Vol.2016-DPS-166 No.38
Vol.2016-CSEC-72 No.38
2016/3/4

Androidアプリケーション実行状態の端末間移送機構の設計と実装

佐藤 皓平[1]　毛利 公一[2]　齋藤 彰一[1]

概要：モバイル端末向けOSであるAndroidを搭載した端末が広く普及しており，一人のユーザが複数の端末を所持する場合がある．この時，ユーザは複数の端末を状況や用途に応じて使い分けることが考えられる．例えば，移動中には小型端末を利用し，オフィスでは大型のタブレット端末を利用するといった状況である．このような状況では複数の端末でシームレスに作業環境を切り替えることが作業効率向上に必要となる．本論文では，別端末上へ現在使用中のアプリケーションを移行するための基盤を構築する．この基盤は，ある端末で行っている作業を別端末上で再開できる．基盤は，移行元端末で動作しているアプリの状態を取得し，取得した状態を端末間通信を利用し送信する．移行先端末でこの状態を用いてアプリを復元する．このときの端末間通信にはBluetoothを利用する．この基盤をAndroidOSの機能として提供する．本論文ではこの基盤の実装の詳細と，基盤を利用した際の所要時間を計測と考察について述べる．

キーワード：Android, Bluetooth, 端末連係

1. はじめに

AndroidOSを搭載したスマートフォンやタブレット端末が普及している．必ずしも一人に一台ではなく，一人で二台以上の端末を所持することも一般的となった．このときユーザは複数の端末を用途に応じて使い分けることが考えられる．例えば，移動中に小型携帯端末を使ってファイルを編集し，オフィスでは，大型タブレット端末を用いて続けて同じファイルに対して作業を行うことが考えられる．このような状況で，作業の引き継ぎを行うためには次のような操作が必要となる．

(1) 小型端末でファイルを保存
(2) ファイルを端末間で移動
(3) タブレット端末でアプリケーションを立ち上げ，編集位置まで移動

これらの作業のうち，「ファイルの移動」を行うための手段として「SDカードに代表される外部記憶機器を経由」，「クラウドストレージを経由」，「ファイル転送用アプリを利用」などがある．しかし，いずれの手段もファイル編集という目的とは直接関係ない作業を行う必要があり，クラウドを利用する場合でもファイルの同期を待つ時間が発生する．このように，Android端末をまたいだ作業の引き継ぎは現状では非効率的であるといえる．

また，ユーザは複数端末の利用状況を区別して管理する必要がある．例えば，SNSへ画像を投稿する場合，投稿したい画像が保存されている端末とは別の端末でアプリを起動することがある．この時，先ほど述べたようなファイルの移動を行うか，別端末で再びアプリケーションを起動してから目的である画像の投稿を行う必要がある．これらのように，複数の端末を柔軟に使い分けることは非常に煩雑である．そこで，複数端末の柔軟な使い分けを容易にする機構が必要である．

本論文では複数のAndroidOS搭載端末間でアプリケーションの実行状態を移送するための基盤を提案する．この基盤は，アプリケーションの現在の状態と利用しているファイルを別端末に移送する機能を有する．本機能により，ユーザは端末間で実行中のアプリケーションの作業内容を移し替え，端末をまたいで作業を引き継ぐことが可能となる．本論文では，AndroidOSのソースコードに手を加えて作成したカスタムOSの詳細と，実際にGooglePlayで配信されているアプリ（以後，実アプリとする）の実行状態を移送する評価について述べる．

以後，2章で提案の前提となるAndroidOSの知識を述べ，3章で提案とその詳細を述べる．4章では実装について述べ，5章で評価を行う．7章で関連する研究に触れ，8

[1] 名古屋工業大学大学院
Nagoya Institute of Technology
[2] 立命館大学 情報理工学部
Ritsumeikan University

Vol.2016-DPS-166 No.38
Vol.2016-CSEC-72 No.38
2016/3/4

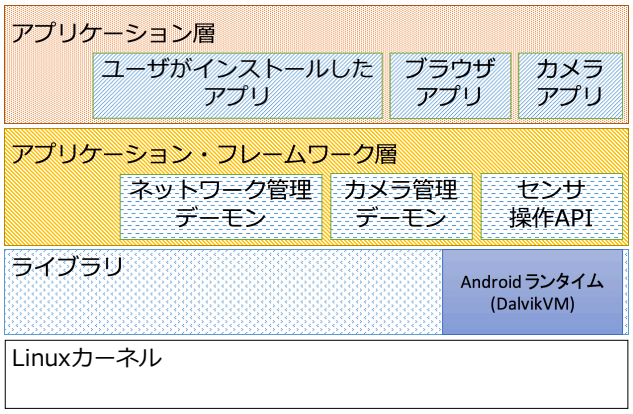

図 1 AndroidOS のレイヤー構成

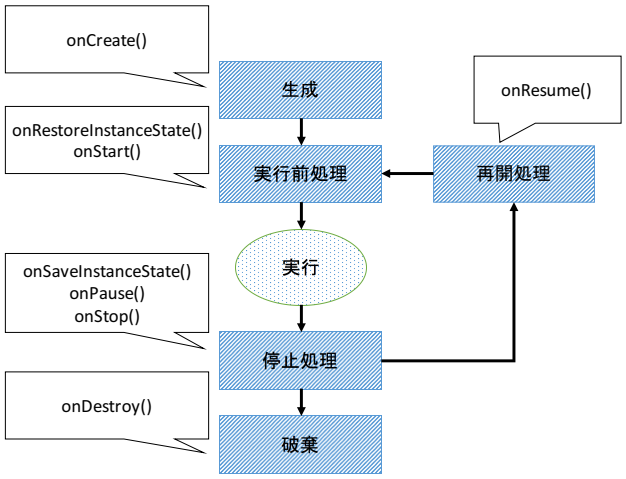

図 2 Activity のライフサイクル

章でまとめる．

2. Android の構成

本章では提案を述べる上で必要となる AndroidOS に関する知識を挙げる．

2.1 Android のレイヤー

AndroidOS は図 1 のようにカーネル層，ライブラリ層，Android ランタイム，アプリケーション・フレームワーク層，アプリケーション層から構成されている．アプリケーション・フレームワーク層は，開発者がアプリケーション開発を行う際に利用する Application Programming Interface（API）を提供する．API は Java メソッドの形で提供され，これらの API を利用することでアプリケーションは OS が提供する機能を利用することができる．ここでいう機能の例として次のようなものがある．

- 端末にインストールされているアプリケーションの一覧を取得
- 端末の通信状態を取得
- ファイルの読み書き

また，フレームワーク層では AndroidOS の機能を司るデーモンが実装され，起動している．

2.2 Android アプリケーションの構成

Android アプリケーションは一般的に Java を用いて記述され，“Activity” と “Service” から構成されている．Activity はユーザの目に触れる UI 部分を制御し，Service はユーザの目に触れない部分の処理を制御する．ユーザの目に触れない処理とは「ファイルのダウンロード」や「音楽の再生」などがある．また，すべてのアプリケーションは Activity を一つ以上含んで構成される．加えて，すべての Activity は何らかの形で “Activity.java” を継承している．この “Activity.java” はフレームワーク層に属する．

2.2.1 Activity のライフサイクル

すべての Activity は図 2 に示すライフサイクルを持つ．図中では各段階で呼び出される Activity.java 内で実装されているメソッドの例を合わせて示している．Activity が生成されると onCreate() メソッドが実行される．現在の Android アプリケーション開発では，開発者がこのメソッドをオーバーライドすることを定めている．そのため，開発者が各 Activity に対して onCreate() メソッドを記述することで自由に起動処理を定義することができる．生成された Activity は実行を開始する前に，実行前準備を行う．この準備処理は開発者が定義したときに主に発生する．この前処理が終了した後，Activity は実行を開始する．

ユーザがバックボタンかホームボタンを押すことで，実行中の Activity は見えなくなる．このとき Activity は停止状態へ移る．また，同時に実行される Activity は 1 つであり，現在実行中の Activity とは別の Activity が Running へ移るときにも Activity は停止状態へ移る．Activity が停止状態へ移る際，再び Running へ移るために必要な情報をメモリに配置する．停止状態に移った Activity 自身もメモリ上に残留し，ユーザよって呼び出された時に実行準備を行い再び Running へ移る．メモリが枯渇し開放する必要があるとき，停止状態にある Activity が破棄される場合がある．

2.3 Bundle と Intent

AndroidOS において重要な役割を果たすクラスとして Bundle クラスと Intent クラスがある．これらのクラスはアプリケーション開発で広く利用されているだけでなく，OS の機能を司る上でも重要な役割を果たす．

2.3.1 Bundle クラス

Bundle クラスはアプリ開発において Activity 内の変数の一時的な保存に用いられる．一時的な変数保存は，Activity の状態復元のために行われる．Activity が破棄されると，内部で利用していた変数が失われる．開発者やユーザが望んだタイミングで破棄が行われるとは限らず，ユーザから見ると途中だった処理が失われることになる．このため，

AndroidOS では Activity の破棄を行う際に Activity 内部の変数を Bundle オブジェクトに格納する．Activity の状態復元用の情報は Activity プロセスとは別のメモリ上に残留する．そのため Bundle オブジェクトは Activity とは別にメモリ管理され，Activity が破棄されても当該 Activity に紐付いた Bundle オブジェクトは残留する．Activity は，再生成される際に Bundle オブジェクトから変数を取り出て自ら適用することで，元の状態を復元する．なお，この時に Bundle へ保存する変数はアプリ開発者が自由に設定でき，保存した変数の復元は開発者の責任で行う．

2.3.2 Intent クラス

Intent クラスは実行中の Activity の遷移，すなわち画面を切り替える際に利用される．Activity の遷移は別の新たな Activity が起動する際に発生する．Android では Activity を起動する機構を “インテント” と呼称している．以後，本論文ではクラスについて述べる際は「Intent」，機構として述べる際は「インテント」と記述する．

Intent クラスには遷移先 Activity へ渡すパラメータや起動オプションを指定することができる．例えば，メールに記載されている Web リンクを開く際にメールアプリからブラウザアプリへ遷移するが，この時にリンクを追加情報として渡すことができる．また，ファイラーアプリからファイルを選択し，エディタアプリなどを開く際には指定されたファイル情報が渡される．このような遷移を行う際に，遷移先のアプリケーションでどのような動作を行うことを期待するかを表す “Action” を指定することができる．例えば遷移先のアプリケーションで電話をかける場合には “ACTION_DIAL” を指定する．以上のように複数のアプリケーションが連携する際にインテントが用いられる．また，ユーザがアプリケーションを新規に立ち上げる時にもインテントが用いられている．このときシステムによってインテントが実行されている．

3. 提案

本論文では Android 端末間でアプリケーションの実行状態を移送する基盤を設計する．この基盤はアプリケーションが実行中に持つべき情報をすべて移送する機能を持ち，ユーザにシームレスな端末切り替え手段を提供する．

3.1 概要

提案基盤は，複数の AndroidOS 搭載端末間でアプリケーションの実行状態を移送する機構を持つ．実際に移送するデータは実行中の Activity の実行状態とする．これは，Activity が同時に複数実行されることはないことからユーザが操作している Activity を移行できればアプリケーションを移行することと同じであるからである．ユーザが利用している端末で実行している Activity の実行状態を取得して送信し，移送先端末でこの実行状態を受信して元の Activity を起動と状態の復元を行う．受信側では元の Activity を起動するために，送信側で利用しているアプリケーションと同一のものをあらかじめインストールしておく必要がある．また，Activity の実行状態は，2.2.1 節と 2.3.1 節で述べたように，Activity がユーザのコントロールを離れる際に生成される Bundle オブジェクト内に格納されている．そこで，提案基盤はこの Bundle オブジェクトを用いて元の Activity の状態を復元する．Bundle オブジェクトを取得し，移送する上で必要となる情報を追加して端末間通信により送受信を行う．この時の端末間通信には Bluetooth を利用する．受信側端末では，送られてきた Bundle オブジェクトから Activity の情報を取り出して起動し Bundle オブジェクトを渡す．起動された Activity 内で Bundle オブジェクトから変数を取り出して元の状態を復元する．この時の復元は，既存のアプリ開発で行われている Activity の状態復元の機構をそのまま利用する．なお，実行状態は次に挙げる情報の総称とする．

- 元の Activity の処理中で利用されていた変数の値
- 元のアプリケーションの名前と実行していた Activity の名前
- 元の Activity が持っていた Intent オブジェクト
- 元の Activity が利用していたファイル

これらの情報のうち，利用されていた変数の値については開発者の責任で Bundle オブジェクトに保存及び取り出しを行う必要がある．

3.2 構成

本基盤は，次の 2 点で構成する．

(1) 端末間の通信を管理するデーモン
(2) アプリの実行状態を取得し，通信管理デーモンに渡す機構

デーモンは 2.1 節で示した Android のフレームワーク層に追加する形で実装を行い，デーモンとの連携機構は Activity.java を拡張することで実現する．また，このデーモンは，送信元で実行されていた Activity を起動する機能を実現する．以後，このデーモンのことを Migrater と呼称する．

Activity.java を拡張することで，すべてのアプリの Activity で Migrater との連携機能を利用することができる．これは，2.2 節で述べた通り，すべてのアプリが何らかの形で Activity.java を継承したクラスを一つ以上利用しているためである．

3.3 提案基盤の利用法

基盤を利用して移送を行う方法として次の 2 つを提案する．

(1) 通知領域に現れる移送開始用のアイコンをタップする方法

(2) Activity.java 内に用意したメソッド (以下スターターメソッドとする) を呼び出す方法

いずれの場合でも，処理開始後はまずユーザに送信先端末を選択させるダイアログを表示する．

3.3.1 アイコンのタップ

Activity を起動する際，画面上部に専用のアイコンを表示させる．このアイコンをタップすることで，ユーザは任意のタイミングで利用中の Activity の実行状態を移送することが可能となる．あらゆる Activity の実行中で移送を実行することが可能であるが，開発者が適切な復元情報を格納していない Activity を移送した場合には，正しく復元できるかどうか保証されない．

3.3.2 スターターメソッドの利用

2 つ目の方法は，スターターメソッドを実行することで移送する方法である．この方法を利用する場合，アプリケーション開発者がスターターメソッドを呼び出す処理を記述し，ユーザに移送実行用のインタフェースを提供する必要がある．移送できる Activity が開発者次第となるが，開発者は必要な実行状態を適切に指定できるため確実に移送先で復元を行うことができる．また，こちらの方法では実行状態の他に Activity で利用しているファイルを送信することが可能である．

3.4 提案基盤の動作

提案基盤の全体像および動作例を図 3 に示す．ユーザが専用アイコンをタップするか，スターターメソッドが実行されると 1) 送信先端末選択用ダイアログが表示される．このダイアログはユーザが利用していた Activity とは別の Activity により表示される．そのため，ユーザが利用していた Activity は停止状態へ状態遷移する．この時生成される 2)Bundle オブジェクトを Migrater へ渡す．3) 送信先が選択されると，Migrater が別端末上で待機している Migrater と Bluetooth を用いて通信を確立し，実行状態を送信する．4) 受信側の Migrater は，受け取った実行状態から元の起動していた Activity の情報を取り出し，インテントを用いて起動する．この時，送られてきた Bundle オブジェクトを Intent オブジェクトに格納し Activity に渡す．起動された受信側 Activity では，受け取った Bundle オブジェクトを，元々自らが生成したオブジェクトとして復元を行う．

また，ファイル送信を行う場合，スターターメソッドを実行してダイアログ表示 Activity を起動する処理と並行してファイル I/O 用スレッドを立ち上げ，対象ファイルを tmpfs 領域に書き込む．ここに書き込まれたファイルを Migrater が読み出し，Activity 復元用の情報と合わせて送信する．なお，送信元で起動していた Activity は移送処理後に終了する．

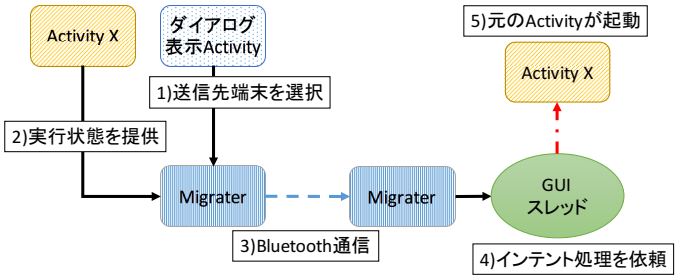

図 3 提案基盤の全体像と動作例

4. 実装

本章では 3 章で述べた提案を実現するために行った実装の詳細について述べる．実装は図 3 中の 1) から 4) を実現するために Migrater の追加，Activity.java の拡張およびダイアログ表示 Activity の追加を行う．

4.1 Migrater の OS への追加

2.1 節で述べたフレームワーク層に Migrater を追加する．フレームワーク層に追加することで Migrater を OS の機能として利用することが可能となる．また，Migrater は 2.2 節で述べた Service を利用して実装を行う．これは，OS の機能を司るフレームワーク層に新機能を追加するためには，Service として実装する必要があるためである．

Migrater には，Activity から Migrater を利用するためのインタフェースと Bluetooth による送受信を管理する機構を実装した．これらインタフェースを利用して Activity は実行状態を Migrater へ提供する．また，Bluetooth の通信にはサーバ・クライアントモデルを採用した．さらに，処理を行う Activity を制御する機構を実装した．これは，不要な処理を削減するために行う．例えば，通話を行う Activity を移送することは考えにくい．そのため，このような移送を行うことが考えられない Activity を含むアプリケーションでは処理を行わないように判定する機構を作成した．また，Activity から呼び出すことでその Activity が移送処理の対象となるか否かを真偽値を返すメソッドを用意した．

4.2 Activity.java の拡張

Activity.java を拡張し，Migrater を利用するための機能を追加した．追加した機能を次に挙げ，各機能の詳細について述べる．

(1) Migrater のインタフェースを利用するメソッド
(2) 移送を開始するためのトリガーとなる通知を表示する処理
(3) 受信側端末で起動された時に復元処理を行うための準備
(4) 受信したファイルのストレージへの書き込み

(1) は Migrater と Activity の連係に利用し，(2) はユーザが基盤を利用して移送処理を開始するためのトリガーとして利用する．(3) は別端末上で正しく Activity の状態を復元するための必要な処理であり，その一連の処理の中で必要があれば (4) で受信したファイルをストレージへ格納する．

4.2.1 Migrater を呼び出すメソッド

Migrater を呼び出すためのメソッドには大別して 2 種類がある．

(1) ユーザの操作に連動して自動的に実行されるメソッド
(2) アプリ開発者が明示的に呼び出すメソッド (スターターメソッド)

自動的に実行されるメソッドは，通知領域に現れるトリガー用通知アイコンをタップすることで実行される．これは，3.3.1 節で述べたユーザが任意のタイミングで処理を行う方式に対応する．メソッドが実行されると Activity の実行状態の移送が開始される．この通知アイコンは Activity.java の拡張により表示されるため，すべての Activity で現れることとなる．そのため，ユーザが任意の Activity について実行状態の移送を行うことができる．

開発者が明示的に呼び出すメソッドは 3.3.2 節で述べた開発者が意図して移送を行う方式に対応する．このスターターメソッドを利用すると，開発者が任意のタイミングで実行状態の移送を行うことができるため，アプリの機能の 1 つとして機構を利用することができる．また，このメソッドは引数として String 型配列を受けることができる．この配列の要素としてにファイルの絶対パスを格納すると，送信する実行状態の中にそのファイルを含めることが可能となる．メソッド内では，受け取った絶対パスを元に該当ファイルをオープンし tmpfs 領域へファイルをコピーする．

4.2.2 通知の表示

3.3.1 節と 4.2.1 節で述べた通知を表示させる．Activity が画面に表示されるタイミングで処理されるメソッド（onStart）をフックして通知を表示させる．すべての Activity で表示させる必要は無いため，4.1 節で述べた判定機構を用いて表示を行うかどうかを判定する．また，画面に表示されていない Activity について移送を行うことも無い．そこで，Activity が表示されなくなるタイミングで表示されている通知を消去する．これも Activity が画面から消えるタイミングで処理されるメソッド（onPause）をフックして処理を行う．

4.2.3 復元処理の準備

受信側端末で起動された Activity は，そのままでは元の状態を復元することはできない．これは，受信側端末上で起動した Activity は，新規にアプリケーションが起動した状態となっているためである．そこで，復元を行うための準備を行う必要がある．まず，Bundle オブジェクトが元からあったように見せかける必要がある．次に，送信側 Activity が起動したときに受け取っていた Intent オブジェクトの中に復元に必要な情報が格納されている可能性がある．そのため，送信元で利用していた Intent オブジェクトを利用できるように，受信側の Intent オブジェクトを差し替える必要がある．差し替え後に，移送された Activity であることを意味し，本機構で提案する Action の “ACTION_MIGRATED” を指定し，Activity の実行を開始する．

4.2.4 受信したファイルの書き込み

最後に，ファイルの書き込みについては，ファイルの所有者をアプリケーションと一致させるため Activity 内部でファイル書き込みを行う．Activity の起動処理中にファイルの書き込みを行うスレッドを立ち上げる処理を追加した．

5. 評価

本章では提案基盤を実装した AndroidOS を搭載した端末を利用して，基盤を追加したことによって発生するオーバヘッドと移送処理にかかる時間を計測し，結果をまとめる．また，既存アプリケーションに手を加えることなくこの基盤が利用可能であるかを確認する．

5.1 評価方法

基盤を利用して評価用に作成したテストアプリケーションを移送する．このテストアプリケーションはスターターメソッドを利用しており，ファイルを送信する機能が利用できる．評価は次の場合について行う．

(1) Activity.java に追加した復元準備処理によるオーバヘッド
(2) 通信が確立してから Activity が起動するまでの時間
(3) 1kB から 1024kB までのファイルを 1 つ含めて移送を行った時の通信時間及び起動時間
(4) 受信側端末で復元に必要な時間

これらの各場合において，処理を 10 回行った際の結果の平均値を示す．

また，GooglePlay からダウンロードした実アプリについて移送処理が行えるか，復元は可能かも検証する．このときの評価環境を表 1 に示す．

表 1 評価環境

使用端末	PandaBoard ES
CPU	1.2 GHz ARM Cortex-A9 デュアルコア
メモリ	1 GB
OS	AndroidOS 4.4 ベースのカスタム OS
Bluetooth	Bluetooth v2.1 + EDR

5.2 測定結果

まず, 復元準備処理の実装をしたことで発生した Activity

の起動オーバヘッドの平均値は 0.7781msec となった．このオーバヘッドは基盤の利用の有無に関わらず発生するオーバヘッドとなるが，端末の利用に際して十分に小さい値である．

次に，ファイルを含めた移送処理の測定結果を表 2，3 に示す．表 2 の列は左から送信するファイルのバイト数，Migrater が読み込むために Activity が tmpfs 領域へ書き込む時間，端末間で通信を確立するまでの時間，実行状態を Bluetooth を用いて送信する時間を msec で示している．表 3 の列は左から受信するファイルのバイト数，実行状態を受信する時間，受信が完了してから Activity を起動するまでの時間，起動した Activity が状態を復元する時間，Migrater が tmpfs 領域に書き込んだファイルを Activity が読み出してストレージに書き込む時間を msec でを示している．

表 2 と表 3 より，いずれも送信するファイルサイズに比例してデータを送受信する処理時間が大幅に増加している．また，ファイルの tmpfs 領域への読み書きと Activity の起動時間もファイルサイズの増加に伴い処理時間が増加している．これは，2 つの処理がともにファイルを扱う処理であることが原因であると考えられる．一方で端末間で通信を確立するまでの時間と，復元処理を行う時間はファイルサイズによらずほぼ一定の値を示した．これらの処理はファイルに関係ない処理をしているためこのような結果になったと言える．

5.3 実アプリ検証結果

検証に利用した実アプリケーションと復元処理の成否を表 4 に示す．表中の「成功」とは Activity が起動し，復元が正しく行われたことを示し，「起動」は Activity が起動したが復元が行われなかった場合を示す．

GooglePlay で配信されているアプリケーションから，作業の引き継ぎという基盤作成の目的と合致するアプリケーションを選択し検証を行った．いずれの Activity も基盤を利用し受信側で再び起動することが確認できた．復元に成功した Activity では実験のためテキストボックスに入力した文章とそのカーソル位置の復元や選択したチェックボックスの復元された．復元が行えなかった Activity は，文章やチェックボックスの値が消えていた．ここから，内部で必要な値が復元されていないことがわかり，開発者が復元のための処理を記述していなかったと推測される．

6. 提案基盤の制限

本章では基盤を利用する上でアプリ開発者が留意する点をまとめる．まず，基盤を利用して Activity の実行状態を移送，復元を行うために次の 2 点を満たす必要がある．

(1) Activity#onSaveInstanceState() メソッドをオーバーライドし必要となる変数を Bundle に保存する
(2) Activity#onCreate() メソッドか Activity#onRestoreInstanceState() メソッドで，Bundle から変数を復元する

これらの実装を施していない Activity を移送する場合，元の Activity が受信側端末で立ち上がるだけで元の状態は全く復元されない．加えて，元の変数が存在しないことによる実行例外が発生することも考えられる．

続いて，満たすと移送後の Activity が起動しない条件を挙げる．

(1) FragmentPagerStateAdapter を利用している
(2) Bundle オブジェクトに格納しないオブジェクトを復元処理中に参照する
(3) Bundle に Parcelable オブジェクトを格納し復元時に取り出そうとする

FragmentPagerStateAdapter はディスプレイを横方向にスワイプしたときに画面が切り替わるインタフェースを作成する際に利用されるクラスである．また，Parcelable オブジェクトはあるインスタンスをバイト列へ変換する目的で利用される Java インタフェースでありこれを implement したクラスのオブジェクトを指す．これらの場合，条件 1，2 では “NullPointerException” が，条件 3 では “ClassNotFoundException” が発生し，アプリが強制終了する．これらの問題解決が今後の課題となる．

最後に，AndroidOS で利用されている SQLite について基盤では対応をしていないため動作の保証ができない．

7. 関連研究

本章では提案基盤の機構と関連のある先行研究について端末間でのアプリケーションの連携と端末間でのファイル移送機構という 2 つ観点から差異について述べる．

7.1 端末間におけるアプリケーションの連携

長原らの研究 [1] では，端末内で利用しているインテント機構を，端末の外に対して利用できるようにしている．この機構では家庭内の家電機器を対象にしている．例えば端末内に保存されている画像を表示するときに，端末にインストールされているビューワアプリを利用するのではなく，家庭内にあるデジタルフォトフレームで表示することを可能としている．しかし，送信先機器が汎用端末ではなく，家電などを対象としていることから，導入コストがかかることと，送信先機器で特定の処理しかできないという点で本提案機構と異なる．

また Hung らの研究 [2] では，クラウド環境で仮想的に Android アプリケーションを実行する環境とそのマイグレーション機構を提案している．仮想環境での実行により動作環境をユーザが自由に定義できることと，複数の端末で同じ環境にアクセスすることが可能であることが特徴である．しかし，インターネット接続環境が利用できない状

表 2 送信計測結果

サイズ (バイト)	tmp への書込 (msec)	通信確立時間 (msec)	送信処理 (msec)
1k	8.472	413.1	24.63
2k	4.672	660.5	67.79
4k	5.133	910.7	57.44
16k	7.370	938.4	187.9
32k	7.349	855.7	441.4
256k	25.68	913.8	3986
512k	32.23	753.1	7824
1024k	52.14	1026	12192

表 3 受信計測結果

サイズ (バイト)	受信処理 (msec)	起動処理 (msec)	復元処理 (msec)	tmp からの読込 (msec)
1k	22.46	6.104	1.416	10.71
2k	36.81	5.127	1.599	11.11
4k	53.55	6.009	1.346	11.93
16k	197.3	20.19	1.178	8.483
32k	363.7	27.39	1.379	11.50
256k	2990	208.7	1.688	30.35
512k	5879	273.4	1.196	41.80
1024k	11814	565.8	0.7751	67.23

表 4 検証したアプリケーションとその Activity

検証したアプリ	検証した Activity	成否
Jota Text Editor（テキストエディタ）	Editor	成功
	Setting	起動
shazam（音楽検索サービス）	MainActivity	成功
	MusicDetailsActivity	成功
twitter 公式アプリ（SNS サービス）	MainActivity	起動
	AccountSettingActivity	起動
	ProfileActivity	起動

況では仮想環境にアクセスすることができず，作業を行うことができない．このインターネット接続環境が必須であるという点で本提案機構とは異なる．

Borriello らの研究 [3] では Android 端末を用いてピアツーピアネットワークを構成するミドルウェアである Sip2Share [4] を利用して，Android アプリケーションをマイグレーションする機構を実装している．この研究の目的は，既存のアプリケーションからサービス中心のアプリケーションへの移行をサポートする基盤やツールを作成することにある．つまり，動作の主体はリモートに存在し手元の端末ではそれを利用するといったシステムを形成する．既存のアプリケーションをサービス中心へと移行するという目的が本論文と全く異なる．本論文では複数の端末間で何度も移送を行える基盤を目的としている．

7.2 容易な端末間ファイル転送

池松らの研究 [5] では複数の計算機間でコピーアンドペーストを可能にする．この研究によりスマートフォン，タブレット端末，PC 間で，画面に表示されている情報をやり取りすることが可能になる．しかし，このシステムを利用するために専用のサーバが必要になり，インターネット接続環境が利用できない場合このシステムを使うことができない．これもまたインターネット接続環境が必須であるという点で本提案機構と異なる．

Joen らの研究 [6] では複数の端末間でシームレスにファイルを送受信する基盤を作成した．この基盤は複数の端末感で Bluetooth または WiFi を用いてアドホックネットワークを形成しファイルをやり取りする．基盤は 2 つの通信方式のうち，利用可能でかつ高速なものを選択する．この基盤を利用することでファイルの共有は容易に行うことができるが，作業の引き継ぎという点では完全であるとは言えず，作業引き継ぎ効率という点で本研究と異なる．

8. まとめ

本論文では容易な端末間連携のためのアプリケーション実行状態移送基盤について述べた．本基盤はユーザが行っている処理を別端末上で引き継ぐことを目的としており，移送対象を現在起動中の Activity のみに絞った．Activity のライフサイクルを利用した実装を行い，既存アプリケーションの変更を行うことなく提案基盤を利用できることを確認した．また，作業中のファイルの転送機能を実現した．また，作成した基盤に対し時間的な評価を行い，本基盤によるオーバヘッドが軽微であることを確認した．

今後は基盤が利用できないアプリケーションについての調査を進め，より多くのアプリケーションで基盤が利用できるよう方法を検討する．

参考文献

[1] 長原裕希，伊藤孝宏，安積卓也，西尾信彦：分散インテント: 組込み機器連携を実現する Android インテントの分散拡張フレームワーク，電子情報通信学会論文誌 D，Vol. 97, No. 4, pp. 845–856 (2014).
[2] Hung, S.-H., Shih, C.-S., Shieh, J.-P., Lee, C.-P. and Huang, Y.-H.: An online migration environment for executing mobile applications on the cloud, *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, IEEE, pp. 20–27 (2011).
[3] Borriello, A., Melillo, F. and Canfora, G.: Migrating Android applications towards service-centric architectures with Sip2Share, *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, IEEE, pp. 413–416 (2013).
[4] Canfora, G. and Melillo, F.: Sip2Share-A Middleware for Mobile Peer-to-Peer Computing., *ICSOFT*, Vol. 12, pp. 445–450 (2012).
[5] 池松香，椎尾一郎：記憶の石: マルチタッチを利用したデバイス間情報移動，情報処理学会論文誌，Vol. 55, No. 4, pp. 1344–1352 (2014).
[6] Jeon, M., Kim, S.-K., Yoon, J.-H., Jo, J. and Yang, S.-B.: Short paper: Seamless file sharing for Android devices, *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pp. 189–190 (online), DOI: 10.1109/WF-IoT.2014.6803153 (2014).