケーションマニフェストの中の最小のセンシング間隔を採用し, センサの利用は,1 つでもセンサを利用するアプリケーションがある場合、そのセンサのセンシングが行われる同期シナリオが生成される．サーバサイドのマイニングモジュールのマイニング間隔などを考慮した同期シナリオの生成は今後の課題である．

4.　実　　装

4.1　実 装 環 境

クライアントサイドフレームワークを Android SDK[10] を用いて実装した．センシング携帯端末としては，HTC 製の Android Dev Phone 1(以降 ADP1 と呼ぶ) を用いた．ADP1 は物理センサとして GPS，加速度センサ，コンパス，温度センサが搭載されている．また、提案するクライアントサイドフレームワークはバックグラウンドでの実行をはじめ，センシングを司るモジュール，サーバサイドフレームワークと通信を行うモジュール，マイニングを行うモジュールなど多くのプロセスが並行して動作しなければならない．そのため，iPhone OS のようにシングルプロセスで制限されているものではなく，マルチプロセスが許容されているモバイルプラットフォームである Android を利用した．

サーバサイドフレームワークは，アプリケーションサーバとデータベースサーバとして 2 台を用意する．クライアントサイドフレームワークから HTTP での通信を受け付けるためのアプリケーションサーバとして，Ubuntu OS 上に Apache 並びに Tomcat を動作させた．また，同環境上で可視化ツールも動作する．データベースサーバは MySQL を用いて実現した．本研究で用いた機器の詳細を表表 **1** にまとめる．

4.2　フレームワークのプロトタイプ実装

提案したフレームワークのプロトタイプ実装として，現在は，クライアント側でセンシングを行い，得られたセンサデータをサーバ側で管理，マイニングを行うモジュールまで実装している．

端末には Bluetooth が搭載されているが、利用した Android OS 1.6 では開発アプリケーションからアクセス可能な API が組み込まれていないため、android-bluetooth プロジェクト[11] により提供されている Bluetooth ライブラリを利用したところ、周辺 Bluetooth 機器の検索は可能となったため、センサデータとしてロギングした。

また, サーバとの通信は HTTP を用い，データのアップロードなどは POST メソッドを用いてクライアントサイドフレームワークが管理しているデータベースに存在する最新の情報をサーバサイドフレームワークへアップロードする．クライアントサイドフレームワークでは，管理しているセンサデータのうち，どこからのデータをアップロードする必要があるかを示すインデックスを常に保持している．送受信されるデータのフォーマットは JSON 形式で統一した．JSON 形式は，非常にシンプルかつ軽量なフォーマットであり，送受信されるデータサイズの削減に役立つ．

4.3 サンプルアプリケーションの実装

本フレームワーク上で動作するアプリケーションのサンプルを示す．現在，フレームワークに実装されていない機能に関しては，アプリケーション側に実装している．

4.3.1 省電力機構

ライフログの中でも位置情報を蓄積するためには, 常にセンサなどのデバイスの電源を起動しておく必要がある．しかし，バッテリーの容量の限りがあるので，位置情報を常に取得し続けることは難しい．そこで，消費電力の大きいデバイスをできるだけ使わないように、段階的に加速度センサ, Wi-Fi, GPS の順に使うセンサデバイスを切り替えることで消費電力を減少させる．図 **5** にセンサデバイスの切り替えアルゴリズムの概要を示す．

停留状態から屋内移動状態及び，屋内移動状態から停留状態への状態遷移については加速度センサの値が閾値を超えるかどうかで遷移する．したがって処理自体は端末内で完結している．しかし，屋内移動状態から屋外移動状態及び屋外移動状態から屋内移動状態へ状態遷移にするかどうかの判定は，過去の Wi-Fi 及び GPS のログをマイニングした結果を用いて，現在の位置が GPS を取得できる場所なのかを判定することで行った．よって，この状態遷移を行う部分については本フレームワークが適用できると考えられる．

表 **1** フレームワークの動作環境
Table 1 Experimental environment for evaluation of the framework

	センシング携帯端末	アプリケーションサーバ	データベースサーバ
CPU	Qualcomm MSM7201A 528MHz	Intel(R) Pentium(R) D 2.80GHz	Intel(R) Xeon(R) 3070 2.66GHz
Memory	ROM:256MB RAM:192MB	1.5GB	8GB
OS	Android OS 1.6	Linux 2.6.32	Linux 2.6.32
Database	SQLite 3.5.0	-	MySQL 5.0.84

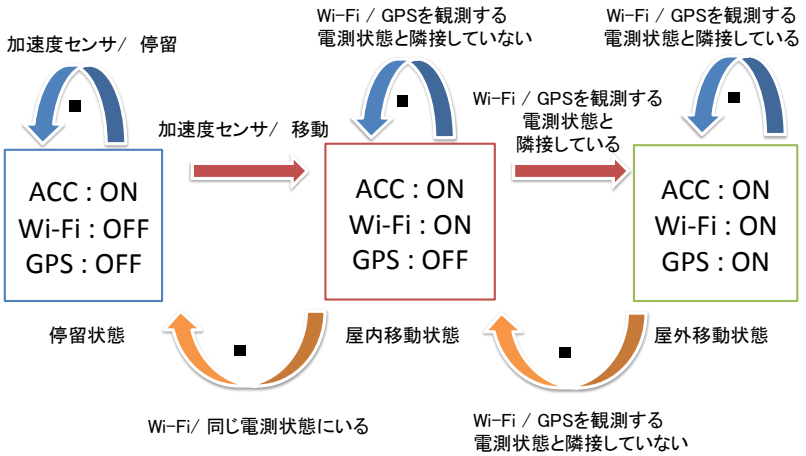

図 **5** 省電力機構の状態遷移図
Fig. 5 State transition diagram of power saving system

図 **6** に処理のフローを示す． まず，GPS マイニングモジュールがライフログデータベースのセンサログをマイニングし，結果をセマンティックデータベースに格納する．クライアントサイドフレームワークは同期シナリオに記述されたタイミングでセマンティックデータベースの同期を行う．クライアントアプリケーションはクライアントサイドフレームワークから得たリアルタイムなデータとセマンティックデータベースのデータとをマッチングを行う．その結果を用いて Wi-Fi 及び GPS のロギングを操作することで消費電力を減少させる．

4.3.2 クライアントサイドフレームワーク設定ツール

クライアントサイドフレームワークの動作確認やサーバサイドフレームワークとの連携などの設定が可能なツールを作成した．本ツールは，例えば「サービスは利用しない（クライアントアプリケーションは導入しない）が，ライフログそのものは取得し，閲覧したい」といったことも想定し，クライアント上で動作し，センシングのシナリオを定義することが可能な，フレームワークの設定を行う機能を提供する．

本ツールは図 **7** に示すように，クライアントアプリケーションとして実装した．設定ツールのメイン画面を図 **8** に示す．設定ツールが起動された際，クライアントサイドフレームワークが有する同期シナリオを読み込み，図 **9** のように表示される．ユーザは設定画面より各センサの ON/OFF やインターバルを手動で変更することが出来る．

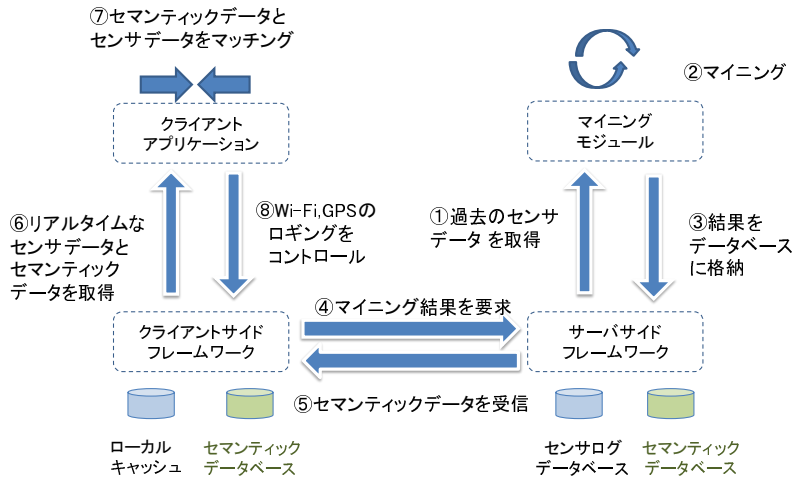

図 **6**　本フレームワークを適用した省電力機構
Fig. 6　Power saving system built on top of the proposal framewrok

また，ユーザやデバイス識別のためのアップロードの設定が行える．あらかじめ登録しておいたユーザ名，パスワードを入力し，Check ボタンをクリックすることでサーバへの認証要求を送信する．認証が成功すれば，同じくユーザ名に紐づいて登録されているデバイス名を取得し表示する．ユーザは，複数の中からデバイス名を選択する．以降，そのユーザ名，パスワード，デバイス名を利用してサーバとの協調連携が開始される（図 **10**）．

4.3.3　ライフログデータ可視化ツールの実装

前述の設定ツールによって，本フレームワークはマイニングツールなしにライフログセンシングが可能となっている．つまり，マイニングを行う前段階として，センサが日常的に示す値のみを取得可能である．またそれらを可視化することは，実世界の数値を用いたマイニング手法を考案する上で直感的であり，実用性を求めるためにも効果的である．そこで，マイニングアルゴリズムの研究者を対象とした，サーバサイドフレームワーク上で動作するライフログデータ可視化ツールを実装した．

実装したライフログ可視化ツールは，サーバサイドフレームワークとウェブアプリケーションフレームワークを連携させることで実現した．今回の実装では．ウェブアプリケーションフレームワークとして，Click Framework を用いた．Click Framework は Java で開発された軽量なウェブアプリケーションフレームワークであり，以下の特徴がある．

- フレームワーク自体が非常にシンプルであるにもかかわらず，コンポーネント指向を実現している
- 学習コストが低い
- 豊富な UI コンポーネントが用意されている

図 **11** に可視化ツール概要図とベースとなる機能画面について示す．ライフログの可視化は，位置を持つデータを地図上にマッピングする手法とグラフを用いる手法の 2 種類が存在する．

地図をベースとした可視化

GPS から得られたライフログデータを可視化するには，緯度経度情報を反映しやすい地図を用いることが効果的である．したがって，本機能は Click Framework 上で Google Maps API v3 を用いて作成した．

本機能では図 11 に示す①の地図上に，GPS によって観測された緯度・経度情報をベースとする移動軌跡を表示する．②に表示されたデバイスリストから対象となる GPS 搭載デバイスを選択し，③の設定項目より観測時間やインターバル，精度のフィルタ等を指定する．

さらにオプションとして，GPS 測位が不可能になったポイント（消失）と再度測位可能

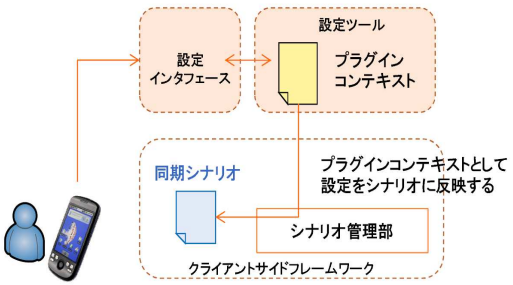

図 **7**　設定ツール概要図
Fig. 7　Configuration tool

図 **8**　設定ツールメイン画面
Fig. 8　Main screen of configuration tool

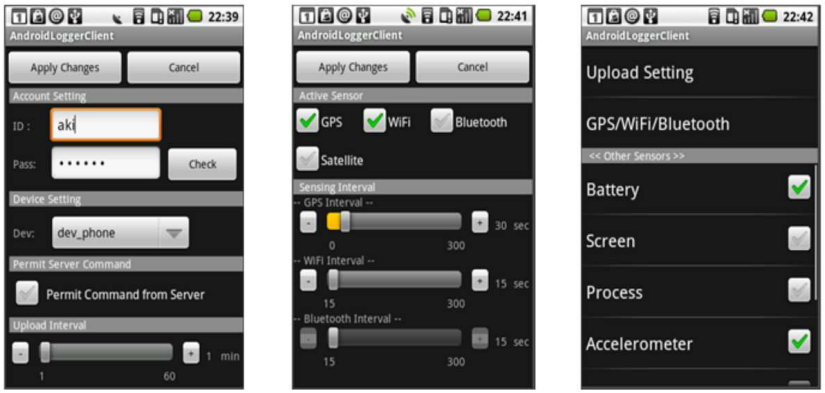

図 **9** アップロード，ライフログセンシング設定画面
Fig. 9 Upload and sensing configuration screen

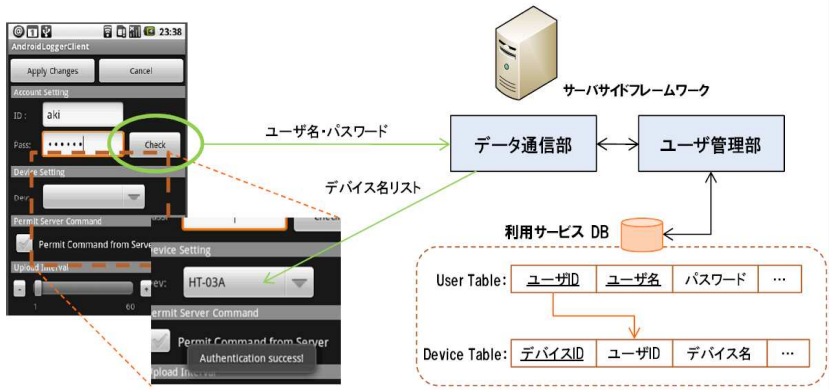

図 **10** ユーザ認証とデバイスリスト取得
Fig. 10 User authentication

になったポイント（復活）を可視化する「Lost & Found 機能」(図 **12**) と，GPS デバイスから得られた速度を色分けによって可視化する「Speed 機能」(図 **13**) を提供する．Lost & Found 機能において，GPS が最後に観測されてから 2 分以上 GPS が観測されなかった場合に，最後に観測された地点を GPS の消失点とする．また，消失してから初めて GPS が観測された地点を GPS の復活点とする．本来屋内に入ったなどの抽象化したコンテキストを用いるべきであるが，ここでは暫定的に固定の閾値を設けた．

一方の Speed 機能では，速度域ごとに色を変化させる．速度域は，0km/h（停止・青色），0〜4km/h（人間の歩行速度・水色），4〜15km/h（自転車・深緑色），15〜30km/h（バイク・緑色），30〜60km/h（自動車・黄色），60〜100km/h（電車・橙色），100km/h〜（一部電車，GPS のスパイク・赤色）として，人の行動における一般的な閾値によって定義した．

グラフをベースとした可視化

緯度経度を持たないライフログデータについては，時間軸に沿って表示するため，グラフにて表示する (図 **14**)．バッテリー残量の遷移や GPS の速度変化などの表示をサポートしている．グラフを生成するために，jFreeChart[12] のライブラリを用いて実装した．地図をベースとした可視化と同様の操作が可能である．

また地図ベース，グラフベースの双方にダウンロード機能を備えている（図 11,14 中 A）．現在可視化しているライフログデータを csv 形式でダウンロード出来る．

図 **11** 地図をベースとした可視化画面
Fig. 11 Visualization of location

図 **14** グラフによる可視化画面
Fig. 14 Visualization of battery level

図 **12** Lost and Found オプション
Fig. 12 Visualization of location about gps lost and found

図 **13** Speed オプション
Fig. 13 Visualization of speed

5. まとめと今後の課題

5.1 ま と め

本論文では，はじめに人の行動に伴って取得されたライフログデータを用いて行動を解析する，コンテキストアウェア分野に関する研究やサービスについて紹介した．コンテキストアウェアサービスが提案されるに伴って効率的にライフログデータのセンシング及び管理を行うためのフレームワークの研究が行われている事を述べ，関連研究を挙げた. そして，関連研究における課題からリアルタイム性を確保した上で，高負荷なマイニングを実現するという本フレームワークの要件を示し，センシング携帯端末とウェブサーバの協調連携を行うフレームワークを提案した.

提案フレームワークでは，資源の豊富なサーバ上で高度なマイニング処理を行い, 計算・記憶・電力資源の十分とは言えないセンシング携帯端末ではマイニング結果とリアルタイムで取得したセンサデータを用いてマッチングを行いサービスを提供する．また, 提案フレームワークにおけるクライアント/サーバの連携手法として，各キャリアのポリシを考慮し，一般的に用いることが出来る HTTP を用いた．HTTP は 1 回の通信で完結すること，通信の開始はクライアントからのみ発生することを考慮し，同期シナリオを作成することで，クライアント/サーバの協調連携を実現している.

その後，提案フレームワークのプロトタイプ実装について述べ，現在実装されている機能をもちいて省電力機構，クライアントサイドフレームワーク設定ツール，ライフログデータ

可視化ツールについて述べた.

5.2 今後の課題

今後は, 同期シナリオと API の不足部分を追加実装し, それらを用いたサンプルアプリケーションの実装とフレームワークの評価を行う.

- 同期シナリオと API の追加実装
  本フレームワークでは、同期シナリオを用いてセンシング端末と協調連携するアプリケーションフレームワークを提案しクライアント側でセンシングを行い，得られたセンサデータをサーバ側で管理，マイニングを行うモジュールまで実装した. しかし, 本フレームワークで提案した同期シナリオの生成機構やサーバ側のマイニングモジュールや, クライアント側のアプリケーションに対して提供する API に関しては実装が不足しており，今後追加実装していく.

- API を適用させたサンプルアプリケーションの作成
  4 章で記したサンプルアプリケーションは現在, 完全に API を適用した形で作成されていない. よって, 今後はサンプルアプリケーションに API を適用し, 提案フレームワークの評価を行う.

参 考 文 献

1) 鄭哲成，西尾信彦：グループドリブンのサービス提供に向けたグループコンテキスト管理機構，マルチメディア，分散，協調とモバイル（DICOMO2007）シンポジウム (2007).
2) COMEVENT, Online (2010). http://comnica.com/event/index.jsp.
3) Google Latitude, Online (2010). http://www.google.com/intl/enus/latitude/intro.html.
4) 青木政勝，瀬古俊一，西野正彬，山田智広，武藤伸洋，阿部匡伸：ライフログのための位置情報ログデータからの移動モード判定の検討 (ライフログ活用技術とその課題, オフィス情報システム, デジタルドキュメント, 一般)，情報処理学会研究報告. DD, [デジタル・ドキュメント]， Vol.2008, No.70, pp.7–12 (2008-07-17).
5) David, P.-C. and Ledoux, T.: WildCAT: a generic framework for context-aware applications, *MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, New York, NY, USA, ACM, pp.1–7 (2005).
6) Devaraju, A., Hoh, S. and Hartley, M.: A context gathering framework for context-aware mobile solutions, *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, ACM, pp.39–46 (2007).
7) Apache Struts, Online (2010). http://struts.apache.org/.
8) Apache Wicket, Online (2010). http://wicket.apache.org/.
9) Click Frameworkt, Online (2010). http://click.sourceforge.net/.
10) Android SDK, Online (2010). http://developer.android.com/sdk/index.html.
11) android-bluetooth, Online (2010). http://code.google.com/p/android-bluetooth/.
12) JFreeChart, Online (2010). http://www.jfree.org/jfreechart/.