Vol.2014-CDS-10 No.8
2014/5/22

Androidにおけるセンサ情報の記録・再生機能の設計

久米 由花[†1]　玉井 森彦[†1]　荒川 豊[†1]

概要：Android には多数のセンサが搭載されている．本研究では，これらの情報を仮想化し，端末内で記録・再生したり，他端末からネットワーク経由で共有する仕組みを提案する．端末内のセンサ類を記録・再生することで，スマートフォンを用いた行動認識システムなどのデバッグ時にアルゴリズムを修正しながら，同じルートを歩くといった被験者実験を端末内で再現可能となる．また，端末間でセンサを連携することで，バッテリの消費を分散したり，これまでになかった同一センサを複数台使ったアプリケーションの提案などが可能になると考えている．

Kume Yuka[†1]　Tamai Morihiko[†1]　Arakawa Yutaka[†1]

1. はじめに

近年，様々なセンサを搭載した Android 端末が普及している．搭載されている代表的なセンサとしては，加速度センサ，ジャイロセンサ，照度センサ，近接センサ，温度センサ，重力センサ，湿度センサなどが挙げられる．さらに 2012 年には，世界初の放射能センサを搭載した端末もソフトバンクから発売されている．また，スマートフォンの低価格化や低コストな MVNO（Mobile Virtual Network Operator: 仮想移動体通信事業者）の普及に伴い，複数台の端末を保有し，通信の種類や相手先に応じて使い分けているユーザも多くなっている．そして，複数台の端末を使い分けるだけではなく，複数台の端末で機能を連携させる（例えば，テザリングのようにインターネットに接続している端末の通信機能を周辺の端末間で共用することによって，周辺のインターネットに接続していない端末がインターネットを利用することが出来るようになる）ような使い方も増えている．

このような背景のもと，我々は，Android 端末に搭載された様々なセンサを仮想化し，テザリングのように別の端末のセンサ値をネットワーク越しに自端末で利用可能にする仕組みとして，mockSensor を提案，実装している [1] [3]．仮想化とは，近隣端末のセンサ値をネットワーク経由で，あたかも自らの端末に搭載されているセンサの様に利用することである．mockSensor では，加速度センサを仮想化の対象として，Android OS 内でその制御を行う SensorManager というクラスを独自拡張し，予め端末に内蔵されている加速度センサから出力される値以外の値を設定可能にしている．mockSensor は，複数台端末間のリアルタイムな機能共有に主眼をおいており，[3] では，SensorManager の拡張に加え，ある端末の加速度センサ値を Bluetooth に送出する Android アプリケーションと，別の端末でその値を Bluetooth から受け取り，拡張された SensorManager に渡す Android アプリケーションを実装し，ある端末で取得した加速度センサの値を別の端末の加速度センサの値として偽装できることを示している．機能共有は，機能分散と考えることもでき，複数台端末間で端末の役割を分散させることによって各端末の電力消費を分散し，保有する端末のトータルライフを延長することも可能となる．そのため，機能分散に必要な電力（主にセンサデータを Bluetooth 通信を介して複数端末間で送受信する際に消費する電力）と，機能分散による省電力効果のトレードオフについても評価し，消費電力の高いセンサであれば，機能分散による省電力化が可能であることを示している [2]．

提案手法の実装は，Android OS 4.0 から搭載された擬似ロケーション（mockLocation）機能の実装を参考にしている．mockLocation は，ユーザが任意の緯度，経度を端末に設定することを可能にする．実際の位置を偽る機能を OS 標準の機能として提供する目的としては，1) 高性能な専用 GPS 受信機を外部接続して利用することを可能にするため，2) 近年増加している位置連携アプリケーションの

[†1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

デバッグを容易にするため，3) 位置というプライバシ情報を保護するため，という理由が考えられる．位置情報だけではなく，センサに対しても，目的 1) のように外部端末との連携を可能にしたものが mockSensor である．

一方，近年，目的 3) のようにプライバシ保護を目的としたセンサ値の偽装も提案されている [4]．近年，スマートフォン用のフィットネスアプリケーションが増えている．こうしたアプリケーションの多くは，Facebook などのソーシャルネットワーキングサービスと連携しており，日々の運動ログを友人に見せることができるようになっている．これは，同じような運動をしているユーザに対してアプリケーションを宣伝する目的もあるが，日々の運動ログを友人に見せることでユーザ間の競争を促し，運動に対するモチベーションを維持させるという目的もある．しかしながら，ユーザが恒常的に運動をするようになると，運動を行わなかった場合，友人に運動を怠ったことを知らせることになりかねない．そこで，Gino Buzzelli らは，フィットネスアプリケーションを利用して運動した時の位置情報や加速度センサの値をすべて偽装することができる PocketMocker[4] を開発している．PocketMocker は，センサ値の記録と再生が可能であり，過去の運動ログの位置情報や加速度センサなどの情報を用いて，あたかも今フィットネスアプリケーションを使用して運動を行っているように見せることが可能となる．

本論文では，我々がこれまでに開発した mockSensor に対して 2 つの拡張を行うことを提案する．1 つ目は，加速度センサ以外の Android 端末に搭載されているセンサも仮想化し，仮想化したセンサ値をリアルタイムに他端末から利用可能にする事である．2 つ目は，PocketMocker のように，センサ値の記録・再生を可能にする事である．これにより，センサ A はリアルタイムに別端末からデータを取得しつつ，センサ B は過去に記録したデータを再生し，センサ C は端末に搭載された通常のセンサを利用するといったことが可能になり，複数のセンサを利用したアプリケーションのデバッグが容易になる．今回は，これらを実装するためのコントロールアプリケーションの拡張について検討した結果を報告する．

以降，第 2 章では関連研究として mockSensor と PocketMocker を挙げ，それぞれの手法について詳しく説明する．第 3 章では mockSensor の機能を拡張した提案手法について説明する．そして最後に第 4 章で本論文のまとめと今後の課題について述べる．

2. 関連研究

関連研究として，Android 端末に搭載されている加速度センサの値を仮想化し複数台端末間でリアルタイムに共有する mockSensor[1] [3] と，Android 端末に搭載されている様々なセンサ値を偽装する PocketMocker[4] の 2 つの手法について説明する．mockSensor では，端末間でセンサデータを通信する際に，各端末で消費される電力を軽減した方法を用いており，その通信方法についても説明する．

2.1 mockSensor

まず, mockSensor について説明する．Android OS では，アプリケーションでセンサを利用する際，SensorManager を介して抽象化されたセンサを利用する．ここで使用するセンサ値は自端末に内蔵されているセンサの値でも，他端末から取得したセンサの値でも構わない．mockSensor ではアプリケーションからセンサ値を手入力で設定できるようにするために，Android OS と SDK の再コンパイルを行い，いくつかの API を追加している．追加した API は以下の通りである．

- **仮想化するセンサを登録する API**
  仮想化したいセンサとそのセンサのサンプリング間隔を指定することによって仮想センサとして登録できる．
- **仮想センサに指定したセンサを解除する API**
  センサを仮想化している間，端末間でデータの通信が行われるのでセンサを仮想化しない場合は仮想センサから解除しておく．
- **仮想センサに指定したセンサの値とそのタイムスタンプを入力する API**
  仮想センサに登録されていない場合は，入力結果が反映されない．

これらの API を追加することにより，仮想化アプリケーションから，仮想化するセンサとそのサンプリング間隔を指定して仮想センサとして登録することができる．

次に，加速度センサを仮想化するための仮想化アプリケーションにおいて，リモート端末とローカル端末で設定するパラメータについてそれぞれ説明する．リモート端末では，センサの差分値に対する閾値とサンプリング間隔を設定し，Bluetooth 通信によってセンサデータを送信するローカル端末を選択する．ローカル端末では，待機時間とセンサイベントを受信済みか確認する間隔を設定し，Bluetooth 通信によってセンサデータを受信するリモート端末を選択する．それぞれの端末でこれらのパラメータを設定することで，mockSensor では加速度センサの仮想化を行うことができる．

2.2 mockSensor の通信方法

リモート端末とローカル端末間でセンサデータを送受信する方法として 3G 回線や WiFi を利用することも考えられるが，mockSensor は近隣端末間でセンサデータを共有することを前提としているため，通信による消費電力が低い Bluetooth 通信を用いて複数台端末間でセンサデータの送受信を行う．他の通信方法と比較して Bluetooth 通信によって消費される電力は低いが消費電力は通信回数に

依存するため，リモート端末で取得したセンサデータをローカル端末に送信する間隔が短い（頻繁にデータの送受信を行う）と各端末の消費電力の消耗が激しくなってしまう．しかし，リモート端末で取得したセンサデータをローカル端末に送信する間隔が長い（データの送受信があまり行われない）と仮想化センサの精度が低くなってしまう．mockSensor では，これらの問題を解決するために，Bluetooth による通信回数を減らすことで各端末の消費電力を軽減しつつ，アプリケーションが求める精度を満たす方法を提案している．

具体的には，リモート端末側では，リモート端末で取得したセンサデータと以前に取得しキャッシュに残していたセンサデータとの差分値に閾値を設定し，差分値が閾値以上である場合はローカル端末に取得したセンサデータを送信し，そうでない場合は取得したセンサデータをローカル端末に送信しないこととする．ローカル端末側では，リモート端末から受信したリモート端末のセンサデータを使用する．リモート端末から一定時間以上センサデータ受信できない場合，差分値が閾値以下であると判断しキャッシュに残していた以前のセンサデータを使用する．このように閾値を設定することで，リモート端末とローカル端末間の通信回数を減らしつつ，アプリケーションが求める精度を満たす通信を提案している．この閾値やサンプリング間隔は，アプリケーションが求める精度と実チップからの出力頻度に応じてセンサ毎に決定する必要がある．

2.3 PocketMocker

次に，PocketMocker について説明する．PocketMocker は，Android 端末に内蔵されたセンサのデータではなく偽装したセンサデータをアプリケーションに利用することによってユーザのプライバシを保護することを目的としている．偽装センサデータをアプリケーションで利用するために，Android 端末のプラットフォームまたはカーネルを改変する必要がある．

PocketMocker のアーキテクチャについて説明する．PocketMocker を利用して，GPS から取得される位置情報やネットワーク情報，加速度センサやジャイロセンサのようなセンサなどの情報を記録し，記録した過去のセンサデータを読み出し，アプリケーション上で再生することによってセンサデータを偽装する．つまり PocketMocker を利用すれば，アプリケーション上で利用するセンサの値は端末に内蔵されているセンサの値をリアルタイムに使用するだけでなく，偽装データとして過去に記録したセンサデータも使用できる仕組みになっている．

3. 提案手法

提案手法では，我々がこれまでに提案している mockSensor の 2 つの機能を拡張する．1 つ目は，Android 端末に搭載されている様々なセンサの仮想化を行い，仮想化したセンサ値を複数台端末間でリアルタイムに共有可能にする機能である．mockSensor では，加速度センサの仮想化と加速度センサ値を複数台端末間でリアルタイムに共有することが出来る．そこで mockSensor を拡張し，加速度センサ以外の Android 端末に搭載されている様々なセンサも加速度センサと同様にセンサの仮想化を行い，リアルタイムに複数台端末間でセンサ値を共有する機能の実現を目指す．その際に，センサ毎に適切なサンプリング間隔についても検討する必要がある．2 つ目は，PocketMocker のようにセンサ値を記録・再生する機能の追加である．拡張機能の提案の 1 つ目で説明したように，仮想化した様々なセンサ値をローカル端末側でリアルタイムに使用するだけでなく，コントロールアプリケーションと StorageManager を用いてローカル端末内のファイルに記録しておく．センサ値を記録することによって，ローカル端末でいつでもセンサ値を読み出し，アプリケーションに利用することが可能になる．mockSensor では，リアルタイムに取得したセンサ値しか利用することが出来なかったが，センサ値を記録・再生する機能を追加することによって，仮想化されたセンサ値を非リアルタイムにも利用することが可能になると考えられる．

以降，第 3.1 節では提案手法のアーキテクチャについて，第 3.2 節では提案手法における各端末の処理フローについて，どちらの節も既存手法と提案手法を比較しながら説明する．第 3.3 節では検討事項として，ここで設計した手法を実現する際に検討するべき項目を説明する．

3.1 アーキテクチャ

提案手法のアーキテクチャを図 1 に示す．図中の黒色の矢印は, センサデータの流れを表す．ここでは, mockSensor と同様にセンサの値を仮想化し複数台端末間でリアルタイム共有する際に，センサデータを送信する端末を「リモート端末」，センサデータを受信する端末を「ローカル端末」と呼ぶこととする．また，図中の “仮想化 APP”とは，リモート端末で取得したセンサデータを Bluetooth 通信を介してローカル端末に送信するアプリケーションで，図中の “コントロール APP”とは，Bluetooth 通信を介して受信したリモート端末のセンサデータをローカル端末で利用するためのアプリケーションである．“StorageManager” では，ローカル端末側の仮想化アプリケーションで受信したリモート端末のセンサデータを記録するためにファイル書き込みを行ったり，センサデータが書き込まれたファイルの読み出しを行う．“カスタム SensorManager” は，SensorManager を独自拡張したものであり，ローカル端末でセンサデータを偽装する部分である．具体的には，端末に内蔵されたセンサチップから取得したセンサデータを利用するか，Bluetooth 通信や仮想化アプリケーション

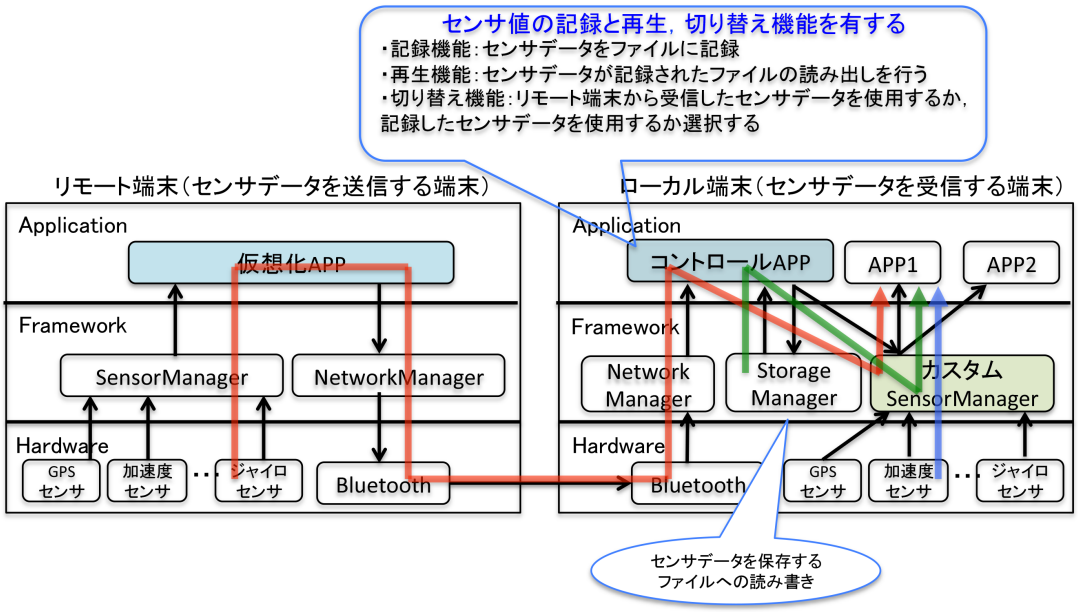

図 1 提案手法のアーキテクチャ

を介して取得した偽のセンサデータを利用するか選択することができる．

3.1.1 提案手法のセンサデータの流れ

提案手法における 3 通りのセンサデータの流れを説明する．1 パターン目は，リモート端末で端末に内蔵されているセンサデータを取得し，仮想化アプリケーションと Bluetooth 通信を用いてローカル端末に送信する．ローカル端末で Bluetooth 通信とコントロールアプリケーションを用いてリモート端末からセンサデータを受信する．ローカル端末側で受信したリモート端末のセンサデータをリアルタイムにローカル端末で使用する（図中の赤色の矢印）．2 パターン目は，コントロールアプリケーションと StorageManager を利用して，ローカル端末内のファイルに保存している過去のセンサデータを読み出し使用する（図中の緑色の矢印）．3 パターン目は，通常の Android OS の動作であるローカル端末に内蔵されているセンサデータを使用する（図中の青色の矢印）．

3.1.2 mockSensor と提案手法の違い

mockSensor と提案手法の違いについて説明する．mockSensor では加速度センサのみを仮想化していたが，提案手法では，加速度センサだけでなくリモート端末に内蔵されている様々なセンサの値を仮想化アプリケーションに送信する．仮想化アプリケーションはリモート端末に内蔵されたセンサから取得したセンサデータを Bluetooth 通信を介してローカル端末に送信する．ローカル端末では，Bluetooth 通信とコントロールアプリケーションを介してリモート端末で取得されたセンサデータを受信する．mockSensor のローカル端末のコントロールアプリケーションは，リアルタイムに取得したリモート端末のセンサデータをローカル端末で使用できるようにしているが，提案手法のローカル端末のコントロールアプリケーションは，mockSensor で実現している機能を拡張した 3 つの機能を有する．3 つの機能とは，リモート端末に内蔵されたセンサから取得したセンサデータをローカル端末内のファイルに記録する機能とローカル端末内のファイルに記録したセンサデータを読み出し，アプリケーション上で再生する機能，ローカル端末側で使用するセンサデータの切り替え機能のことである．1 つ目のリモート端末センサデータをローカル端末内のファイルに記録する機能では，リモート端末で取得したセンサデータを Bluetooth 通信を介してローカル端末に送信し，ローカル端末で Bluetooth 通信とコントロールアプリケーションを介してセンサデータを受信する．そして，受信したセンサデータは，StorageManager を介してローカル端末内のファイルに記録される．2 つ目のローカル端末内のファイルに記録したセンサデータを読み出し，アプリケーション上で再生する機能は，1 つ目のセンサデータを記録する機能で記録したリモート端末内のファイルを読み出し，ローカル端末上で使用するアプリケーション上で再生するためのものである．3 つ目のローカル端末側で使用するセンサデータの切り替え機能は，リモート端末で仮想化したセンサデータを仮想化してローカル端末で使用する場合，リアルタイムに取得したリモート端末の仮想センサデータを使用するか，リアルタイムに取得したローカル端末のセンサデータを使用するか，過去にセンサデータ記録したファイルを読み出しアプリケーション上で使用するか選択し，切り替えることが出来る機能のことである．

3.2 処理フロー

次に，提案手法におけるリモート端末とローカル端末のそれぞれの処理フローについて説明する．リモート端末の

処理フローを図 2 に，ローカル端末の処理フローを図 3 に示す．

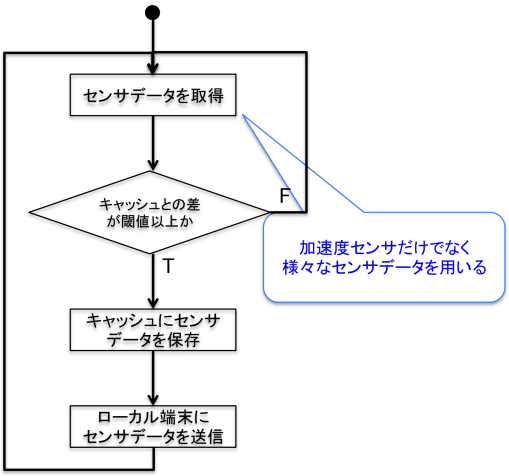

図 2 リモート端末のフロー図

提案手法のリモート端末の処理フローについて図 2 に沿って説明する．リモート端末ではまず，センサイベントを取得する．取得したセンサイベントの値とキャッシュに残っている値の差分を計算し，差分値が閾値以上である場合は取得したセンサイベントを新しくキャッシュに保存し，ローカル端末に取得したセンサイベントを送信する．差分値が閾値以下の場合は，何も行わない．

リモート端末の処理フローにおいて，既存手法である mockSensor と提案手法の違いを説明する．mockSensor では加速度センサのセンサデータのみをローカル端末へ送信するが，本手法では加速度センサ以外の様々なセンサデータもローカル端末に送信する拡張機能を提案する．

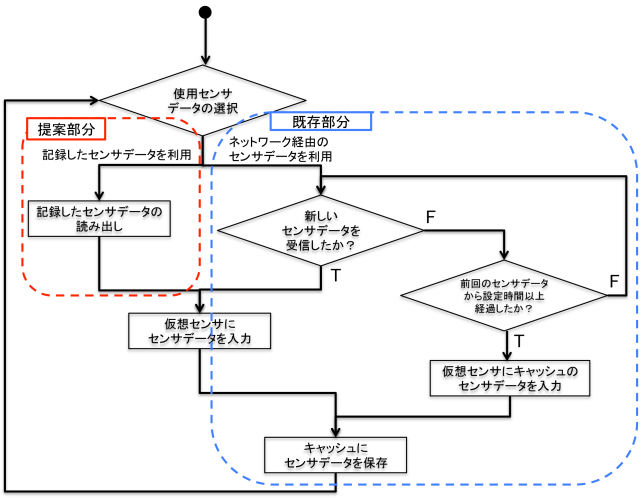

図 3 ローカル端末のフロー図

次に，提案手法のローカル端末の処理フローについて図 3 に沿って説明する．ローカル端末では始めに，Bluetooth 通信を介してリモート端末からリアルタイムに受信したセンサデータを使用するか，以前に取得し記録したセンサデータを使用するか選択する．まず，Bluetooth 通信を介してリモート端末からリアルタイムに受信したセンサデータを使用する場合について説明する．この場合，リモート端末から新しいセンサイベントを受信したかどうか調べる．新しいセンサイベントを受信している場合，仮想センサに受信したセンサイベントを入力し，キャッシュに受信したセンサイベントを保存する．リモート端末から新しいセンサイベントを受信していない場合は，新しいセンサイベントを受信したと判断するまでセンサイベントの確認を行う．次に，以前に取得し記録したセンサデータを使用する場合について説明する．この場合，記録しているセンサデータを読み出し，仮想センサにセンサイベントを入力し，キャッシュにセンサデータを保存する．

ローカル端末の処理フローにおいて，既存手法である mockSensor と提案手法の違いを説明する．mockSensor では，リモート端末から Bluetooth 通信を介してリアルタイムに受信したセンサデータをローカル端末で使用する処理のみであった．提案手法では，mockSensor で用いられていたリアルタイムに取得したリモート端末のセンサデータをローカル端末で利用する処理に加えて，以前，取得し記録したリモート端末のセンサデータをローカル端末で利用する処理を提案し追加した．そして，どちらのセンサデータを利用するか選択して利用する方法を提案している．

3.3 検討事項

これまで説明した提案手法を実現するに当たって，決定しなければならないパラメータについて以下にまとめる．

- **センサ毎の閾値**
  リモート端末では，端末に内蔵されているセンサから取得した値と過去にキャッシュに残していた値との差分値を閾値として設定する．この差分値が設定した閾値以上である場合は，新たしく取得したセンサデータをローカル端末に送信し，そうでない場合は，新しく取得したセンサデータをローカル端末に送信せず破棄する．リモート端末では，リモート端末とローカル端末間での通信回数を減らし，実データとの誤差が小さくなるような閾値を求めることが検討事項として挙げられる．データの発生頻度は，センサの種類によってデータが発生する頻度が異なり，加速度センサは頻繁にデータが発生するが，GPS センサは頻繁にデータの発生が起こらないと思われる．そこで，センサのデータ発生頻度に合わせた適切な閾値の設定を検討する必要がある．
- **センサ毎のサンプリング間隔**
  ローカル端末で，リモート端末から受信したセンサデータを使用する場合，リモート端末から一定時間以上センサデータが受信できないとき，差分値が閾値以

下であると判断しキャッシュに残していた過去のセンサデータを使用する．ローカル端末では，チップから発生する全てのセンサデータを利用すると消費電力が大きくなってしまう．これは，参考文献 [1] で実証済みである．そこで，できるだけ消費電力を軽減しつつ，実データとの誤差を小さくした閾値の決定方法について検討する必要がある．

- **ローカル端末でのセンサの記録方法**
  ローカル端末で，カスタム SensorManager がアプリケーションに返答した値をコントローラアプリケーションにも返答し，コントローラアプリケーションが StorageManager を介してローカル端末の内部ファイルに記録する方法が必要になる．
- **ローカル端末でのセンサの記録頻度**
  ローカル端末では，リモート端末からリアルタイムに受信したセンサデータを用いるだけでなく，過去に受信したセンサデータを端末内のファイルに記録し，必要なときにファイルを読み出すことによって過去のセンサデータを利用することができる．ここで用いるファイルに適切な間隔でセンサデータを記録する必要がある．センサデータをファイルへ記録する際の間隔が短いと，書き込みを行う場合も読み出しを行う場合も消費電力が大きくなってしまう．一方，センサデータをファイルへ記録する際の間隔が長いと，アプリケーションが求める精度を満たすことができないと考えられる．

4. おわりに

本論文では，Android 端末に搭載された加速度センサを仮想化し，近隣の複数台端末間で仮想化したセンサデータをリアルタイムに共有する手法である mockSensor と，Android 端末に搭載されたセンサを記録・再生しセンサデータを偽装する手法である PocketMocker について説明した．そして，mockSensor において 2 つの機能を拡張した手法を提案し，設計の説明を行った．拡張機能 1 つ目は，様々なセンサを仮想化することである．mockSensor では加速度センサのみを仮想化し，複数台端末間でリアルタイム共有する方法を実現していたが，提案手法では加速度センサ以外の様々なセンサの仮想化と仮想化したセンサデータを複数台端末間で共有する機能を提案した．拡張機能 2 つ目は，mockSensor に PocketMocker と同じようなセンサデータの記録・再生機能を追加することを提案した． 提案手法では，mockSensor で仮想化できるセンサの種類を拡張し，近隣の複数台端末間で仮想化したセンサデータを共有する機能と，取得したセンサデータを記録・再生する機能の 2 つの拡張機能を追加した手法について提案しアーキテクチャとリモート端末とローカル端末それぞれの端末における処理フローを設計し説明を行った．

この手法を実現するために，検討すべき項目が 3 つある．1 つ目はセンサ毎のサンプリング間隔，2 つ目はセンサ毎の閾値，3 つ目はローカル端末でのセンサの記録頻度である．今後は本論文で説明した設計をもとに，先程述べた 3 つの検討事項を考慮しながら実装を進めることが課題として挙げられる．

謝辞

本研究は，科学研究費補助金（25540031）の助成によって行われたものである．

参考文献

[1] 三宅 弘, 荒川 豊, 田頭 茂明, 福田 晃, “Android におけるセンサ単位の機能仮想化”, 情報処理学会研究報告, 2013-MBL-65, Vol. 8, pp. 1-8, Mar. 2013.
[2] 長堀哲, 荒川豊, 田頭茂明, 福田晃, “端末間の機能分散による消費電力平滑化手法の提案,” 情報処理学会研究報告, ユビキタスコンピューティングシステム, Vol.2013-UBI-37, No.53, pp.1–5, 2013 年 3 月.
[3] Yutaka Arakawa, Shigeaki Tagashira, Akira Fukuda, “mockSensor: Faking Remote Sensors As Embedded Sensors for a Functional Enhancement of Android”, SenSys ’13 Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems, Article No. 66, Nov. 2013.
[4] Gino Buzzelli, Nick DiRienzo, Geoffrey Challen, “SmartPhone Users Want to Be Mocked”, HotMobile ’14 The 15th International Workshop on Mobile Computing Systems and Applications, Feb. 2014.