平成 24 年度 修士論文

Bluetooth を用いた
個人用仮想電話網の構築

電気通信大学 大学院情報システム学研究科
情報システム基盤学専攻
0953008　酒井 慎平

指導教員　多田 好克　教授
小宮 常康　准教授
渡辺 俊典　教授

提出日　平成 25 年 1 月 24 日

目次

図目次

第 1 章

はじめに

2007 年 6 月 29 日に音楽プレイヤー iPod と携帯電話の機能を合わせ持った iPhone が、日本では 2008 年 7 月 11 日に iPhone 3G が発売された。2008 年 10 月 22 日に世界初の Android 搭載搭載携帯電話として T-Mobile G1 が、 2009 年 5 月 19 日に日本初の Android 搭載携帯電話として HT-03A が発売された。これらの携帯電話はスマートフォンと呼ばれ、これらの発売以前にあった携帯電話と比べ、一様に高性能・高機能である。

インプレス R&D の調査 [2][3] では、表 1.1 のようにスマートフォンの所持率は年々増加している。

スマートフォン利用者には携帯電話を 2 台もしくはそれ以上持つ利用者がいる。この利用形態を一般に 2 台持ちまたは複数台持ちと呼ぶ。日経 BP コンサルティングの調査 [4][5] では、少なくとも 1 台はスマートフォンを所持している 2 台持ち利用者は 2011 年 6 月時点で全体の 5.1%、2012 年 6 月時点でも全体の 5.1% 存在した。電気通信事業者協会 [6] の調査 [7] では、携帯電話の契約台数は年々増加の傾向にあり、2011 年 6 月時点では 1 億 2124 万 7 千台、2012 年 6 月時点では 1 億 2577 万 3 千台と増加している。このため、2 台持ち利用者の全体に対する比率は変わらないが、2 台持ち利用者の絶対数は増加していると言える。

1 年の推移のみで、「増加傾向にある」と推察するのは早計だが、2 台持ち利用者は少

表 1.1　スマートフォン所持率の推移 (インプレス R&D 調べ)

調査時期	スマートフォン所持率 (%)	母数 (人)
2008 年 9 月	2.6	3,178
2009 年 9 月	4.0	3,004
2010 年 9 月	9.0	2,878
2011 年 4 月	14.8	3,321
2011 年 10 月	22.9	93,468
2012 年 5 月	29.9	5,639
2012 年 10 月	39.8	85,514

なからず存在する。携帯電話を複数台持つ理由は様々だが、以下のような意見が挙げられる。

- 通話用の回線契約とデータ通信用の回線契約を別々に行った方が、スマートフォン用の通話とデータ通信をまとめた回線契約より使用料が安い。
- スマートフォンはバッテリのもちが悪く、バッテリ切れで電話に出られない場合があるので、従来の携帯電話も一緒に持っていないと安心できない。
- スマートフォンはアプリケーションの応答がなくなったり、不意に OS が再起動したりする場合が従来の携帯電話より多く、信用ができない。
- 現在契約している通信キャリアと別の通信キャリアから使いたいスマートフォンが発売されているが、現在の通信キャリアを解約したくない。
- 同じ通信キャリア同士の通話やメールの使用料にアドバンテージがあり、頻繁に通話やメールを行う相手に合わせた通信キャリアを追加で契約した方が、合計の使用料が安い。

通話用の携帯電話とデータ通信用の携帯電話を分けて所持している場合は、用途を分けることができて便利であるという意見がある反面、複数台持つのは煩わしいという意見もある。

ここで、従来の携帯電話を通話用、スマートフォンをデータ通信用として 2 台持ちの利点と欠点を挙げる。

利点として、以下の項目が挙げられる。

- 従来の携帯電話を 1 台持つより、スマートフォンを持つことで、先進的な技術の恩恵に与ることができる。
- スマートフォンを 1 台持つより、スマートフォンよりバッテリのもちが良い従来の携帯電話を持つことで、通話というメールより緊急性の高いとされることの多い連絡手段に応えることのできる時間が延びる。
- 処理を 2 台に分散することによって、1 台当たりのバッテリ消費量が減り、1 台で利用する場合よりもバッテリの駆動時間が延びることが期待できる。
- 日本国内の通信キャリアでは、通話用の回線契約とデータ通信用の回線契約を別々に行った方が、合計の使用料が安くなる場合が多い。

欠点として、以下の項目が挙げられる。

- 2 台持つことによって、1 台より所持しなければいけない重量、体積が増える。
- 操作しなければならない対象が 2 台に増える。
- 充電等、管理する手間が 2 台に増える。

ここで利用シーンを考えてみると、欠点で挙げた項目で、能動的に携帯電話を使用する場面で当てはまるものは、「操作しなければならない対象が 2 台に増える」点が一番大きいものだろう。この欠点を解消することで、複数台持ちに対するニーズが増え、携帯電話需要の増加、複数台持ちというマーケットの拡大等が見込める。本研究ではこの欠点を解消する手法を提案し、設計・実装を行う。

本論文の構成を以下に示す。第 2 章で本研究の目的を定めるための要件定義を行う。第 3 章で本研究で利用された下層技術について述べる。第 4 章で本研究のシステムを設計し、第 5 章で実装を行う。第 6 章で実験を行い、第 7 章で実験結果に対する考察を行う。第 8 章で問題点と今後の課題を述べ、第 9 章で結びの章とする。

第 2 章

目的

複数台持ちの欠点が、操作しなければならない対象が増えることであるのは第 1 章で述べた。この欠点を本研究の提起する問題とする。

本研究の提起する問題の解決手法を考えるために、携帯電話の操作について、利用者にどのような操作が求められるかを挙げる。

2.1 利用者に求められる操作

携帯電話を操作する際、利用者に求められる操作の流れを書き出す。通話時とデータ通信時の操作で分けて考える。

(*) がついている操作は、利用者の意思が介在しうる操作である。

2.1.1 通話時の携帯電話の操作

通話の切っ掛けとして、発信と着信がある。発信または着信が通話に繋がり、通話の切断によって操作は終了する。

発信時の操作

1. 電話番号を入力する。又は、電話帳から電話番号を選択する。(*)
2. オフフックボタンを押す。
3. スピーカを耳元に、マイクを口元に近付ける。
4. 電話が繋ったら会話を始める。

着信時の操作

1. 着信の合図を確認し、オフフックボタンを押す。
2. スピーカを耳元に、マイクを口元に近付ける。
3. 電話が繋ったら会話を始める。

通話時の操作

1. 相手の会話に合わせて会話を行う。(*)

切断時の操作

1. 耳元から携帯電話を遠ざける。
2. 相手から切断されていなければ、オンフックボタンを押す。

通話操作全体の流れ

以上を踏まえて通話操作全体の流れを図にすると、図 2.1 のようになる。

2.1.2 データ通信時の携帯電話の操作

データ通信を伴う操作は、基本的に何らかアプリケーションを実行することで発生する。

新規メールの作成、乗換案内アプリでの経路検索を例として挙げ、一般的なアプリケーションの操作としてまとめる。

新規メールを送信する時の操作

1. メーラを立ち上げる。
2. 新規メールの送信を選択する。
3. 宛先、件名、本文の入力欄が表示される。
4. 宛先にメールを送りたい人を選択する。(*)
5. 送信したいメールの件名を入力する。(*)
6. 送信したいメールの本文を入力する。(*)
7. 送信ボタンを押す。
8. 送信が完了したらメーラを閉じる。

乗換案内アプリの操作

1. 乗換案内アプリを立ち上げる。
2. 出発地を入力する。(*)
3. 目的地を入力する。(*)
4. 目的に合わせた時刻設定を行う。(*)

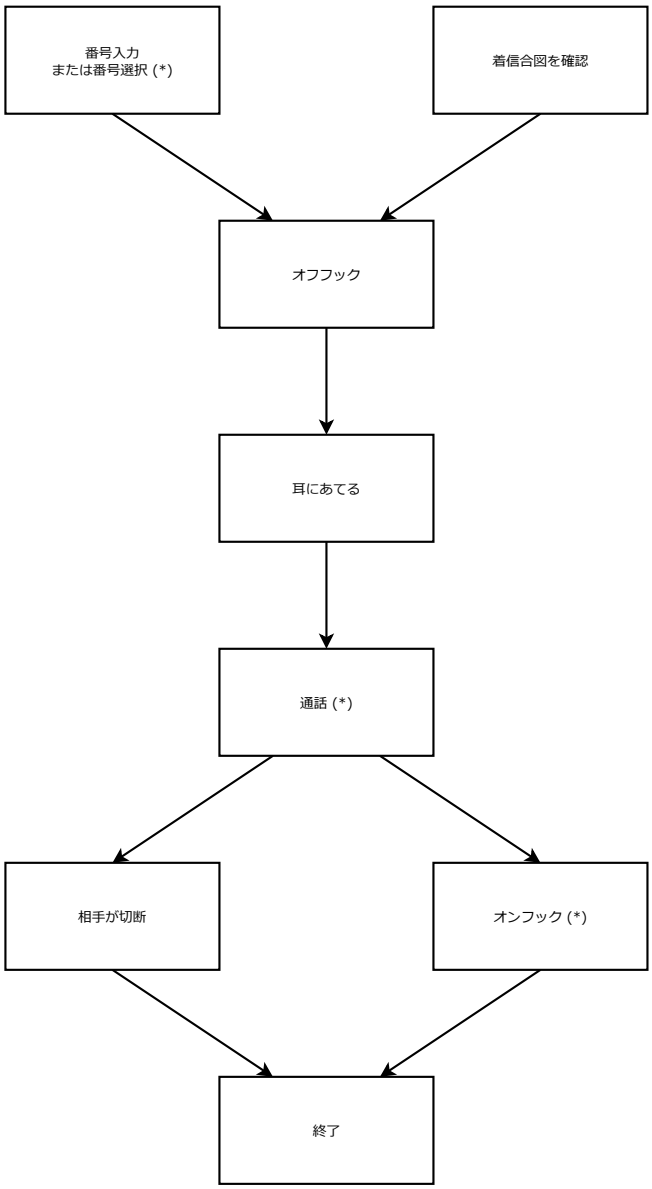

図 2.1　通話操作全体の流れ

5. 経路検索ボタンを押す。
6. 結果が表示される。
7. 結果を理解したら乗換案内アプリを閉じる。

一般的な操作

1. 利用したいアプリケーション (メーラ、ブラウザ、乗換案内等) を立ち上げる。(*)
2. 出力された画面表示に従って入力を行う。(*)

3. 利用目的が達成したら、アプリケーションを終了する。(*)

データ通信操作全体の流れ

以上を踏まえ、さらにブラウザの操作の例を加えて図にすると、図 2.2 のようになる。

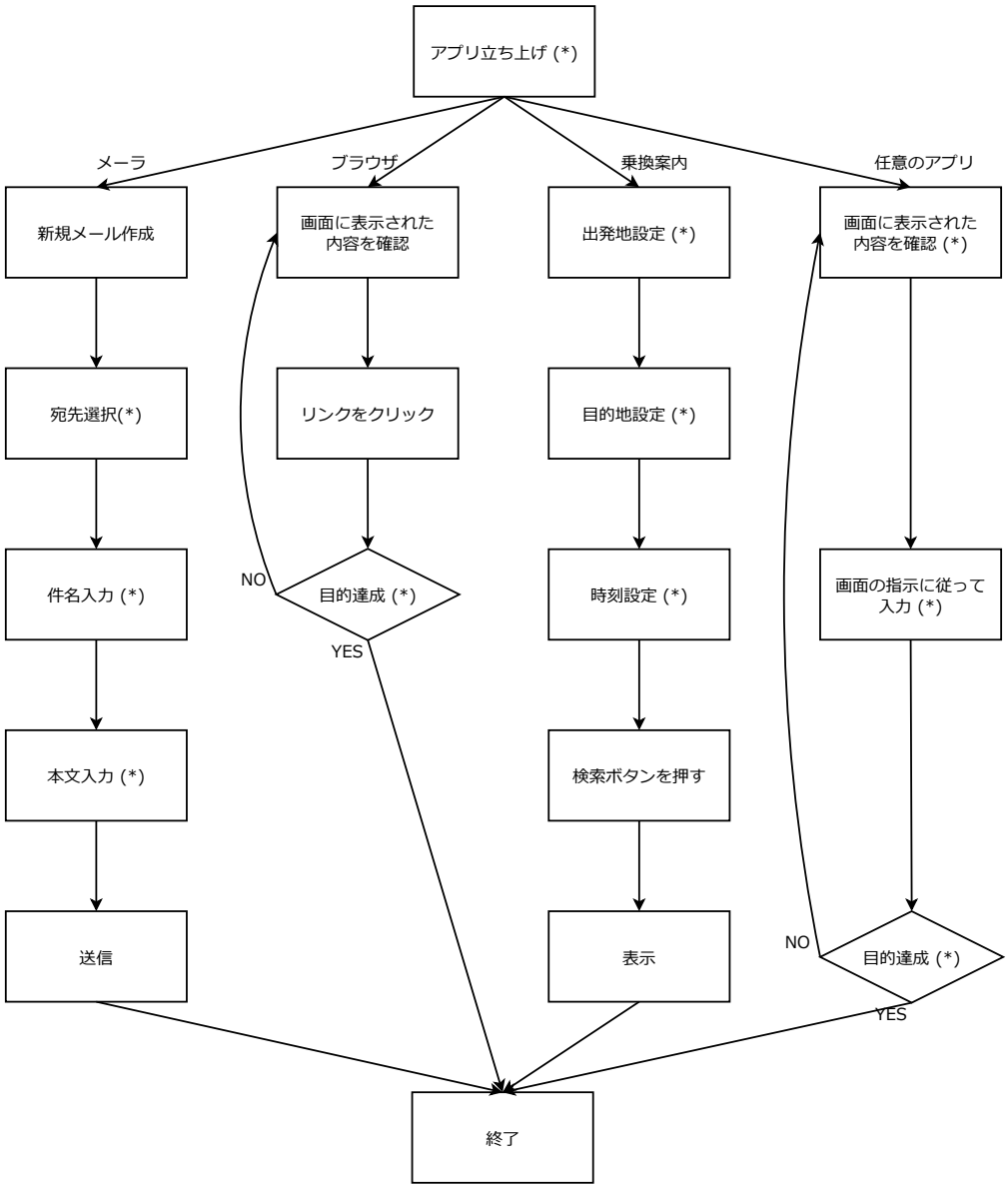

図 2.2　データ通信操作全体の流れ

2.2　問題の解決手法

本研究で提起された問題を解決する手法として、以下の 3 つの手法を考えた。

1. 利用者が操作を行わなくても、自動的に操作が行われるようなシステムを用意し、利用者の負担を減らす。
2. 利用者が直接実行する汎用的な操作を取り決め、携帯電話が何台になろうとも 1 つの操作で複数台の携帯電話を操作できるようにし、利用者の負担を減らす。
3. 統合的に操作できる機器を用意し、1 つの機器で複数の携帯電話を操作することで、利用者の負担を減らす。

1 について、利用者の意思が介在する操作が行われる部分を自動化することは難しい。例えば、通話の流れにある接続相手との通話の部分について、利用者の状況に応じた定型文を返答とすることは可能だが、話したいことをそのまま返答とすることは不可能である。また、データ通信の操作においても、どのようなアプリケーションを立ち上げたいかという流れの開始地点において、本人の意思が介在している。これらを自動化する場合は、思考を読み取るシステムが必要となり、非現実的である。したがって、携帯電話の操作を完全に自動化することは現実的ではない。

2 について、利用者と携帯電話の間に仮想的なネットワークを生成し、そのネットワークに操作をブロードキャストすることによって、携帯電話を個別に操作せずに済むようなシステムを構築すればよい。人間がブロードキャスト可能なインターフェースとして、音声が挙げられる。利用者の発した音声を携帯電話のマイクで受信し、音声認識によって操作を実行すれば実現可能である。

ただし、この手法で発着信を行った場合、通話するために結局対象の携帯電話を手に取り、耳にあてなければならない。さもなければ、携帯電話を耳にあてなくても聞こえる音量で相手の音声をスピーカから出力しなくてはならない。これでは通話の内容が第三者に筒抜けであり、使い所は非常に限られるだろう。筒抜けになることを覚悟で通話をしたとして、通話を切断するときに、切断するためのキーワードを携帯電話に向かって話さなければならないだろう。その通話切断用のキーワードが意図しないタイミングで会話の中で発生したとしたら、通話が切れてしまう。音声認識と通話は相性が悪く、何か補助が必要である。無線のヘッドセットを接続しておけば、いくつかの不都合は解決するだろうが、それはもはや 2 単体の条件を満たしていない。

3 について、遠隔操作の行える機器やソフトウェアなどのインターフェースを利用することにより、1 台の機器で複数台の機器を操作することが考えられる。例えば複数台の携帯電話から待ち受けができるハンドセットを接続すれば、同時に着信があった場合を除いて、通話の問題は解決するだろう。ただ、操作する携帯電話を減らすために、新しい機器を追加することは本末転倒に思える。そこで、性能や拡張性の高いスマートフォンにソフトウェアとして遠隔操作用のインターフェースを実装することによって、所持する機器を増やすことなく操作する機器を減らすことができる。本研究はこの方向性で問題の解決を

試みる。

ここで、前述の携帯電話の操作を比較すると、通話操作の方がデータ通信時の操作と比べてループがなく、また意思による判断・決定の幅が狭い。そのため、規定の手順を作り易い。

よって実現可能性を考慮し、本研究では遠隔操作の対象をデータ通信の操作ではなく通話操作に絞る。これは実際の利用形態にも則している。

複数台持つ機器の用途の内訳として、データ通信用携帯電話より、通話用の携帯電話を多く持つケースが見受けられる。これは以下の理由等からデータ通信用携帯電話を複数台持つ必要性が少ないからである。

- 各通信キャリアの通話使用料に比べ、本研究時点でのデータ通信の使用料にはあまり差異がない。
- 各キャリア専用サイトや従来の携帯電話専用のサイト、スマートフォン専用のサイトへのアクセスではない、一般的なインターネットアクセスを行った場合、内容に差異が出ない。

逆に、通話料金制度は、同キャリア同士であれば通話料が無料になる等各キャリアに付加価値があり、通話を行う連絡相手によっては複数のキャリアと契約することも有効な選択肢の一つとなる。

通話を遠隔操作するためのハンドセットの接続規格として考えられるものとして、以下のものが挙げられる。

- イヤホン・マイク用コネクタ
- データ転送用コネクタ
- Bluetooth
- Wi-Fi

このうち、イヤホン・マイク用コネクタとデータ転送用コネクタは、一対多の接続形式を取ることができないので、複数台の携帯電話の待ち受けを行うという目的に合致しない。

残る Bluetooth と Wi-Fi は、スマートフォンへの搭載率は高いが、従来の携帯電話への搭載率はスマートフォンほどではない。しかし、Bluetooth を搭載している携帯電話の多くは、ヘッドセットプロファイルまたはハンズフリープロファイルを実装している。これを利用すると、ハンドセットの実装のみで複数台の携帯電話の待ち受けという目的を達成することができる。Wi-Fi を選択した場合、通話用の携帯電話と、ハンドセットとなる通信用の携帯電話のどちらにもソフトウェアを実装しなければならない。通話用の携帯電

話が従来の携帯電話であった場合、Wi-Fi で通話制御を行うソフトウェアを作成することはできず、アプリケーションとして導入することができないため、Bluetooth を利用する方が適用性が高い。

これらのことから、本研究では Bluetooth 搭載携帯電話に対象を絞り、Bluetooth ハンドセット機能を使用して複数台の携帯電話と接続し、操作する携帯電話の数を減らすことを目的とする。また、本研究ではハンドセット機能を実現するために、ハンズフリープロファイルを実装する。

実装対象のプラットフォームとして、スマートフォン向けのプラットフォームである Android を選択する。対抗するプラットフォームとして、従来の携帯電話や、スマートフォンである iPhone が挙げられるが、これらのプラットフォームは Bluetooth を制御できる API を公開している範囲が狭く、本研究の提案手法に含まれるハンズフリープロファイルの実装が不可能と思われるからである。

Bluetooth ハンズフリープロファイルを実装するための要求として、以下の項目を挙げる。

1. ***Bluetooth* を搭載していること。**
   Bluetooth を搭載していないと、Bluetooth を用いた通信はできないので必須である。
2. ***Bluetooth* の制御ができること。**
   Bluetooth を搭載していても、制御ができなければ意図した通信を行うことはできない。
   また、研究に関する部分を実装することに注力するため、Bluetooth 制御以外は手を加えなくても良いことが望ましい。
3. **受話器の操作に相当する処理ができる入力インターフェースがあること。**
   着信に対してのオフフック、発信に対してのダイヤルとオフフック、通話に対してのオンフックの操作ができるインターフェースは、ハンドセットとして必須である。
4. **通話のためのスピーカとマイクがあること。**
   通話の際に使用できるスピーカとマイクは、ハンドセットとして必須である。
5. ***(*オプション*)* 電話番号を表示できる出力インターフェースがあること。**
   着信時に相手の電話番号を表示できる出力インターフェースがあるとなお良い。

これに対して、Android 搭載スマートフォンをプラットフォームとすることで、ほとんどの要求を満たすことができる。

1. Bluetooth を搭載している端末が多い。
2. API が公開されており、Bluetooth を制御するための API も含まれているので、アプリケーションを作ることで制御ができる。
3. 受話器の操作に相当するボタンを設置することで、タッチパネルで操作できる。
4. もともと電話機であり、スピーカーもマイクもある。
5. ディスプレイがあり、電話番号の表示ができる。

自分でプロファイルを実装することによって、本研究の応用として以下のようなことが考えられる。

- 任意のタイミングで音声接続を行い、電話がかかってきていない状態でもハンドセットと携帯電話でトランシーバーのように扱える。
- 待ち受けしている携帯電話 2 台以上と同時に接続し、互いの接続をさらに接続することによって、3 者以上の通話ができるようになる。

第 3 章

関連技術

3.1 本研究で使用した下層技術

本研究では、性能と拡張性の高いスマートフォンに遠隔操作アプリケーションを実装する。接続規格として Bluetooth を用い、Bluetooth のプロファイルの一つであるハンズフリープロファイルで実際の接続を行う。アプリケーションを実装するスマートフォンのターゲットプラットフォームとして Android を選択した。

Hands-Free Profile 1.5 仕様書 [8] の要求により、通話の制御には AT コマンドが使用される。

3.1.1 Bluetooth

Bluetooth とは、Bluetooth SIG[*1][1] の策定する無線技術であり、携帯機器や固定機器を接続するケーブルを置き換えることを目的としている。本研究では電話回線を通した通話を行う携帯電話と、ハンドセットとなるスマートフォンを繋ぐケーブルの代用として Bluetooth が使われる。

Bluetooth 機器同士の接続は、マスターとスレーブの役割に分かれ、ピコネットという接続形態を形成する。ピコネット内において、マスター 1 台に対してスレーブは 7 台まで同時に接続することができる。それによって、本研究の要素の 1 つである、ハンドセットに対する同時接続が実現される。

現在 Bluetooth は様々な機器に使われている。パソコンやヘッドフォン、携帯電話などへ採用されているだけでなく、医療機器やゲームコントローラー、フィットネス関連製品など、幅広く採用されている。Bluetooth SIG の調査では、Bluetooth 搭載端末の出荷状況は 90 億台とされ、2012 年だけで 20 億台以上の Bluetooth 搭載機器が出荷された [9]。Bluetooth の携帯電話への搭載率は、2011 年で 73% であり、2016 年には 90% に達すると予想されている [10]。

Bluetooth コアシステムは、無線送受信機、バースバンドおよびプロトコルスタックで

[*1] Bluetooth Special Interest Group

構成される。また、Bluetooth コアシステムはいくつかの層に分かれるが、大きく分けると Bluetooth コントローラと Bluetooth ホストに分かれ、その 2 つを繋ぐホスト・コントローラ間インターフェース (HCI[*2]) によって接続される。

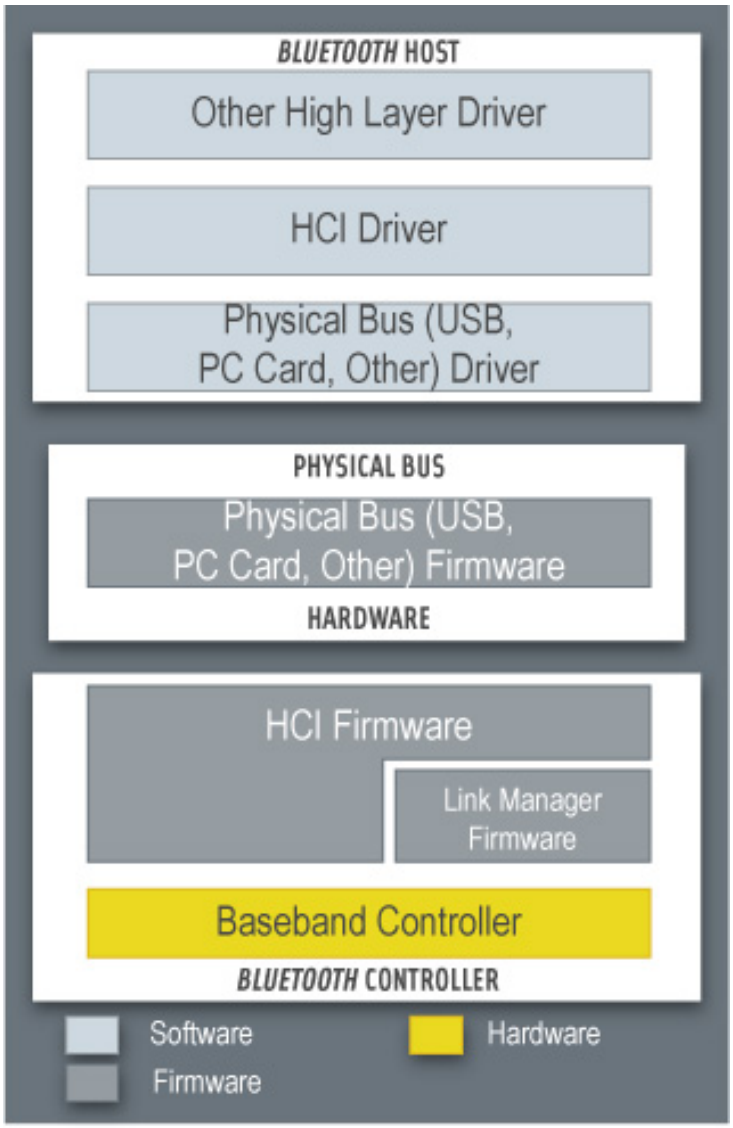

図 3.1　Bluetooth ソフトウェアスタックの下位層 ([1] より)

図 3.1 は、Bluetooth ソフトウェアスタックの下位ソフトウェア層の概観である。

Bluetooth コントローラは、ハードウェア寄りである 3 つの下位層のリンクマネージャ層、ベースバンド層、無線層と、HCI ファームウェアを指す。

Bluetooth ホストは、HCI ドライバと、物理バス[*3]ドライバ、それ以外の高レベル層の

[*2] host to controller interface
[*3] USB や PC カード等

ドライバ、L2CAP や各プロファイルを指す。

Bluetooth HCI は、Bluetooth コントローラに HCI ファームウェア、Bluetooth ホストに HCI ドライバとして実装され、Bluetooth ホストから Bluetooth コントローラに、またその逆にアクセスするための統一的な手段を提供する。

プロファイルの実装には、主に Bluetooth ホストの技術を利用することになる。その中でも本研究に関係の深い技術として、コントローラからデータトランスポート、ホストからプロトコルとプロファイルについて取り上げる。

データトランスポート

Bluetooth 機器同士でデータトランスポートを行う場合、非同期通信と同期通信のどちらかを選択することができる。それぞれに異なる論理トランスポート、論理リンクが提供される。

非同期接続型 (ACL) **論理トランスポート**　ACL[*4]は、非同期でデータを送受信するための論理トランスポートである。主にデータ通信に使われる。

ACL は、LMP 制御信号と L2CAP 制御信号、およびベストエフォート型ユーザーデータの送信に使われる。LMP 制御には、論理トランスポートと論理リンクの設定および制御、並びに物理リンクの制御が含まれる。L2CAP は、上位レベルプロトコル多重化、パケット分割と再組み立て、およびサービス品質情報の伝達をサポートする。

本研究で使用するのはユーザーデータ送信の部分である。

ACL は非同期 (ACL-U[*5]) 論理リンクをサポートする。ACL-U は、すべての非同期およびアイソクロナスフレームユーザーデータの送信に使われる。

アプリケーションは、ACL-U に可変サイズフレームで (最大でチャネルに取り決められた最大値まで) データを送信し、送信されたフレームはリモート機器上の対応するアプリケーションに同じ形式で配信される。

同期接続型 (SCO) **論理トランスポート**　SCO[*6]は、同期的にデータを送受信するための論理トランスポートである。主に音声通信に使われる。

SCO 論理トランスポートは、ピコネット上のクロックに同期する 64kb/s の情報を送信する。通常この情報は符号化音声ストリームである。8KHz のサンプリングレートで 8bit の符号化をされた音声データであれば、64kb/s のビットレートとなる。

[*4] Asynchronous Connection-Oriented (ACL) logical transport

[*5] ACL User data

[*6] Synchronous Connection-Oriented (SCO) logical transport

SCO は同期 (SCO-S[*7]) 論理リンクをサポートする。SCO-S はフレームなしのストリームで配信されるアイソクロナスデータの送信に使われる。

Bluetooth コアシステムは、SCO-S 論理リンクを使って、アイソクロナスで一定速度のアプリケーションデータの直接トランスポートをサポートする。SCO-S 論理リンクは、物理チャネル帯域幅を予約し、ピコネットクロックを固定する定速トランスポートを提供する。データは定期的に固定サイズパケットでトランスポートされる。

プロトコル

Bluetooth コア仕様では、Bluetooth コアシステム内部での全てのシグナルのやりとりにはプロトコルが定められている。

Bluetooth コア仕様で定められたプロトコルの中で下位層のものが、無線 (RF) プロトコル、リンク制御 (LC) プロトコル、リンク管理 (RM) プロトコル、論理リンク制御適応プロトコル (L2CAP) である。また、サービス層プロトコルとして、サービス発見プロトコルがある。

コア仕様に含まれないプロトコルとしては、RFCOMM、AVCTP、AVDTP 等がある。

この中で、サービス発見プロトコルと RFCOMM について取り上げる。

サービス発見プロトコル (SDP)

SDP[*8]は、機器がどのようなサービスを提供しているかを通知するためのプロトコルである。SDP は全ての Bluetooth アプリケーションに必須である。

Bluetooth 機器は用途によって提供しているサービスが違う。接続するリモート機器がどのようなサービスを提供しているかは、リモート機器の SDP サーバにサービス問合せを送信し、ローカル機器の SDP クライアントがサービス問合せ応答を受信する。本研究を例にとると、通話用の携帯電話がハンドセットの役割となるスマートフォンにサービス問合せを送信し、スマートフォンはハンズフリープロファイルを提供していると応答する。その応答を受信することで、接続相手がハンズフリープロファイルを提供していることが分かる。

このような仕組みで SDP を読み取り、サービスを発見するプロファイルがサービス発見アプリケーションプロファイル (SDAP[*9]) である。

[*7] SCO Stream data

[*8] Service discovery protocol

[*9] Service Discovery Application Profile

RFCOMM

RFCOMM は、シリアルケーブル回線設定と RS-232 シリアルポートの状態をエミュレートし、シリアルデータを転送するためのプロトコルである。

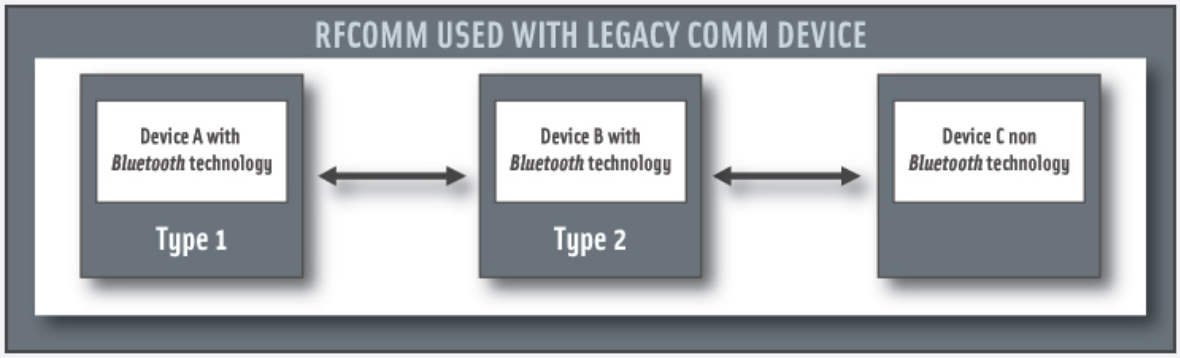

図 3.2　RFCOMM を使用したレガシー COMM デバイスとの接続 ([1] より)

シリアルポートエミュレーションを提供することで、RFCOMM はレガシーシリアルポートアプリケーションをサポートする。図 3.2 に示すように、片側は Bluetooth 無線技術を通じて通信して、他の側で有線インターフェースを備えるモジュールなど、他の構成をサポートできる。デバイス C がモデムであれば、デバイス A からダイヤルアップすることができる。本研究の場合、デバイス A がハンドセット役のスマートフォン、デバイス C が通話用の携帯電話であり、ハンドセットが AT コマンドを送ることで、ダイヤルや通話への応答を行う。

プロファイル

プロファイルとは、Bluetooth コアシステムや、Bluetooth コアシステムを利用して作られたソフトウェアスタックを使用し、Bluetooth 機器が他の Bluetooth 機器と通信するための動作規定である。Bluetooth 機器として振舞うには、何らかの Bluetooth プロファイルを解釈できなければならない。

Bluetooth プロファイルには、広範な利用事例のため、様々なものがある。その中でも、ヘッドセットプロファイルと、本研究の実装目標の 1 つであるハンズフリープロファイルを取り上げる。

ヘッドセットプロファイル (HSP)　HSP[*10]は、コンピューターや携帯電話のようなBluetooth 機器と、Bluetooth ヘッドセットがどのように通信すべきかを規定するものである。

代表的な利用シーンとして、携帯電話と Bluetooth ヘッドセットを接続しての通話が挙げられる。

HSP はオーディオゲートウェイ (AG[*11]) とヘッドセット (HS[*12]) の 2 つの役割を定義する。

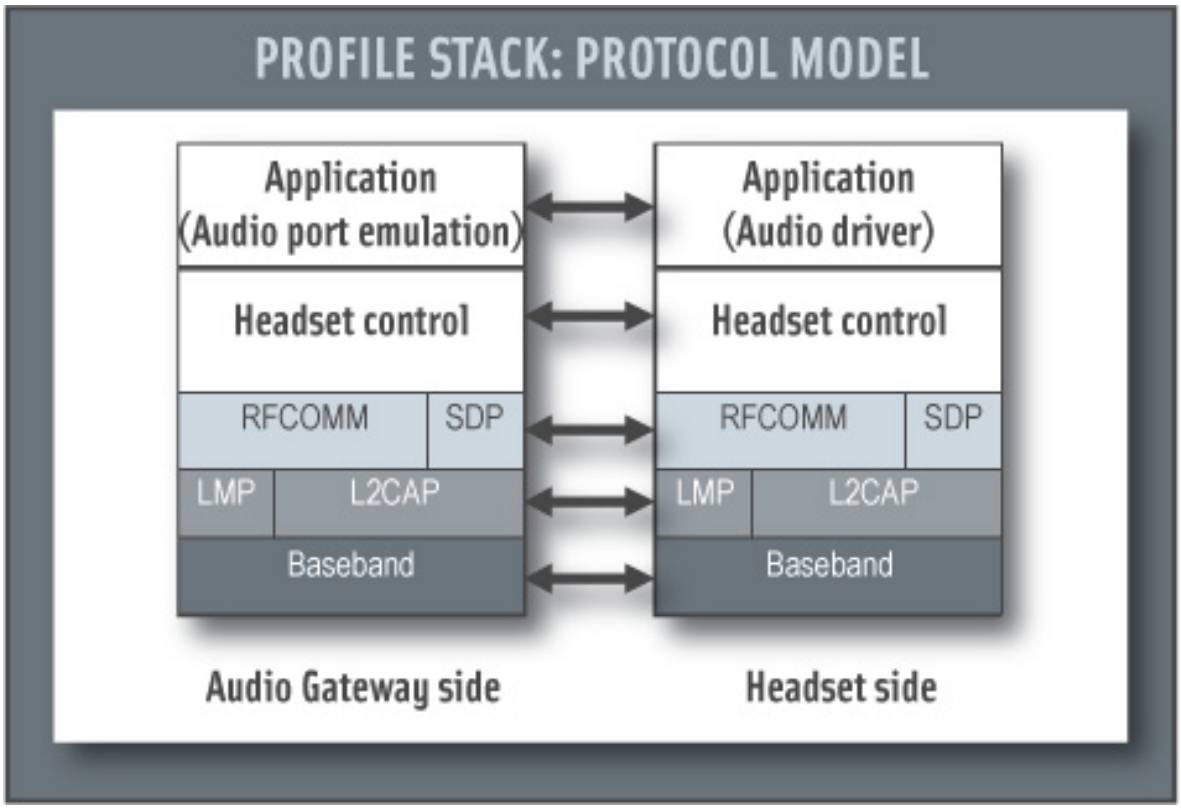

図 3.3　HSP スタックのプロトコルモデル ([1] より)

図 3.3 は HSP スタックのプロトコルモデルである。SDP を利用してサービス発見を行い、RFCOMM で互いを接続する。機器の制御は RFCOMM 上で送受信する AT コマンドで行われる。このモデルには示されていないが、音声の送受信のために SCO リンクが要求される。

HSP は着信はできるが発信はできない。そのため本研究の実装としては向かない。

[*10] Headset Profile
[*11] Audio Gateway
[*12] Headset

ハンズフリープロファイル (HFP)　HFP[*13]は、ハンズフリー機器で電話を発着信するために、どのようにゲートウェイ機器が使われるべきかを規定するものである。

代表的な利用シーンとして、携帯電話とカーオーディオを接続し、車の運転中に手を塞がずに通話できる、車載のハンズフリーキットが挙げられる。また、携帯電話とBluetooth ヘッドセットを接続しての発着信、通話が挙げられる。

HFP はオーディオゲートウェイ (AG) とハンズフリー装置 (HF[*14]) の 2 つの役割を定義する。

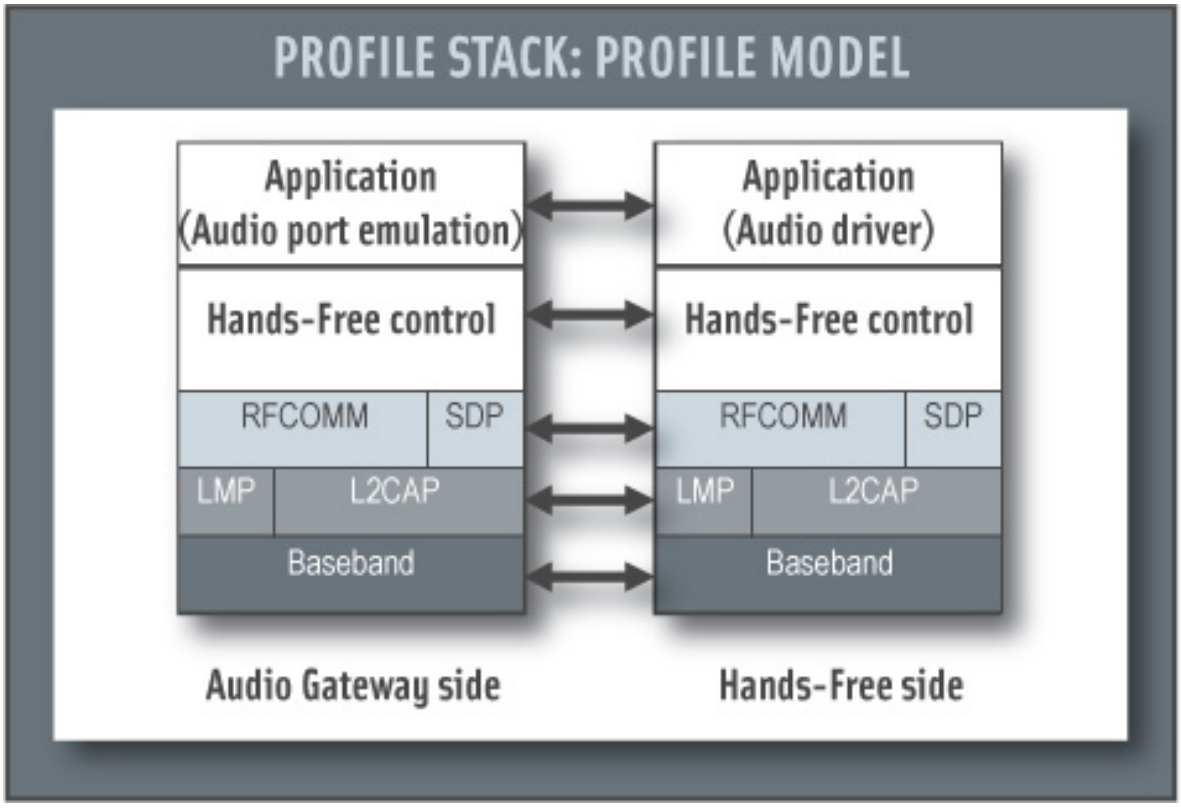

図 3.4　HFP スタックのプロトコルモデル ([1] より)

図 3.3 は HFP スタックのプロトコルモデルである。SDP を利用してサービス発見を行い、RFCOMM で互いを接続する。機器の制御は RFCOMM 上で送受信する AT コマンドで行われる。このモデルには示されていないが、音声の送受信のために SCO リンクが要求される。

HFP は AG の発着信を制御できる。本研究ではこのプロファイルの HF の役割を実装

[*13] Hands-Free Profile
[*14] Hands-Free

すること目指す。

3.1.2 AT コマンド

AT コマンドは Hayes Computer Products が自社のモデムを制御するために策定したコマンド群の通称である。ヘイズ AT コマンドやヘイズコマンドセットとも呼ばれる。

AT から始まる文字列で構成される「コマンド」を送出し、「リザルトコード」を受け取るのが基本的な形である。この AT という文字列から AT コマンドと呼ばれる。AT は attention の略である。

AT コマンドは TIA[*15]によって、TIA/EIA-602[11][12] として基本部分が標準化されている。各ベンダーは TIA/EIA-602 を独自に拡張して利用している場合が多い。Bluetooth HFP で使われる AT コマンドも拡張されており、Bluetooth 固有のコマンドは AT+B で始まる。

3.1.3 Android

Android とは、Google が開発する携帯情報端末向けプラットフォームである。Linux ベースのモバイル用オープンソースオペレーティングシステム、ミドルウェア、主要なアプリケーションからなるソフトウェアスタックパッケージで構成されている。

Google は Android SDK[*16]としてアプリケーション開発者向けの開発キットを提供しており、その中に Bluetooth を制御するための Bluetooth パッケージという API が用意されている。そのため、Bluetooth の制御は比較的容易に行うことができる。

また、Android の Bluetooth 制御の実体は、Linux 標準の Bluetooth スタックである BlueZ である。BlueZ には HFP FP を実装するためのパッチが作成されており、Android SDK の API から制御することが不可能だったとしても、BlueZ のライブラリを入れ替える等の実装方法が考えられる。

そして、Android 搭載端末は Bluetooth の搭載率が高く、日本国内で販売されている Android 搭載端末はほぼ全て Bluetooth を搭載している。

これらのことから、本研究のプラットフォームとして適していると言える。

[*15] Telecommunications Industry Association
[*16] Software Development Kit

3.2 本研究と関連する製品

3.2.1 WILLCOM WX01S SOCIUS

SOCIUS[13] は、2011 年 10 月 6 日に WILLCOM より発売された [14]、セイコーインスツル製の PHS 端末である。

携帯電話に Bluetooth が搭載され、HFP AG の機能を持ったスタックが搭載されることは珍しいことではなくなった。SOCIUS は、これまで携帯電話に搭載されることのなかった HFP HF の機能を持ったスタックを搭載することで、SOCIUS 自身で電話の発着信ができるだけでなく、Bluetooth で他の携帯電話に接続し、他の携帯電話で電話の発着信ができる、業界初[*17]の Bluetooth ハンドセット機能搭載 PHS 端末である。

その後も、少なくとも日本市場では、Bluetooth ハンドセット機能搭載携帯電話または PHS 端末は販売されていない。

ハンドセット機能を持った PHS 端末ということで、本研究の目的と非常に合致しており、SOCIUS と通話用の携帯電話の組合せで解決するように思われる。しかし、SOCIUS は従来の携帯電話と同等の PHS 端末であり、SOCIUS と組合せる場合、スマートフォンの技術的恩恵を与ることができなくなってしまう。また、SOCIUS にはソフトウェアの拡張性がなく、本研究の発展的な目的である、複数台の Bluetooth 通信網による仮想電話網を構築することはできないと思われる。

スマートフォンのような先進的な技術を持った端末を拡張し、ハンドセットとして機能させることが、あくまで本研究の目的である。

3.2.2 一般的な Bluetooth ハンドセット・ヘッドセット

携帯電話に搭載されているのではない、単機能の Bluetooth ハンドセット・ヘッドセットというものも当然存在する。携帯電話と同様に電話帳保存機能を備え、電話帳から番号を選んで発信、着信した電話番号を電話帳から引いて登録名を表示等の機能があるものもある [15][16]。しかし、複数台持ちにハンドセットを加えると機器の管理の煩わしさが増す、Bluetooth デバイス同士で通話できるように拡張することができない等、本研究の目的には似わない。

[*17] 2011 年 9 月 21 日現在、国内の主要携帯電話機市場において、セイコーインスツル株式会社調べ。

第 4 章

システムの設計

Android 上に Bluetooth HFP HF を実装するための設計を行う。
以下が本研究の目的と照らし合わせた、ハンドセットとしての最低限の機能である。

1. Bluetooth HFP AG と接続し、待ち受けができる。
2. 電話を発信できる。
3. 電話を着信できる。
4. 通話できる。
5. 通話を切れる。
6. 複数台の待ち受けができる。

上記を実装した上で、追加機能として実装する項目を以下とする。

7. 着信した電話番号を表示できる。
8. 保留ができる。
9. 着信拒否ができる。
10. リダイヤルができる。
11. ボイスダイヤルができる。
12. 電話番号から電話帳を引いて表示できる。

これらの機能が要求する仕様を考える。
1〜6 の全てにおいて、Hands-Free Profile 1.5 仕様書 [8] に準拠した接続、データ転送が要求される。
2 においてはダイヤルと発信、3 においては着信への応答、5 においては通話の切断のためのイベントを発生させる GUI が要求される。

4.1 HFP HF が使用する技術

HFP は RFCOMM で接続され、AT コマンドで制御を行い、SCO-S リンクを利用した音声の転送が行われる。

4.1.1 サービスレコードの登録

HFP HF がサービスとして登録されていないと、HFP AG はその機能を利用することができない。HFP HF としてサービスレコードを登録する必要がある。

4.1.2 RFCOMM 接続

HFP が行う通信は、機器制御のデータと音声の転送がある。機器制御のデータは RFCOMM 接続を通して行われる。そのため、RFCOMM 接続を確立させる手続きが必要となる。

AT コマンド

AG – HF 間の制御は AT コマンドで行われる。HF が AT コマンドを発行し、AG がリザルトコードを返す。そのため、AT コマンドのジェネレータと、リザルトコードのパーサが必要となる。

4.1.3 音声の転送

ハンドセットとしての音声の正しい経路は、AG のスピーカを HF のスピーカに接続し、HF のマイクを AG のマイクに接続するような経路となる。この HF と AG の間の転送方法は、SCO-S リンクを用いたストリーム転送となる。

HF と AG、そして AG に対する通話相手への音声の流れを図にすると、図 4.1 のようになる。通常のオーディオ入出力から SCO-S リンクへ接続を切り替える必要がある。

4.2 GUI の設計

本研究の要求を満たす GUI として、図 4.2 の構成と遷移を示す。

4.2.1 画面構成

接続画面、ダイヤル画面、通話画面の 3 つの画面で構成される。

接続画面

デフォルトの画面である。最初に表示される。

接続可能な Bluetooth 機器リストから AG を選択し、接続を行う。または、AG からの接続待ちを行う。

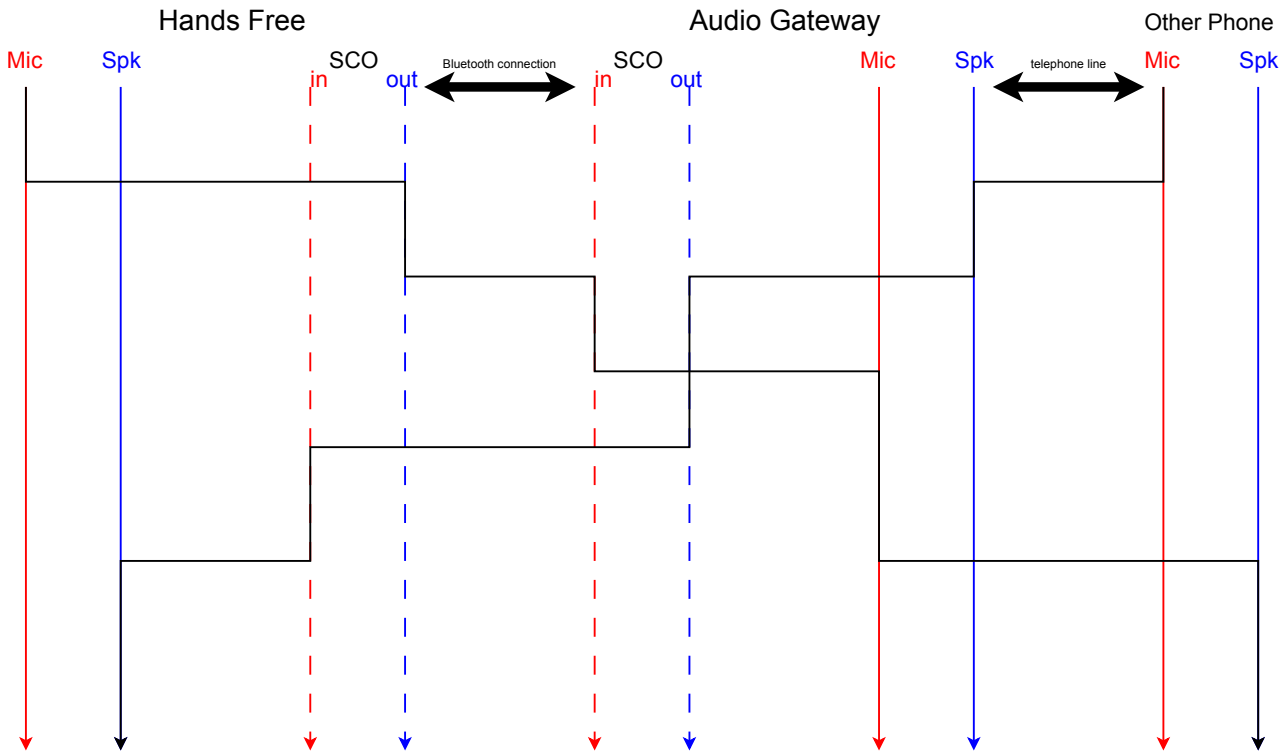

図 4.1　HF、AG、AG の電話相手の間の音声の流れ

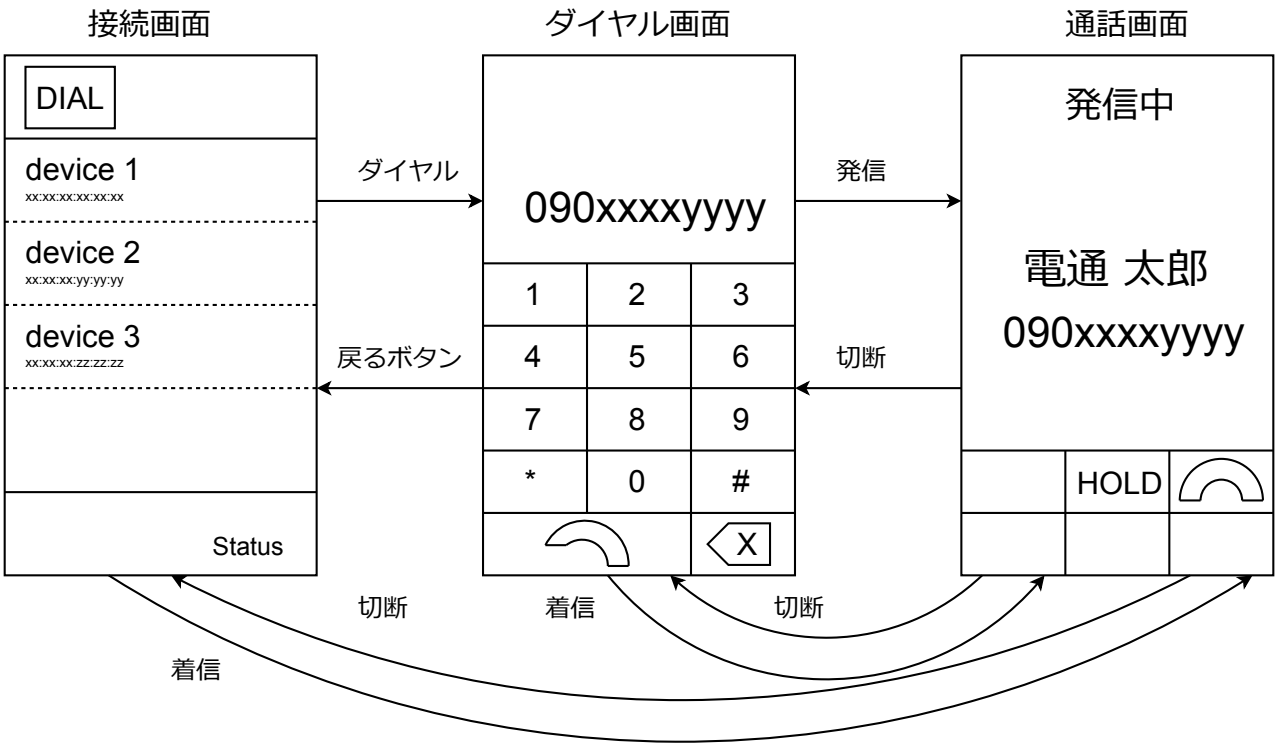

図 4.2　GUI の構成と遷移

AG との接続後にダイヤルができるよう、ダイヤルボタンを設置する。

ダイヤル画面

ダイヤル時に使用する。ダイヤル用のテンキーと一字削除ボタン、発信用のオフフックボタンを備える。テンキーで入力した番号はテンキーの上部に表示される。オフフックボタンを押すことで、表示されている番号に発信することができる。

通話画面

発信、着信、通話時に使用する。状況に応じて画面上部のメッセージが「発信中」「着信中」「通話中」に変化する。

発信、着信、通話の相手の番号と、電話帳に番号が保存されていればその名前を画面中央に表示する。発着信、通話制御のために、オンフックボタン、保留ボタン、着信拒否ボタン等を画面下部に備える。

4.2.2 画面遷移

接続画面からダイヤルボタンを押すことでダイヤル画面に遷移し、ダイヤル画面から Android 標準の戻るボタンを押すことで接続画面に遷移する。

ダイヤル画面から発信ボタンを押すことで通話画面に遷移し、通話画面で通話が切断されることでダイヤル画面に遷移する。

接続画面、ダイヤル画面どちらからも着信を受けることで通話画面に遷移し、通話画面で通話が切断されることで元の画面に遷移する。

4.3 本研究で実装するもの

本研究で実装すべきものは以下の 6 項目となる。

- サービスレコードの登録。
- RFCOMM 接続。
- AT コマンドジェネレータ。
- SCO-S リンクの確立。
- SCO-S リンクとオーディオとの接続。
- GUI の作成。

第 5 章

システムの実装

5.1 Android SDK の API レベルの選定

Android の最新バージョンは Jelly Bean(バージョン 4.2、API レベル 17) だが、Google の調査 [17] によると、2013 年 1 月現在、Gingerbread(バージョン 2.3.7、API レベル 10) 以下の端末が過半数である。Honeycomb(バージョン 3.0、API レベル 11) から BluetoothProfile インターフェースが追加され、本研究でも有用であるかと思われたが、BluetoothProfile のサブクラスである BluetoothHeadset は AG の実装であり、BluetoothProfile インターフェースから HF の実装を行うのは難しいと感じたため、利用する API レベルを 10 以下に落として実装を行った。実際に利用した API レベルは 8 である。

5.2 RFCOMM の作成とサービスレコードの登録

listenUsingRfcommWithServiceRecord(String name, UUID uuid) を利用することによって、RFCOMM のサーバーソケットを作成するとともに、SDP のレコードも登録される。uuid に 0000111E-0000-1000-8000-00805F9B34FB を指定すると、HFP HF としてのサービスを提供していることになる。サーバーソケットなので、accept() を実行して AG の接続を待つことになる。

HF の方から接続する場合は createRfcommSocketToServiceRecord(UUID uuid) を利用し、RFCOMM のソケットを作成する。こちらはクライアントソケットなので、connect() を実行することで、事前に指定した機器に対し uuid で示されるサービスに接続を試みる。uuid に 0000111F-0000-1000-8000-00805F9B34FB を指定することで、HFP AG に接続を試みる。

5.3 AT コマンド

Android のソースを見ると、BluetoothHeadset クラスに AT コマンドのコマンドパーサ、リザルトコードジェネレータが存在するが、今回必要なのはコマンドジェネレータ、リザルトコードパーサである。また、API としても公開されていないので自前で実装することになる。

AT コマンドの形式は 2 種類に分けられる。ひとつは “ATA” や “AT+CNUM” のような、パラメータを持たないコマンド単体のものである。もうひとつは “AT+CMER=3,0,0,1,0” や “AT+CIND=?” のような、=の後にパラメータを持つものである。この 2 種類のコマンドを、前者のパラメータを null とすることでパラメータの有無を判別し、統一的なメソッドで扱えるようにする。

リザルトコードの形式もまた 2 種類に分けられ、“<cr><lf>+CIEV: 3,0<cr><lf>OK<cr><lf>” のような AT コマンドの AT を抜いた部分と返り値と結果の組み合わせを持つものと、“<cr><lf>OK<cr><lf>” や “<cr><lf>ERROR<cr><lf>” のような結果のみをもつものがある。これも後者を null として統一的なメソッドで扱い、さらに AT コマンド、パラメータ、リザルトコードの形式を 1 つのデータとして扱うことで、データを 1 箇所に集めることができる。

5.4 SCO-S リンクの確立とオーディオとの接続

SCO-S リンクを扱えるクラスやメソッドは Android SDK として公開されていない。Android のソースを見ると ScoSocket というクラスがあるが、非公開である。また、その ScoSocket クラスも Gingerbread までしか存在せず、以降のバージョンでの SCO の扱いは、BluetoothSocket クラスに統合されている。

SCO ソケットとオーディオを接続するための setBluetoothScoOn(boolean on) というメソッドがあるが、“Request use of Bluetooth SCO headset for communications.” とあるので、図 4.1 の AG 側で行われるような、SCO ソケットの in をマイクに、スピーカを out に割り当てるような処理だと思われる。

この 2 つが実装できなければ、音声転送を行うことができず、ハンドセットとしての機能を果たすことができない。

5.4.1 Android 非公開 API

Android SDK として公開されている API は、Android 全体のソースコードから見ると一部のものである。SDK として公開しない理由は不明で、推測するしかないが、Android

OS 内部の動作を外部から操作され、Android OS の動作が不安定になるというようなことを避けるために、アプリケーション開発者から操作できる API を制限しているなどの理由が妥当なところであろう。

この非公開部分の API(通称「非公開 API」) は、Java リフレクション API を使うことで利用できる。

公開されている API には SCO-S リンクを確立する方法がないため、リフレクションを使って ScoSocket クラスを利用し、SCO-S リンクを確立させる。

5.5 GUI の実装

Android Developer Tools(ADT) にはグラフィカルな GUI エディタがついており、GUI パーツをドラッグアンドドロップすることで簡単に GUI を作成することができる。図 5.1 は接続画面、図 5.2 はダイヤル画面、図 5.3 は通話画面を ADT で配置したものである。

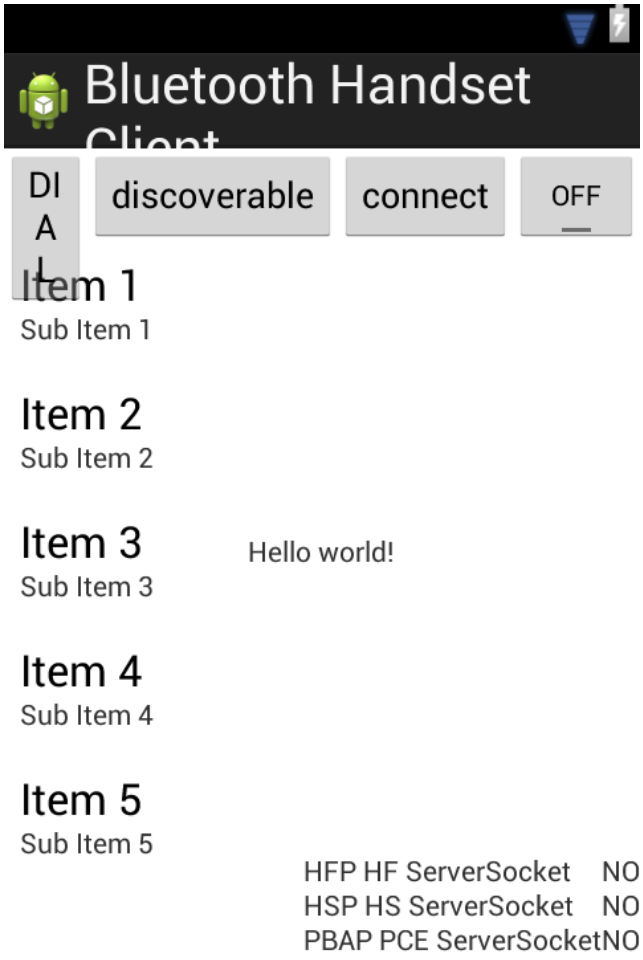

図 5.1　ADT で作成した接続画面

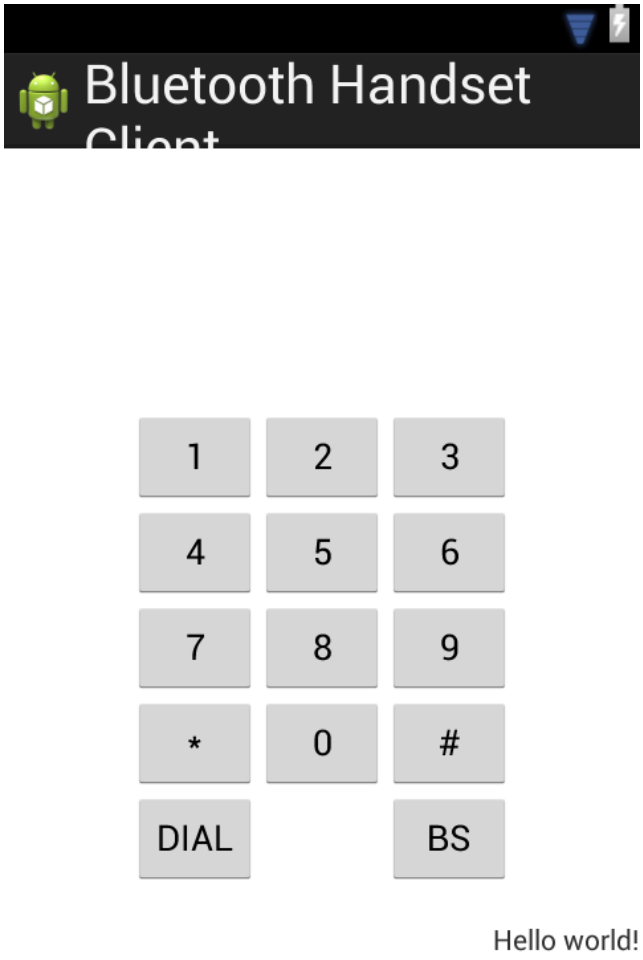

図 5.2　ADT で作成したダイヤル画面

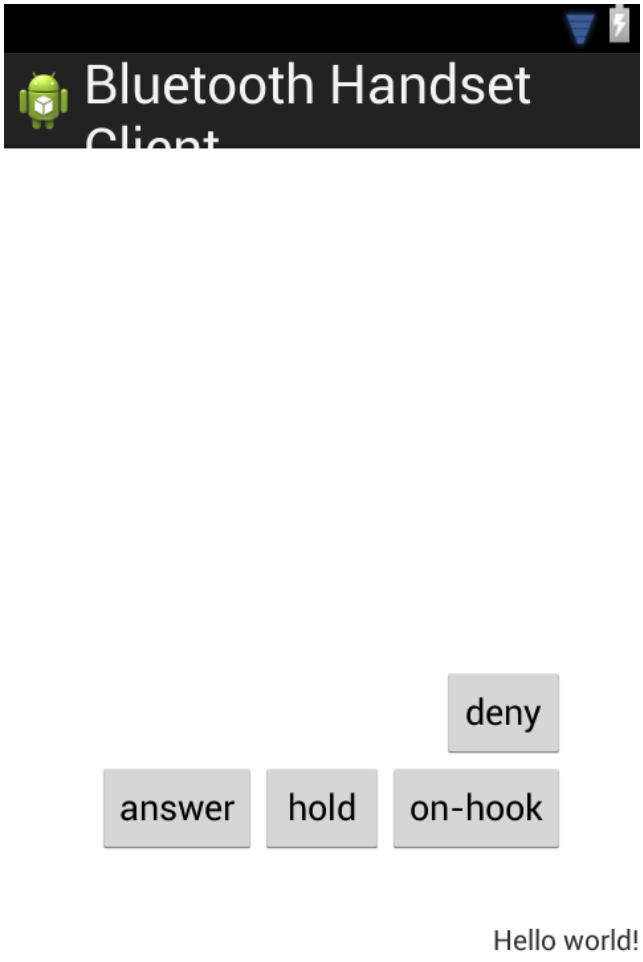

図 5.3　ADT で作成した通話画面

第 6 章

実験

表 6.1 に記載された機器を用いて実験を行った。
実験のログは付録 A を参照されたい。

6.1 RFCOMM の作成とサービスレコードの登録

Android バージョン 2.3 以下の L-04C と SH-13C は listenUsingRfcommWithServiceRecord(String name, UUID uuid) の引数に HFP FP の UUID を与えて実行することに成功したが (付録 A.1 参照)、バージョン 4.0 以降の機器は付録 A.2 のような例外が出て実行できなかった。

サーバーソケットだけでなく、クライアントソケットから connect() を行う方法でも接続を試みたが、HFP HF としてのサービスレコードが登録されていないと、connect() で接続しても HFP HF と認識されることはなかった。

この結果から、以降の実験は全て Android バージョン 2.x の機器で行っている。断りがない限り、実行している機器は L-04C である。

表 6.1 実験で用いた Android 搭載機器

製造元	製品名	モデルナンバー	Android バージョン
LG Electronics	Optimus chat	L-04C	2.2
SHARP	AQUOS PHONE f	SH-13C	2.3.4
SAMSUNG	GALAXY NEXUS	SC-04D	4.0.1
SAMSUNG	GALAXY Note	SC-05D	4.0.4
SONY	Xperia SX	SO-05D	4.0.4
SONY	Sony Tablet S	SGPT113JP/S	4.0.3

6.2 AT コマンド

HF と AG を接続した RFCOMM に、HF から HFP 仕様書に準拠した AT コマンドを発行することによって、意図通りのリザルトコードが返り、HFP サービスとしての接続が確立した (付録 A.3 参照)。

また、図 6.1 のようにダイヤル画面でダイヤルし、DIAL と書いてある発信ボタンを押すことで、通話画面に遷移し、ATD コマンドが発行され、図 6.2 のように発信することができた (付録 A.4 参照)。HFP 接続中に着信することによって、図 6.3 のように通話画面に遷移し、着信することができた。ANSWER ボタンを押すことによって、ATA コマンドが発行され、応答することができた。

図 6.2 と 6.3 が同じ画面に見えるのは、設計段階ではあった通話画面上部の「発信中」「着信中」の表示を実装していないからである。

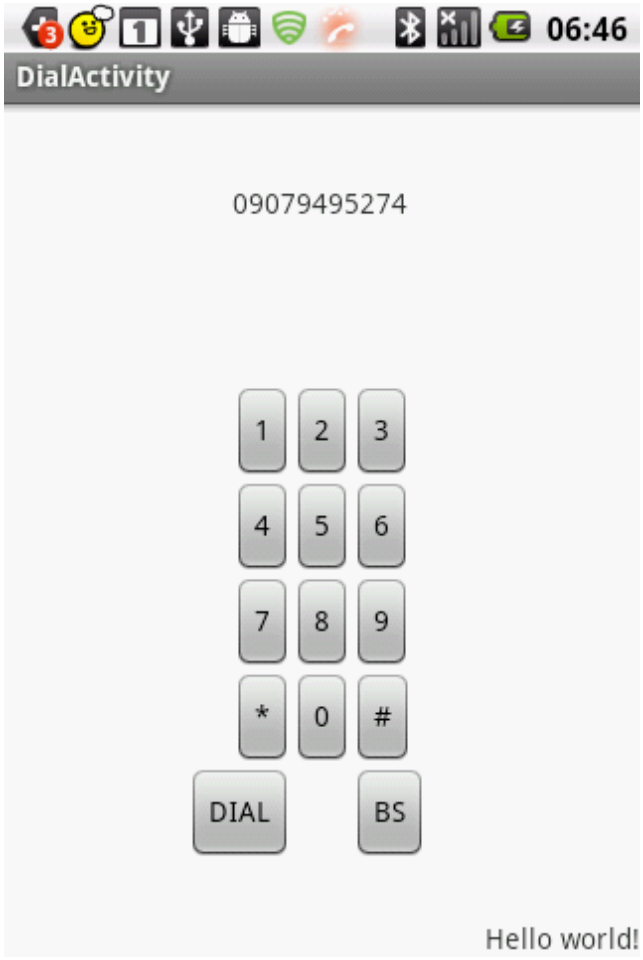

図 6.1 実装したダイヤラでダイヤルしている様子

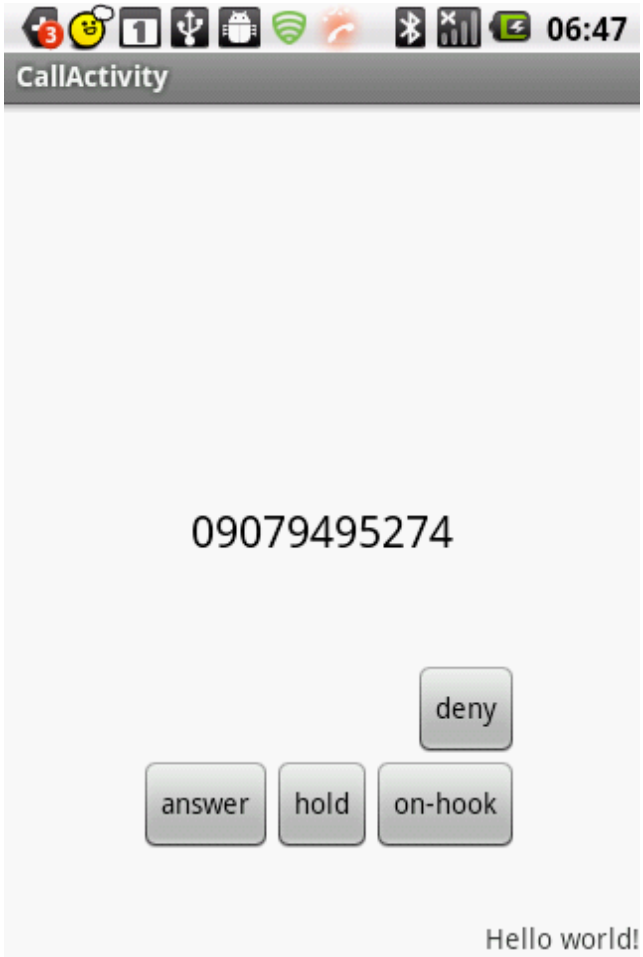

図 6.2　ダイヤルした番号に発信している様子

6.3　非公開 API を用いた SCO-S リンクの確立とオーディオとの接続

ScoSocket クラスにリフレクションをし、SCO ソケットを作成して accept()、connect() を試したが、どちらも AG と接続されることはなかった。ログにある `[BTUI] [### SCO ###] doAccept()... mState(2) [1:READY/2:ACCEPT/3:CONNECTING/4:CON` というのが accept() を実行した結果である。

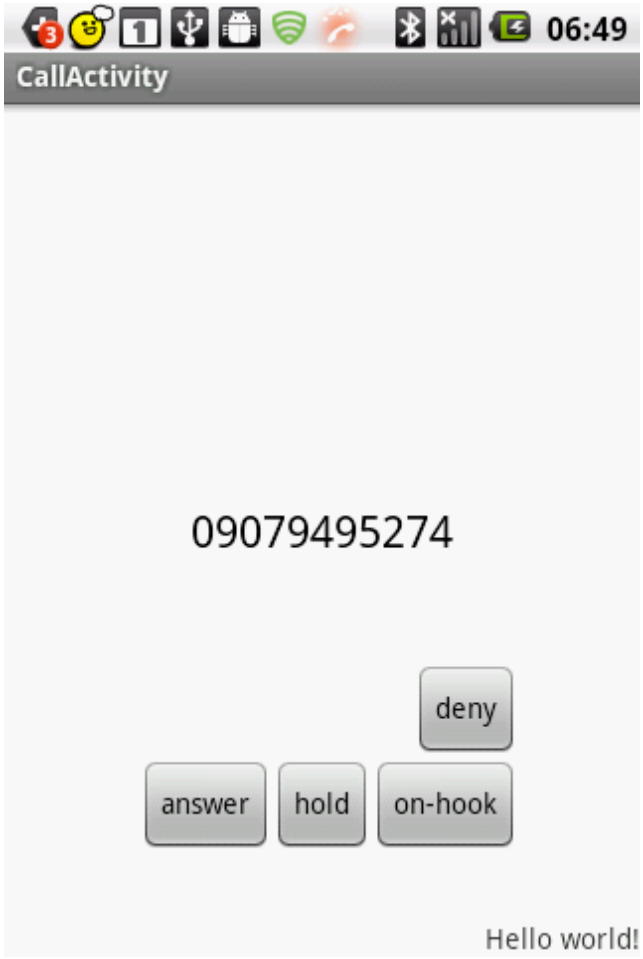

図 6.3　HFP 接続中に着信している様子

第 7 章

考察

Android バージョン 2.x に関しては、実験は部分的には成功したと言える。しかし、実験全体を見てみると、失敗したものの方が多い。

その中でも一番大きな要素を占めるのは、音声転送の失敗である。SCO-S リンクが確立しないことが原因で、ハンドセットとしての能力を備えることができなかった。この問題が解決できれば、通話の制御は可能であり、機能追加や GUI の見直し等を行うことによって、本研究の目的を達成することができると考えている。

7.1 実験データの妥当性

7.1.1 制御データの妥当性

制御データの送受信が確認できる機器で実験を行ったところ、AT コマンドとしての形式で送信され、正しく受信できることが確認できている。この AT コマンドは Hands-Free Profile 1.5 仕様書に定められた手順で発行・受信されているため、HFP HF として妥当な制御データを送受信できていると言える。

また、Bluetooth SIG の認定を受けた機器 3 台に対して実験を行い、接続・制御ができたことより、HFP HF として HFP AG に対して適切な互換性を持っていると考えている。

本来ならば Bluetooth SIG の定める Bluetooth Qualification Program のような基準を満たすかを確認するべきであるが、Bluetooth SIG 公認のテストプログラムを受けるにはまず Bluetooth SIG のメンバーになる必要があり、さらに第三者機関による認定を受ける必要がある。現段階ではそのような高レベルの妥当性は必要ないと考えた。本研究の有効性が一般的に認知され、広く利用されるような展開があれば、認定を受けることも検討したい。

7.1.2 音声データの妥当性

今回の実験では残念ながら音声転送に失敗したため、音声データの妥当性を調査することができなかった。成功した場合には音声データの音質について妥当であるか、人間の耳による感覚的な調査と、以下のような手順の定量的な調査を行う予定であった。

1. HFP HF を実装したスマートフォンの音声出力 (スピーカ)、入力 (マイク) の録音が出来るよう準備する。
2. 通話の出力 (スピーカ)、入力 (マイク) の録音を行え、録音データが取り出せる電話機を用意する。
3. HFP AG となる電話機と 2 の電話機で通話を行い、1 のスマートフォン、2 の電話機共に録音を行う。
4. データを取り出し、1 の入力と 2 の出力、2 の入力と 1 の出力の波形データを比較し、どの程度一致しているかを調べる。

この調査方法で、本研究の実装を通してどの程度音声データが劣化または損失するのかを調べることができる。人間の耳による感覚的な調査により、本研究の実装が通話として有効な音質を提供することができるかを調べることができる。

また、HF・AG 間の Bluetooth の電波状況を変化させて調査することによって、どのようにデータが変化するかを調べることができる。それを人間の感覚と組合せることにより、どの程度音声データが劣化すると通話に支障が出るかが分かり、本研究が有効な電波状況がどの程度かを求めることができる。

7.2 本研究の有効性の検討

本研究の目的は、複数台持ちユーザが操作する携帯電話の台数を遠隔操作によって減らし、ユーザの利便性を向上させることであった。実験結果により、ユーザの利便性を向上させ、目的を達成できているかを検討する。

図 7.1 は図 2.1 を実装状況で色分けした図である。青が実装完了した動作、赤が実装できなかった動作である。

実装できなかった動作は、主に音声転送ができなかったことが原因である。

着信合図の確認動作については、HFP AG の実装には着信音を転送できるものとできないものがあり、着信音を転送できるものについては通話より前に SCO-S リンクを確立し、音声転送をする必要がある。着信合図を確認するためには、AG からの着信音転送以外にも、画面の表示を変更する、HF 側が独立して音を鳴らす、バイブレーションさせる

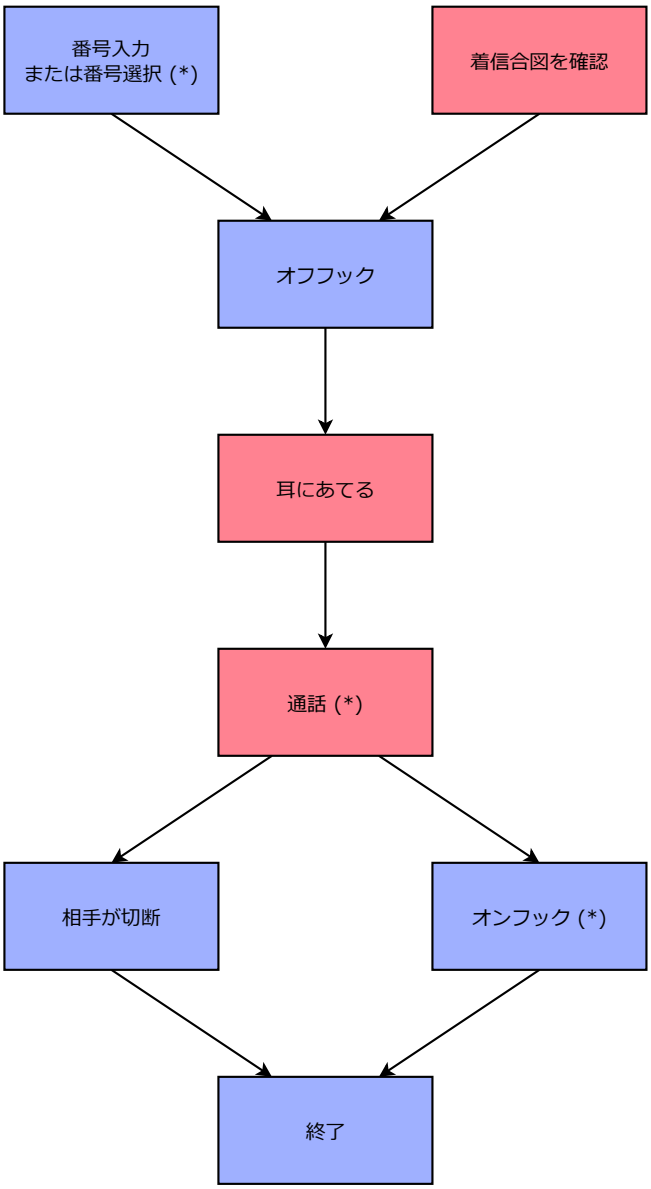

図 7.1　通話操作全体の流れの実装状況

等の合図を出すことで実装できる。現時点ではハンドセットでの通話ができず、着信合図が重要な動作となるまで至らなかったため画面表示以外の実装を見送ったが、着信音の転送以外は実装するための手法の見通しは立っており、将来的に実装する予定である。

画面に対する目視による確認しかできないため、通知方法としては確実性に劣るが着信合図は確認できる。しかし通話動作は音声転送が実装できていないためできない。そのため、発着信のあった電話機のスピーカから通話音声が流れ、同電話機のマイクから通話音声を拾う。これにより、ハンドセットではなく電話機を耳にあてるという動作が発生し、結局電話機を取り出さなければならない。

以上から、図 7.1 のような流れで従来行われていた通話操作を完全に遠隔操作することはできなかった。従来の操作と同じく、発着信のあった電話機を耳にあてるという物理的所作が発生する点で、現時点での実装では有効であるとは言い難い。

現時点での実装で実用にするならば、常にスピーカが耳に、マイクが口元にあたっている状況で利用すれば良いことになる。例えば、補助として通話用携帯電話にヘッドセットを接続し、装着することで、耳にあてるという動作を省略し、ハンドセットで統合的に発着信操作を行えるようになる。

7.2.1 通話網の構築について

現時点での実装では音声の転送に失敗したため、応用として挙げていた Bluetooth による通話網の構築は実現できなかった。そのため、机上の議論となってしまうが、実験が成功していたと仮定した上で、通話網の構築について議論したい。

HFP のみで通話網が構築できていたかについては、通話網を構築するための実装・実験を行っていないので現時点までの実装を行った知見からの予想となるが、構築できなかったのではないかと考える。これは HFP があくまで AG から HF への単方向の転送のために定められているからであり、また音声転送のトリガーが発着信のみに限られていることによる。

図 7.2 は、A を B と C のハンドセットとして使い、利便性の向上をはかるという本研究を表した図である。

この場合、B に発生した通話を A で受ける、C に発生した通話を A で受けることは想定している。しかし、A で発生した通話を B または C で受けることは想定されていない。AG から HF への通話転送は行えるが、HF から AG への通話転送は行えない。

つまり、B に発生した通話を A で受け、さらに C に転送することで B–C 間の通話を可能にするという通話網は、現時点では実現できない。

B の通話を C に転送することだけを考えると、図 7.3 のように A が AG と HF を兼任することで、転送経路が確保できる。これは本来 Bluetooth 搭載スマートフォンに実装されている AG の機能と、本研究の HF の機能を同時に動作させれば良い。しかし、今度は逆に C から A への転送経路が断たれ、操作する機器を減らすことができない。

所持する携帯電話を全てスマートフォンにすることで、図 7.3 の A と同様に B・C も AG と HF を兼任することで、転送経路の問題は解決するように思える。しかし、音声転送のトリガーが発着信のみに限られている問題があり、A・B・C 間で任意のタイミングで通話の転送をすることはできない。

AG や HF のような役割のない完全対称で、任意のタイミングで Bluetooth 機器同士で通話を行えるプロファイルとして、Intercom Profile(ICP)[18] がある。ICP はトラン

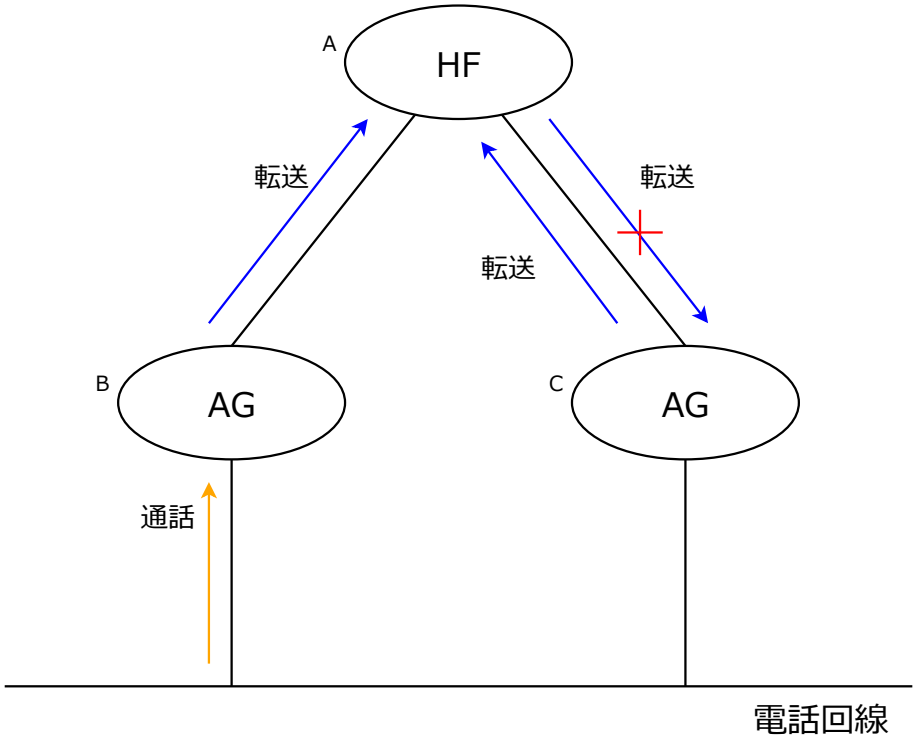

図 7.2　本研究で想定している AG/HF 間の通信方向

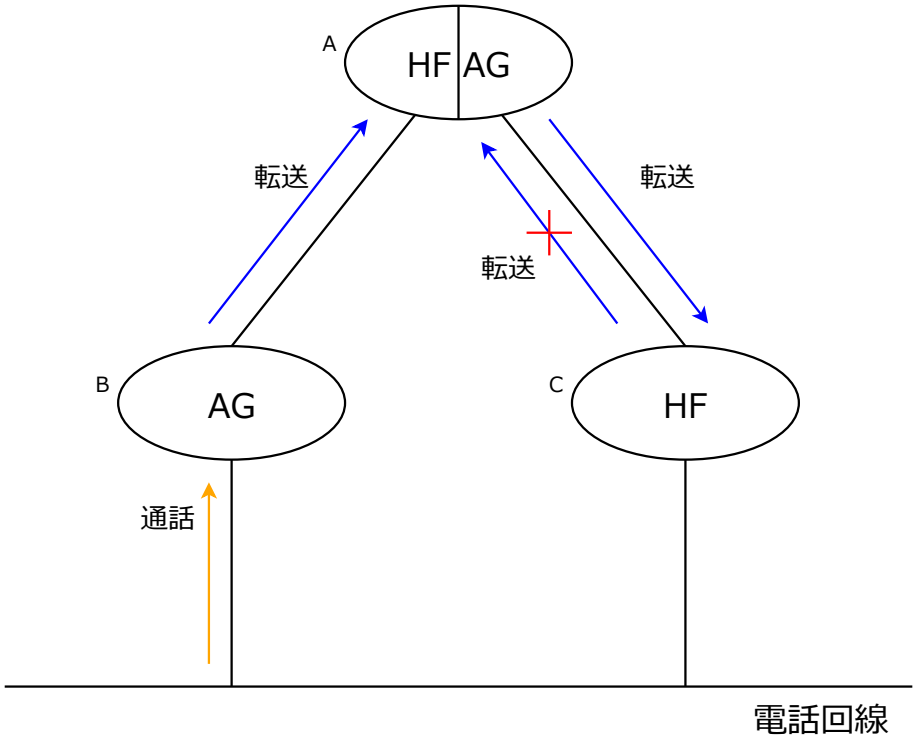

図 7.3　AG と HF を兼任した場合の通信方向

シーバプロファイルとも呼ばれ、同じネットワークにある 2 台の Bluetooth 機器が、公衆電話網を使わずに直接通信することができる。HFP に加え、ICP を合わせて利用することで、Bluetooth を用いた仮想通話網を実現できるのではないかと思われる。

しかし、ICP にも SCO-S リンクが使われるため、ICP を実装するためにも SCO-S リンクを確立できるようにならなければいけない。

7.3 本研究を Android の標準として策定する場合の検討・課題

本研究の対象として設定している複数台持ちユーザがさらに増加した場合、Android 開発者が本研究と同様の目的で、Android の標準機能として遠隔操作機能を追加することが考えられる。この時、本研究とどのような違いがあるか、また仕様を決定するにあたって本研究で得られた知見をもとにどのような提案ができるかを考える。

7.3.1 通話機能に対して遠隔操作を行う場合

本研究と同様に通話機能に対して遠隔操作を行う場合、本研究と同じく接続に Bluetooth を使用し、ソフトウェアとして HFP HF を実装する方向性で良いと考える。

これは、第 2 章で述べた通り、追加のアプリケーションを導入できない携帯電話を操作することを考えると、既存のソフトウェア資源である HFP AG の実装を利用した方が良いと判断したためである。Android 同士の接続のみ考えるのであれば、Bluetooth や Wi-Fi、さらに今後普及させることを念頭に置いた独自規格の無線通信等も考えられるが、現時点でも実現できる手法が存在しているのに、追加の投資を行う必要はないと感じる。

以上を踏まえて、現時点で実装と API として足りない

- AT コマンドや RFCOMM、SCO 接続の確立などをまとめた HFP HF クラス
- HF 用のオーディオの切り替え

を実装し、API を制定すれば標準機能として搭載し、発展させていくことが可能だと考える。

通話網について

この案を利用して通話網を構築するには、第 7 章で述べた通り、HFP と ICP を併用することで実現できると思われる。この場合に必要となるのは、

- ICP に必要な機能をまとめた ICP クラス

- HFP と ICP のオーディオを合わせるためのミキサー機能

等が挙げられる。

7.3.2 通話機能以外に対して遠隔操作を行う場合

本研究では通話操作を主な対象としてきたが、本研究を行うための動機が発生するほど携帯電話が変化したように、今後アプリケーションの操作等、通話機能以外を目的とした複数台持ちが増加することも考えられる。その場合、新たに画面と入力全てを転送するような遠隔操作の規格を採用する必要があるだろう。

通常、携帯電話には一部の機能を遠隔操作できる仕組みは組込まれていても、携帯電話の全機能を任意の操作で遠隔操作するような仕組みは組込まれていない。例えば、Android には ADB[*1] という遠隔操作のできるデバッグ用のソフトウェアが組込まれているが、ADB は標準で有効にはなっておらず、また ADB で Android の全ての操作を行えるわけではない。遠隔操作用のシステムに遠隔操作技術が組込まれていなければ、追加のソフトウェアが必要となる。

ここで、追加のソフトウェアが必要となるため、従来の携帯電話を対象から除外し、スマートフォンのみを対象とする。

ユーザの行える全ての操作を遠隔操作するためには、全てのユーザインターフェースを遠隔地に転送すれば良い。VNC や RDP のように、画面表示、タッチパネルやキーボード等を操作した際のイベント、スピーカやマイク等のオーディオのような、様々なユーザインターフェスに関するデータを送受信する規格が必要となる。Wi-Fi や Bluetooth を介してそのデータを送受信し、画面の描画やイベントの解釈等を行えば、遠隔操作ができるだろう。

VNC であればサーバもクライアントも Android アプリケーションとしての実装が存在するので、それらをもとにプロトタイプを作り評価することで、本研究の通話機能の遠隔操作と同様に具体的な提案ができるのではないかと思う。

通話網について

2 台以上の機器を同時に遠隔操作し、音声をミキサーで処理すれば実現可能であると考える。

[*1] Android Debug Bridge

第 8 章

問題点と課題

実験が失敗したことによって、大きく 2 点の問題点を残した。Android 4.0 において RFCOMM 作成時に HFP HF サービスを登録できない問題と、音声転送のために SCO ソケットを作成しても接続が確立しない問題である。この問題点を解決するにあたって、どのような部分でどのような原因で問題が発生しているのか、問題の解決方法としてどのようなことが考えられるかを検討する。

また、本研究ではこのような問題が発生したにも関わらず、解決することができなかった。何が原因で解決できなかったのか、どのように解決すべきだったのかを検討する。

8.1 RFCOMM 作成時に HFP HF サービスを登録できない問題

これは恐らく Android バージョン 4.0 以上で起こる問題である。実装に利用した `BluetoothAdapter#listenUsingRfcommWithServiceRecord()` のソースコードの中で、問題を起こしている部分のコード断片が以下の部分である。

```
int handle = -1;
try {
  handle = mService.addRfcommServiceRecord(name, new ParcelUuid(uuid), channel,
    new Binder());
} catch (RemoteException e) {Log.e(TAG, "", e);}
if (handle == -1) {
  try {
    socket.close();
  } catch (IOException e) {}
  throw new IOException("Not able to register SDP record for " + name);
}
```

この部分のソースコードは 2.x でも 4.0 でも変わらない。handle が-1 であることでこの例外が発生しているので、`mService.addRfcommServiceRecord()` が-1 を返していることになる。`mService.addRfcommServiceRecord()` のどの部分が-1 を返す原因と

なっているかを調査できればいいが、このメソッドは Binder を使用し、他のプロセスに通信を行っているので、デバッグとして動作を追うことは難しい。

このソースコードの詳細と結果について記す。`listenUsingRfcommWithServiceRecord()` が呼び出された時の一連の動作は図 8.1 のようになっている。mService の型は IBluetooth というインターフェースクラスであり、`mService.addRfcommServiceRecord()` でプロセス間通信を用いて呼び出されるメソッドは IBluetooth をインプリメントした BluetoothService というクラスの `addRfcommServiceRecord()` となる。このように、インターフェースクラスをインプリメントしたクラスの同名のメソッドを、一定の法則でプロセス間通信を用いて呼び出すことができる。図 8.1 の※印がついた部分がプロセス間通信を行い、-1 が返ってきた部分である。

ここで `BluetoothService#addRfcommServiceRecord()` の動作を確認すると、その中でさらに `addRfcommServiceRecordNative()` というメソッドを呼び出している。このメソッドがネイティブバイナリとプロセス間通信を行い、RFCOMM ソケットの作成とサービスレコードの登録を行っていると思われる。`addRfcommServiceRecordNative()` には例外処理はなく、`BluetoothService#addRfcommServiceRecord()` 内には例外処理があるが、上記コードと同じく `Log.e()` を呼び出す処理となっており、実行時に呼び出された形跡はなかった。つまり、try 節内で例外が起きて初期値の-1 が `handle == -1` の判定を成立させているわけではなく、`mService.addRfcommServiceRecord()` が正常終了し、その結果が-1 であったことが分かった。その結果として if 節の内部に入り、ソケットを閉じる時も例外を起こすこともなく、IOException の例外を投げてこのメソッドを抜け、付録 A.2 が記録された。

以上から、この問題の動作を追う場合は BluetoothService クラスとネイティブプロセスを調査する必要がある。BluetoothService クラスのコードは 2.x と 4.0 で大きく変わってはいないので、ネイティブプロセスにあたる下層のソフトウェアスタック、例えば BlueZ の挙動が 2.x と 4.0 で変わったため、このような問題が発生していると現時点では考えている。

8.1.1 解決策の提案

ユーザープロセスから addRfcommServiceRecordNative を直接実行する方法を模索する ScoSocket のようにリフレクションで BluetoothService のメソッドを実行できれば良いが、BluetoothService はサービスプロセスであり、サービスプロセスからはインスタンスを得ることができないので、リフレクションで `BluetoothService#addRfcommServiceRecord()` や `BluetoothService#addRfcommServiceRecordNative()` を呼び出すことはできない。

図 8.1　listenUsingRfcommWithServiceRecord が呼び出された時の一連の動作

ユーザープロセスである `IBluetooth#addRfcommServiceRecord()` がサービスプロセスを実行しているので、同手順で `BluetoothService#addRfcommServiceRecordNative()` を直接実行する方法を考える。しかし前述の通り、インプリメントの元となるインターフェースクラスに同名のプロセス間通信用メソッドがなければならないので、IBluetooth を変更しなければならない。IBluetooth はフレームワークライブラリとして提供されている部分であり、変更するには非常にコストが高い。IBluetooth を継承して `addRfcommServiceRecordNative()` メソッドを持つクラスを作ることも考えたが、こ

れはもはや BluetoothService の元となるクラスではないため、BluetoothService クラスと通信できないことが予想できる。

また、前述の通りネイティブの部分で処理が変更されていると予想しているので、`addRfcommServiceRecordNative()` を直接実行することができたとしても、同じ結果が返ってくることが考えられる。しかし、`addRfcommServiceRecordNative()` を直接実行して同じ結果が返ってきたとすれば、ネイティブの部分が失敗の原因であると特定できるため、この手法を行うことで評価できる部分は存在する。

フレームワークライブラリの処理を変更する IBluetooth クラスの内容の変更のように、BluetoothService クラスを変更してしまえば、意図通りの結果が得られると思われる。

しかし、これは Android OS に手を加えることを意味し、最悪の場合 Android 端末が起動しなくなる。そのようなことを安全に実行できる端末はかなり限られており、またユーザーにそのような負担を強いることは解決策としては良策でないと考える。しかし、本研究を普及させることは難しくなるかもしれないが、実験が失敗していた現時点と比較して評価することは可能となるので、手法の 1 つとして取り組みたい。

ネイティブバイナリを利用する Android の内部で使われている BlueZ、D-Bus を直接利用できれば解決できるのではないかと考える。

ただし、Bluetooth デバイスを直接利用するために、Linux の root と同等のスーパーユーザ権限が必要になる可能性がある。これはフレームワークライブラリの改造よりは程度は低いが同じくユーザーに負担を強いることになる。しかし、前項と同様評価をすることは可能となるので、手法の 1 つとして取り組みたい。

8.2 SCO ソケットを作成しても接続が確立しない問題

本研究の接続ログ (付録 A.7 参照) と、既製品の接続ログ (付録 A.6 参照) を比較して、明らかに違う点がある。接続が成功している既製品のログでは、headset address に値が入っている。しかし、本研究のログでは headset address は null である。

実装する工程で headset address というものを登録する手続きを見落している可能性が高い。この値を正しい値にすることができれば、SCO-S リンクが確立される可能性も十分考えられる。

8.2.1 解決策の提案

AG 側から接続させる方法を模索する SCO 接続が必要になる時に特別な操作が行われているわけではないので、SCO 接続が必要になる前の段階に対して調査を進めなければ

ならない。

AG の実装は Bluetooth の利用準備から調査すると広範に渡るので、明らかに関係があると分かる範囲で調査したところ、以下のようなことが分かった。

- Bluetooth の MAC アドレスを伝える AT コマンドはない。
- Android の AG 実装にも特に AT コマンドを受け取っている記述はない。
- `ScoSocket#connect()` を行う前に、特別な処理で HF の MAC アドレスを調べている記述はない。

HFP HF を実装する時固有の問題ではなく、ScoSocket クラス自体がユーザアプリケーションからうまく扱えないという可能性を調べるため、2 台の Android 機器を用いて、片方は `ScoSocket#accept()`、片方は `ScoSocket#connect()` のみを行うテストプログラムを作成し実行した。

connect を行った機器の結果は以下のようになった。

```
E/BTL_IFC(15145): ##### ERROR : tx_data: write failed (-1)#####
E/BTL_IFC(15145): ##### ERROR : BTL_IFC_CtrlSend: [BTL_IFC CTRL] send failed#####
E/ScoSocket(15145): [BTUI] [### SCO ###] doClose()... mState(1)
 [1:READY/2:ACCEPT/3:CONNECTING/4:CONNECTED/5:CLOSED]
```

また、accept を行った機器の結果は以下のようになった。

```
E/BTL_IFC(15145): ##### ERROR : tx_data: write failed (-1)#####
E/BTL_IFC(15145): ##### ERROR : BTL_IFC_CtrlSend: [BTL_IFC CTRL] send failed#####
E/ScoSocket(15145): [BTUI] [### SCO ###] doClose()... mState(2)
 [1:READY/2:ACCEPT/3:CONNECTING/4:CONNECTED/5:CLOSED]
```

SCO-S リンクの確立はどちらも失敗に終わり、ScoSocket は close されている。

この結果は付録 A.5 でも検出されている。つまり、既存の AG と接続し、AG から SCO-S リンクの接続要求を受け付けたが、なんらかの理由によりリンクの確立に失敗したということになる。

このエラーメッセージは実験機 (L-04C) では`/system/lib/libandroid_runtime.so` に存在し、libandroid_runtime.so に strings で解析を行ったところ、以下のような文字列データを検出した。

```
##### ERROR : %s: write failed (%d)#####
##### ERROR : %s: [BTL_IFC CTRL] send failed#####
```

この結果より、Android 4.0 での RFCOMM 接続の作成と同様、ネイティブの部分まで調査する必要があるだろう。調査によって SCO-S リンクが確立できるようになれば、本来の意図通りの実装を行うことができる。ネイティブの部分を変更する必要が発生した

としても、実験を行う上での手法として評価できる。

SCO での接続を諦め、RFCOMM などで接続する　この方法を取ると、HFP 仕様書から逸脱するため、既存の AG との互換性が失われる。よって、接続できるのは本研究のアプリケーション同士のみとなり、遠隔操作可能な対象が減少してしまう。これは目的に対して好ましくない。しかし、これにより通信ができるようになるならば評価実験を行うことはでき、研究が進展すると思われる。

また実装面でも問題があり、RFCOMM は非同期な通信プロトコルである。通話は送信と受信が同期的に行われるため、SCO のような同期的なリンクを利用した。RFCOMM を用いて通話を行うならば、RFCOMM 上に同期的なデータ通信プロトコルを実装する必要がある。

8.3　実験の失敗に対する問題点

本研究は実験に失敗し、重要な部分でデータを取ることができなかった。失敗した原因に対し、これだけの解決策を提案できたのになぜ実行しなかったのかという点について考えた。

Bluetooth HFP は実現手法の一つに過ぎないのに、手法に拘りすぎた点が挙げられる。Bluetooth を用いた個人用仮想通話網を構築するには、Bluetooth HFP 以外の手法も考えられたはずである。例えば解決策として SCO ソケットを作成しない手法も提案している。また、対象が減る等のリスクは挙げているが、フレームワークを変更するという解決策も提案している。そのような選択肢を選んで実装・評価することで、期待した結果全てを得られる訳ではないだろうが、参考になる結果を得、考察する事はできたのではないかと思う。

実験が失敗したこと事態は必ずしも悪いことではないだろうが、失敗に拘り、手が止まってしまったのは良くないことである。今後の研究に対する姿勢への課題として改善していきたい。

第 9 章

おわりに

本研究では、携帯電話を複数台持つ形態における、携帯電話を 1 台持つ形態と比べて操作しなければならない対象が増えるという問題に対して、複数台ある携帯電話の中からスマートフォンをハンドセットにし、操作をまとめるというアプローチで解決に取り組んだ。

しかし、HFP HF としてのサービスレコードが登録できない、接続プロトコルである RFCOMM のサーバーソケットが作成できない、SCO-S リンクが確立されず、音声転送ができないというハンドセットとして致命的な問題が発生する等、非常に悔いの残る結果となった。

携帯電話市場にとって、スマートフォンはまだ十分に伸びしろのある分野である。この失敗のまま研究を終わりにせず、スマートフォン市場の成長とともにこの研究も成長させるような気持ちで、問題点の解決等続けていきたいと思う。

謝辞

本研究を遂行するにあたっては、いろいろな方々にお世話になりました。

まず、指導教員の多田好克先生には日頃から熱心なご指導、そしてご鞭撻を賜わりました。また、ご多忙中にもかかわらず論文の草稿を丁寧に読んで下さり、大変貴重なご助言をいただきました。ここに厚く御礼申し上げます。

そして、本研究が行なえたことは、研究方針や方法論について議論をし、共に研究生活をおくってきた多田研、小宮研をはじめとする、基盤ソフトウェア学講座の学生諸氏おかげでもあります。最後に、これらの皆さんに感謝いたします。

参考文献

[1] Bluetooth SIG. https://www.bluetooth.org/.

[2] インプレス R&D：“スマートフォン利用率が 22.9 %と倍増、スマートフォンユーザーの twitter 利用率は 40.6 %―8 年目の実績、個人 3304 人企業担当者 1636 人のモバイル利用動向を収録―『スマートフォン/ケータイ利用動向調査 2012』11 月 10 日（木）発売” (2011). http://www.impressrd.jp/news/111108/kwp2012.

[3] インプレス R&D：“スマートフォン利用率は個人が 39.9 %、企業が 41.7 %とほぼ倍増個人のスマートフォンユーザーの facebook 利用率は 38.7 % 9 年目の実績、個人 3262 人企業担当者 1795 人『スマートフォン/ケータイ利用動向調査 2013』　11 月 22 日（木）発売” (2012). http://www.impressrd.jp/news/121120/kwp2013.

[4] 日経 BP コンサルティング：“スマートフォンの国内普及率は 9.5 %　20 歳代が市場を牽引― ニュースリリース” (2011). http://consult.nikkeibp.co.jp/consult/news/2011/0801mobile/.

[5] 日経 BP コンサルティング：“スマートフォンの国内普及率は 18.0 %、1 年でほぼ倍増― ニュースリリース” (2012). http://consult.nikkeibp.co.jp/consult/news/2012/0726sp/.

[6] 社団法人 電気通信事業者協会. http://www.tca.or.jp/.

[7] 社団法人 電気通信事業者協会：“携帯電話・PHS 契約数”. http://www.tca.or.jp/database/index.html.

[8] Bluetooth SIG: “Hands-free profile 1.5” (2005).

[9] Bluetooth SIG: “Grow your business with bluetooth ® technology” (2012).

[10] Andr é：“Handset connectivity technologies – 3rd edition”, Technical report, Berg Insight (2012).

[11] Electronic Industries Alliance: “Data transmission systems and equipment - serial asynchronous automatic dialing and control” (2000).

[12] Telecommunications Industry Association: “Data transmission systems and equipment - serial asynchronous automatic dialing and control” (2005).

[13] WILLCOM: “SOCIUS” (2011). http://www.willcom-inc.com/ja/lineup/wx/01s/index.html.

[14] WILLCOM: “WX01S　＜ SOCIUS：ソキウス＞　～話しやすさを追求した新スタイル～” (2011). http://www.willcom-inc.com/ja/corporate/press/2011/09/21/index_01.html.

[15] GREEN HOUSE: “Bluetooth ミニフォン | GH-BHMPA” (2012). http://www.

green-house.co.jp/products/av/mobilephone/receiver/gh-bhmpa/.

[16] 友和商会：“デジタルオーディオフォン BE03J サポートページ” (2012). http://www.yuwa-shop.jp/be03j/index.html.

[17] Google: “Dashboards — android developers” (2013). http://developer.android.com/about/dashboards/index.html.

[18] Bluetooth SIG: “The official bluetooth sig member website — intercom profile (icp)”. https://www.bluetooth.org/building/howtechnologyworks/profilesandprotocicp.htm.

付録 A

実験時に出力されたログ

A.1 RFCOMM サーバーソケットの作成 (L-04C)

```
01-24 06:51:41.541: I/System.out(15145): bluetooth support.
01-24 06:51:41.551: I/System.out(15145): REQUEST_ENABLE_BT = 0
01-24 06:51:41.561: I/System.out(15145): AQUES : 00:1F:81:00:0
1:00
01-24 06:51:41.561: I/System.out(15145): Bluetooth Keyboard :
98:72:01:00:64:19
01-24 06:51:41.561: I/System.out(15145): F04B : 00:23:26:94:15
:74
01-24 06:51:41.571: I/System.out(15145): ENLITE : 00:03:7A:EE:
39:73
01-24 06:51:41.571: I/System.out(15145): L-04C : B0:89:91:61:3
3:E3
01-24 06:51:41.581: I/System.out(15145): IS01 : A0:DD:E5:27:C5
:6F
01-24 06:51:41.581: I/System.out(15145): SC-05D : 5C:E8:EB:47:
B0:F4
01-24 06:51:41.601: I/System.out(15145): Galaxy Nexus : B0:D0:
9C:02:F6:55
01-24 06:51:41.601: I/System.out(15145): MW-140BT5637 : 00:02:
C7:20:82:8D
01-24 06:51:41.611: I/System.out(15145): SH-13C : 68:79:ED:EB:
26:01
01-24 06:51:41.651: I/System.out(15145): discovery started.
01-24 06:51:41.661: I/System.out(15145): send intent [00001108
-0000-1000-8000-00805F9B34FB]
01-24 06:51:41.671: I/System.out(15145): send intent [0000111E
-0000-1000-8000-00805F9B34FB]
01-24 06:51:41.681: I/System.out(15145): send intent [0000112E
-0000-1000-8000-00805F9B34FB]
01-24 06:51:41.681: I/System.out(15145): toggle button exec.
01-24 06:51:41.691: I/System.out(15145): toggle button execed.
```

```
01-24 06:51:41.731: I/System.out(15145): bas constructor.
01-24 06:51:41.731: I/System.out(15145): bas created.
01-24 06:51:41.741: I/System.out(15145): bas start @ Bluehand
01-24 06:51:42.811: I/System.out(15145): android.bluetooth.Blu
etoothServerSocket@449be040
01-24 06:51:42.831: I/System.out(15145): send intent [00001108
-0000-1000-8000-00805F9B34FB] *OK*
01-24 06:51:42.871: I/System.out(15145): bas start @ Bluehand
01-24 06:51:42.931: I/System.out(15145): android.bluetooth.Blu
etoothServerSocket@449c0038
01-24 06:51:42.941: I/System.out(15145): send intent [0000111E
-0000-1000-8000-00805F9B34FB] *OK*
01-24 06:51:42.981: I/System.out(15145): bas start @ Bluehand
01-24 06:51:43.041: I/System.out(15145): android.bluetooth.Blu
etoothServerSocket@449c15a8
01-24 06:51:43.051: I/System.out(15145): send intent [0000112E
-0000-1000-8000-00805F9B34FB] *OK*
```

A.2 Android 4.x でサービスレコードが登録できない場合の例外

```
01-24 13:24:15.412: W/System.err(19415): java.io.IOException:
Not able to register SDP record for Bluehand
01-24 13:24:15.412: W/System.err(19415):  at android.bluetooth
.BluetoothAdapter.createNewRfcommSocketAndRecord(BluetoothAdap
ter.java:1133)
01-24 13:24:15.412: W/System.err(19415):  at android.bluetooth
.BluetoothAdapter.listenUsingRfcommWithServiceRecord(Bluetooth
Adapter.java:1018)
01-24 13:24:15.412: W/System.err(19415):  at net.ku_ten.androi
d.bluehand.BluetoothConnection.listenUsingRfcommWithServiceRec
ord(BluetoothConnection.java:52)
01-24 13:24:15.422: W/System.err(19415):  at net.ku_ten.androi
d.bluehand.BluetoothAcceptService$BluetoothAcceptThread.<init>
(BluetoothAcceptService.java:101)
01-24 13:24:15.422: W/System.err(19415):  at net.ku_ten.androi
d.bluehand.BluetoothAcceptService.onStartCommand(BluetoothAcce
ptService.java:56)
01-24 13:24:15.422: W/System.err(19415):  at android.app.Activ
ityThread.handleServiceArgs(ActivityThread.java:2370)
01-24 13:24:15.422: W/System.err(19415):  at android.app.Activ
```

```
ityThread.access$1900(ActivityThread.java:127)
01-24 13:24:15.422: W/System.err(19415):  at android.app.Activ
ityThread$H.handleMessage(ActivityThread.java:1221)
01-24 13:24:15.422: W/System.err(19415):  at android.os.Handle
r.dispatchMessage(Handler.java:99)
01-24 13:24:15.422: W/System.err(19415):  at android.os.Looper
.loop(Looper.java:137)
01-24 13:24:15.432: W/System.err(19415):  at android.app.Activ
ityThread.main(ActivityThread.java:4511)
01-24 13:24:15.432: W/System.err(19415):  at java.lang.reflect
.Method.invokeNative(Native Method)
01-24 13:24:15.432: W/System.err(19415):  at java.lang.reflect
.Method.invoke(Method.java:511)
01-24 13:24:15.432: W/System.err(19415):  at com.android.inter
nal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:976)
01-24 13:24:15.432: W/System.err(19415):  at com.android.inter
nal.os.ZygoteInit.main(ZygoteInit.java:743)
01-24 13:24:15.432: W/System.err(19415):  at dalvik.system.Nat
iveStart.main(Native Method)
```

A.3 HFP サービスへの接続完了 (L-04C)

```
01-24 06:51:54.521: I/System.out(15145): I/O Thread created.
01-24 06:51:54.641: E/ScoSocket(15145): [BTUI] [### SCO ###] d
oAccept()... mState(2) [1:READY/2:ACCEPT/3:CONNECTING/4:CONNEC
TED/5:CLOSED]
01-24 06:51:54.641: I/System.out(15145): sco socket accept.
01-24 06:51:54.681: I/System.out(15145): AT+BRSF=127
01-24 06:51:54.741: I/System.out(15145): 19
01-24 06:51:54.761: I/System.out(15145): received:
01-24 06:51:54.761: I/System.out(15145): +BRSF: 99
01-24 06:51:54.761: I/System.out(15145):
01-24 06:51:54.761: I/System.out(15145): OK
01-24 06:51:54.771: I/System.out(15145): empty line
01-24 06:51:54.781: I/System.out(15145): command: +BRSF
01-24 06:51:54.781: I/System.out(15145): parameter: 99
01-24 06:51:54.781: I/System.out(15145): empty line
01-24 06:51:54.791: I/System.out(15145): OK!!!
01-24 06:51:54.791: I/System.out(15145): empty line
01-24 06:51:54.791: I/System.out(15145): {command=+BRSF, resul
t=true, parameter=99}
```

```
01-24 06:51:54.801: I/System.out(15145): recieve length: 19
01-24 06:51:54.801: I/System.out(15145): parsed command: +BRSF
01-24 06:51:54.811: I/System.out(15145): +BRSF: true!!!!
01-24 06:51:54.811: I/System.out(15145): AT+CIND=?
01-24 06:51:54.861: I/System.out(15145): 138
01-24 06:51:54.881: I/System.out(15145): received:
01-24 06:51:54.881: I/System.out(15145): +CIND: ("service",(0-
1)),("call",(0-1)),("callsetup",(0-3)),("callheld",(0-2)),("si
gnal",(0-5)),("roam",(0-1)),("battchg",(0-5))
01-24 06:51:54.881: I/System.out(15145):
01-24 06:51:54.891: I/System.out(15145): OK
01-24 06:51:54.891: I/System.out(15145): empty line
01-24 06:51:54.901: I/System.out(15145): command: +CIND
01-24 06:51:54.901: I/System.out(15145): parameter: ("service"
,(0-1)),("call",(0-1)),("callsetup",(0-3)),("callheld",(0-2)),
("signal",(0-5)),("roam",(0-1)),("battchg",(0-5))
01-24 06:51:54.901: I/System.out(15145): empty line
01-24 06:51:54.901: I/System.out(15145): OK!!!
01-24 06:51:54.901: I/System.out(15145): empty line
01-24 06:51:54.911: I/System.out(15145): {command=+CIND, resul
t=true, parameter=("service",(0-1)),("call",(0-1)),("callsetup
",(0-3)),("callheld",(0-2)),("signal",(0-5)),("roam",(0-1)),("
battchg",(0-5))}
01-24 06:51:54.911: I/System.out(15145): recieve length: 138
01-24 06:51:54.921: I/System.out(15145): parsed command: +CIND
01-24 06:51:54.921: I/System.out(15145): +CIND: true!!!!
01-24 06:51:54.921: I/System.out(15145): AT+CIND?
01-24 06:51:54.981: I/System.out(15145): 30
01-24 06:51:55.001: I/System.out(15145): received:
01-24 06:51:55.001: I/System.out(15145): +CIND: 1,0,0,0,4,0,4
01-24 06:51:55.001: I/System.out(15145):
01-24 06:51:55.001: I/System.out(15145): OK
01-24 06:51:55.011: I/System.out(15145): empty line
01-24 06:51:55.021: I/System.out(15145): command: +CIND
01-24 06:51:55.021: I/System.out(15145): parameter: 1,0,0,0,4,
0,4
01-24 06:51:55.021: I/System.out(15145): empty line
01-24 06:51:55.021: I/System.out(15145): OK!!!
01-24 06:51:55.021: I/System.out(15145): empty line
01-24 06:51:55.031: I/System.out(15145): {command=+CIND, resul
t=true, parameter=1,0,0,0,4,0,4}
01-24 06:51:55.031: I/System.out(15145): recieve length: 30
```

```
01-24 06:51:55.031: I/System.out(15145): parsed command: +CIND
01-24 06:51:55.041: I/System.out(15145): +CIND: true!!!!
01-24 06:51:55.041: I/System.out(15145): AT+CMER=3,0,0,1,0
01-24 06:51:55.101: I/System.out(15145): 6
01-24 06:51:55.121: I/System.out(15145): received:
01-24 06:51:55.121: I/System.out(15145): OK
01-24 06:51:55.121: I/System.out(15145): empty line
01-24 06:51:55.131: I/System.out(15145): OK!!!
01-24 06:51:55.131: I/System.out(15145): empty line
01-24 06:51:55.131: I/System.out(15145): {result=true}
01-24 06:51:55.131: I/System.out(15145): recieve length: 6
01-24 06:51:55.141: I/System.out(15145): parsed command: null
01-24 06:51:55.141: I/System.out(15145): null: true!!!!
01-24 06:51:55.141: I/System.out(15145): AT+CHLD=?
01-24 06:51:55.191: I/System.out(15145): 26
01-24 06:51:55.211: I/System.out(15145): received:
01-24 06:51:55.211: I/System.out(15145): +CHLD: (0,1,2,3)
01-24 06:51:55.221: I/System.out(15145):
01-24 06:51:55.221: I/System.out(15145): OK
01-24 06:51:55.221: I/System.out(15145): empty line
01-24 06:51:55.231: I/System.out(15145): command: +CHLD
01-24 06:51:55.231: I/System.out(15145): parameter: (0,1,2,3)
01-24 06:51:55.231: I/System.out(15145): empty line
01-24 06:51:55.231: I/System.out(15145): OK!!!
01-24 06:51:55.231: I/System.out(15145): empty line
01-24 06:51:55.241: I/System.out(15145): {command=+CHLD, resul
t=true, parameter=(0,1,2,3)}
01-24 06:51:55.251: I/System.out(15145): recieve length: 26
01-24 06:51:55.251: I/System.out(15145): parsed command: +CHLD
01-24 06:51:55.251: I/System.out(15145): +CHLD: true!!!!
01-24 06:51:55.261: I/System.out(15145): AT+CLIP=1
01-24 06:51:55.301: I/System.out(15145): 6
01-24 06:51:55.321: I/System.out(15145): received:
01-24 06:51:55.321: I/System.out(15145): OK
01-24 06:51:55.331: I/System.out(15145): empty line
01-24 06:51:55.331: I/System.out(15145): OK!!!
01-24 06:51:55.331: I/System.out(15145): empty line
01-24 06:51:55.331: I/System.out(15145): {result=true}
01-24 06:51:55.331: I/System.out(15145): recieve length: 6
01-24 06:51:55.341: I/System.out(15145): parsed command: null
01-24 06:51:55.341: I/System.out(15145): null: true!!!!
01-24 06:51:55.341: I/System.out(15145): AT+CNUM
```

```
01-24 06:51:55.381: I/System.out(15145): 38
01-24 06:51:55.401: I/System.out(15145): received:
01-24 06:51:55.401: I/System.out(15145): +CNUM: ,"09055433140"
,129,,4
01-24 06:51:55.401: I/System.out(15145):
01-24 06:51:55.401: I/System.out(15145): OK
01-24 06:51:55.411: I/System.out(15145): empty line
01-24 06:51:55.421: I/System.out(15145): command: +CNUM
01-24 06:51:55.421: I/System.out(15145): parameter: ,"09055433
140",129,,4
01-24 06:51:55.421: I/System.out(15145): empty line
01-24 06:51:55.421: I/System.out(15145): OK!!!
01-24 06:51:55.421: I/System.out(15145): empty line
01-24 06:51:55.441: I/System.out(15145): {command=+CNUM, resul
t=true, parameter=,"09055433140",129,,4}
01-24 06:51:55.441: I/System.out(15145): recieve length: 38
01-24 06:51:55.441: I/System.out(15145): parsed command: +CNUM
01-24 06:51:55.441: I/System.out(15145): +CNUM: true!!!!
01-24 06:51:55.451: I/System.out(15145): 7 > 7
```

A.4 実装したダイヤラを通しての発信

```
01-24 06:52:11.711: I/System.out(15145): send AT <DIAL>
01-24 06:52:11.721: I/System.out(15145): send intent -- 090794
95274
01-24 06:52:11.751: I/System.out(15145): ATD09079495274;
01-24 06:52:12.941: I/System.out(15145): 20
01-24 06:52:12.961: I/System.out(15145): received:
01-24 06:52:12.961: I/System.out(15145): OK
01-24 06:52:12.961: I/System.out(15145):
01-24 06:52:12.961: I/System.out(15145): +CIEV: 3,2
01-24 06:52:12.971: I/System.out(15145): empty line
01-24 06:52:12.971: I/System.out(15145): OK!!!
01-24 06:52:12.971: I/System.out(15145): empty line
01-24 06:52:12.981: I/System.out(15145): command: +CIEV
01-24 06:52:12.981: I/System.out(15145): parameter: 3,2
01-24 06:52:12.981: I/System.out(15145): empty line
01-24 06:52:12.991: I/System.out(15145): {command=+CIEV, resul
t=true, parameter=3,2}
01-24 06:52:18.691: I/System.out(15145): 14
01-24 06:52:18.711: I/System.out(15145): received:
```

```
01-24 06:52:18.711: I/System.out(15145): +CIEV: 3,3
01-24 06:52:18.711: I/System.out(15145): empty line
01-24 06:52:18.721: I/System.out(15145): command: +CIEV
01-24 06:52:18.731: I/System.out(15145): parameter: 3,3
01-24 06:52:18.731: I/System.out(15145): empty line
01-24 06:52:18.731: I/System.out(15145): {command=+CIEV, param
eter=3,3}
01-24 06:52:25.591: I/System.out(15145): send AT <ON_HOOK>
01-24 06:52:25.651: I/System.out(15145): AT+CHUP
01-24 06:52:25.721: I/System.out(15145): 6
01-24 06:52:25.721: I/System.out(15145): received:
01-24 06:52:25.721: I/System.out(15145): OK
01-24 06:52:25.741: I/System.out(15145): empty line
01-24 06:52:25.741: I/System.out(15145): OK!!!
01-24 06:52:25.741: I/System.out(15145): empty line
01-24 06:52:25.741: I/System.out(15145): {result=true}
01-24 06:52:25.751: I/System.out(15145): 14
01-24 06:52:25.761: I/System.out(15145): received:
01-24 06:52:25.771: I/System.out(15145): +CIEV: 3,0
01-24 06:52:25.771: I/System.out(15145): empty line
01-24 06:52:25.771: I/System.out(15145): command: +CIEV
01-24 06:52:25.771: I/System.out(15145): parameter: 3,0
01-24 06:52:25.771: I/System.out(15145): empty line
01-24 06:52:25.781: I/System.out(15145): {command=+CIEV, param
eter=3,0}
01-24 06:52:25.781: D/ScoSocket(15145): android.bluetooth.ScoS
ocket@449e5f40 SCO OBJECT close() mState = 2
01-24 06:52:25.781: E/BTL_IFC(15145): ##### ERROR : tx_data: w
rite failed (-1)#####
01-24 06:52:25.781: E/BTL_IFC(15145): ##### ERROR : BTL_IFC_Ct
rlSend: [BTL_IFC CTRL] send failed#####
01-24 06:52:25.781: E/ScoSocket(15145): [BTUI] [### SCO ###] d
oClose()... mState(2) [1:READY/2:ACCEPT/3:CONNECTING/4:CONNECT
ED/5:CLOSED]
01-24 06:52:25.791: D/ScoSocket(15145): android.bluetooth.ScoS
ocket@449e62e8 SCO OBJECT close() mState = 1
01-24 06:52:25.791: E/BTL_IFC(15145): ##### ERROR : tx_data: w
rite failed (-1)#####
01-24 06:52:25.791: E/BTL_IFC(15145): ##### ERROR : BTL_IFC_Ct
rlSend: [BTL_IFC CTRL] send failed#####
01-24 06:52:25.791: E/ScoSocket(15145): [BTUI] [### SCO ###] d
oClose()... mState(1) [1:READY/2:ACCEPT/3:CONNECTING/4:CONNECT
```

```
ED/5:CLOSED]
01-24 06:52:25.811: E/ScoSocket(15145): [BTUI] [### SCO ###] d
oAccept()... mState(2) [1:READY/2:ACCEPT/3:CONNECTING/4:CONNEC
TED/5:CLOSED]
01-24 06:52:25.811: I/System.out(15145): sco socket accept.
01-24 06:52:25.811: I/System.out(15145): sco off <fake>.
```

A.5 HFP 接続中の着信

```
01-24 06:53:27.061: I/System.out(15145): 50
01-24 06:53:27.081: I/System.out(15145): received:
01-24 06:53:27.081: I/System.out(15145): +CIEV: 3,1
01-24 06:53:27.081: I/System.out(15145):
01-24 06:53:27.091: I/System.out(15145): RING
01-24 06:53:27.091: I/System.out(15145):
01-24 06:53:27.091: I/System.out(15145): +CLIP: "09079495274",
129
01-24 06:53:27.101: I/System.out(15145): empty line
01-24 06:53:27.101: I/System.out(15145): command: +CIEV
01-24 06:53:27.111: I/System.out(15145): parameter: 3,1
01-24 06:53:27.111: I/System.out(15145): empty line
01-24 06:53:27.111: I/System.out(15145): unknown: RING
01-24 06:53:27.111: I/System.out(15145): empty line
01-24 06:53:27.121: I/System.out(15145): command: +CLIP
01-24 06:53:27.121: I/System.out(15145): parameter: "090794952
74",129
01-24 06:53:27.121: I/System.out(15145): empty line
01-24 06:53:27.131: I/System.out(15145): {command=+CLIP, unkno
wn=RING, parameter="09079495274",129}
01-24 06:53:27.151: I/System.out(15145): send intent -- "09079
495274"
01-24 06:53:27.161: I/System.out(15145): get intent -- Intent
{ act=TELEPHONE_NUMBER (has extras) }
01-24 06:53:29.881: I/System.out(15145): 36
01-24 06:53:29.901: I/System.out(15145): received:
01-24 06:53:29.901: I/System.out(15145): RING
01-24 06:53:29.901: I/System.out(15145):
01-24 06:53:29.911: I/System.out(15145): +CLIP: "09079495274",
129
01-24 06:53:29.911: I/System.out(15145): empty line
01-24 06:53:29.921: I/System.out(15145): unknown: RING
```

```
01-24 06:53:29.921: I/System.out(15145): empty line
01-24 06:53:29.921: I/System.out(15145): command: +CLIP
01-24 06:53:29.931: I/System.out(15145): parameter: "090794952
74",129
01-24 06:53:29.931: I/System.out(15145): empty line
01-24 06:53:29.931: I/System.out(15145): {command=+CLIP, param
eter="09079495274",129, unknown=RING}
01-24 06:53:29.951: I/System.out(15145): send intent -- "09079
495274"
01-24 06:53:29.971: I/System.out(15145): get intent -- Intent
{ act=TELEPHONE_NUMBER (has extras) }
01-24 06:53:32.881: I/System.out(15145): 36
01-24 06:53:32.901: I/System.out(15145): received:
01-24 06:53:32.901: I/System.out(15145): RING
01-24 06:53:32.901: I/System.out(15145):
01-24 06:53:32.901: I/System.out(15145): +CLIP: "09079495274",
129
01-24 06:53:32.911: I/System.out(15145): empty line
01-24 06:53:32.911: I/System.out(15145): unknown: RING
01-24 06:53:32.911: I/System.out(15145): empty line
01-24 06:53:32.921: I/System.out(15145): command: +CLIP
01-24 06:53:32.921: I/System.out(15145): parameter: "090794952
74",129
01-24 06:53:32.921: I/System.out(15145): empty line
01-24 06:53:32.931: I/System.out(15145): {command=+CLIP, param
eter="09079495274",129, unknown=RING}
01-24 06:53:32.941: I/System.out(15145): send intent -- "09079
495274"
01-24 06:53:32.961: I/System.out(15145): get intent -- Intent
{ act=TELEPHONE_NUMBER (has extras) }
01-24 06:53:36.041: I/System.out(15145): 36
01-24 06:53:36.061: I/System.out(15145): received:
01-24 06:53:36.061: I/System.out(15145): RING
01-24 06:53:36.061: I/System.out(15145):
01-24 06:53:36.061: I/System.out(15145): +CLIP: "09079495274",
129
01-24 06:53:36.071: I/System.out(15145): empty line
01-24 06:53:36.071: I/System.out(15145): unknown: RING
01-24 06:53:36.071: I/System.out(15145): empty line
01-24 06:53:36.081: I/System.out(15145): command: +CLIP
01-24 06:53:36.081: I/System.out(15145): parameter: "090794952
74",129
```

```
01-24 06:53:36.081: I/System.out(15145): empty line
01-24 06:53:36.091: I/System.out(15145): {command=+CLIP, param
eter="09079495274",129, unknown=RING}
01-24 06:53:36.111: I/System.out(15145): send intent -- "09079
495274"
01-24 06:53:36.131: I/System.out(15145): get intent -- Intent
{ act=TELEPHONE_NUMBER (has extras) }
01-24 06:53:38.901: I/System.out(15145): 36
01-24 06:53:38.921: I/System.out(15145): received:
01-24 06:53:38.921: I/System.out(15145): RING
01-24 06:53:38.921: I/System.out(15145):
01-24 06:53:38.921: I/System.out(15145): +CLIP: "09079495274",
129
01-24 06:53:38.931: I/System.out(15145): empty line
01-24 06:53:38.931: I/System.out(15145): unknown: RING
01-24 06:53:38.931: I/System.out(15145): empty line
01-24 06:53:38.941: I/System.out(15145): command: +CLIP
01-24 06:53:38.941: I/System.out(15145): parameter: "090794952
74",129
01-24 06:53:38.941: I/System.out(15145): empty line
01-24 06:53:38.951: I/System.out(15145): {command=+CLIP, param
eter="09079495274",129, unknown=RING}
01-24 06:53:38.971: I/System.out(15145): send intent -- "09079
495274"
01-24 06:53:38.981: I/System.out(15145): get intent -- Intent
{ act=TELEPHONE_NUMBER (has extras) }
01-24 06:53:40.151: I/System.out(15145): 14
01-24 06:53:40.171: I/System.out(15145): received:
01-24 06:53:40.171: I/System.out(15145): +CIEV: 3,0
01-24 06:53:40.181: I/System.out(15145): empty line
01-24 06:53:40.191: I/System.out(15145): command: +CIEV
01-24 06:53:40.191: I/System.out(15145): parameter: 3,0
01-24 06:53:40.191: I/System.out(15145): empty line
01-24 06:53:40.191: I/System.out(15145): {command=+CIEV, param
eter=3,0}
01-24 06:53:40.201: D/ScoSocket(15145): android.bluetooth.ScoS
ocket@44a132d0 SCO OBJECT close() mState = 2
01-24 06:53:40.211: E/BTL_IFC(15145): ##### ERROR : tx_data: w
rite failed (-1)#####
01-24 06:53:40.211: E/BTL_IFC(15145): ##### ERROR : BTL_IFC_Ct
rlSend: [BTL_IFC CTRL] send failed#####
01-24 06:53:40.211: E/ScoSocket(15145): [BTUI] [### SCO ###] d
```

```
oClose()... mState(2) [1:READY/2:ACCEPT/3:CONNECTING/4:CONNECT
ED/5:CLOSED]
01-24 06:53:40.221: D/ScoSocket(15145): android.bluetooth.ScoS
ocket@449b3ab0 SCO OBJECT close() mState = 1
01-24 06:53:40.221: E/BTL_IFC(15145): ##### ERROR : tx_data: w
rite failed (-1)#####
01-24 06:53:40.221: E/BTL_IFC(15145): ##### ERROR : BTL_IFC_Ct
rlSend: [BTL_IFC CTRL] send failed#####
01-24 06:53:40.231: E/ScoSocket(15145): [BTUI] [### SCO ###] d
oClose()... mState(1) [1:READY/2:ACCEPT/3:CONNECTING/4:CONNECT
ED/5:CLOSED]
01-24 06:53:40.271: E/ScoSocket(15145): [BTUI] [### SCO ###] d
oAccept()... mState(2) [1:READY/2:ACCEPT/3:CONNECTING/4:CONNEC
TED/5:CLOSED]
01-24 06:53:40.271: I/System.out(15145): sco socket accept.
01-24 06:53:40.271: I/System.out(15145): sco off <fake>.
```

A.6 既製品の HFP HF 機器で SCO-S リンクが成功している例

```
D/InCallScreen(1495): ============== dumpBluetoothState() =============
D/InCallScreen(1495): isBluetoothAvailable()...
D/InCallScreen(1495):   - headset state = 2
D/InCallScreen(1495):   - headset address: 00:1B:DC:1D:A3:EA
D/InCallScreen(1495):   - isConnected: true
D/InCallScreen(1495):   ==> true
D/InCallScreen(1495): = isBluetoothAvailable: true
D/InCallScreen(1495): isBluetoothAudioConnected: ==> isAudioOn = false
D/InCallScreen(1495): = isBluetoothAudioConnected: false
D/InCallScreen(1495): isBluetoothAudioConnected: ==> isAudioOn = false
D/InCallScreen(1495): isBluetoothAudioConnectedOrPending: ==> FALSE
D/InCallScreen(1495): = isBluetoothAudioConnectedOrPending: false
D/InCallScreen(1495): = PhoneApp.showBluetoothIndication: true
D/InCallScreen(1495): =
D/InCallScreen(1495): = BluetoothHandsfree.isAudioOn: false
D/InCallScreen(1495): = BluetoothHeadset.getCurrentHeadset: 00:1B:DC:1D:A3:EA
D/InCallScreen(1495): = BluetoothHeadset.isConnected: true
D/InCallScreen(1495): syncWithPhoneState: it's ok to be here; update the screen...
```

（略）

```
D/InCallScreen(1495): isBluetoothAvailable()...
D/InCallScreen(1495):   - headset state = 2
D/InCallScreen(1495):   - headset address: 00:1B:DC:1D:A3:EA
D/InCallScreen(1495):   - isConnected: true
D/InCallScreen(1495):   ==> true
D/InCallScreen(1495): isBluetoothAudioConnected: ==> isAudioOn = false
D/InCallScreen(1495): isBluetoothAudioConnectedOrPending: ==> FALSE
D/InCallScreen(1495): [okToDialDTMFTones] foreground state: AC
TIVE, ringing state: false, call screen mode: NORMAL, result: true
D/InCallControlState(1495): InCallControlState:
D/InCallControlState(1495):   manageConferenceVisible: false
D/InCallControlState(1495):   manageConferenceEnabled: false
D/InCallControlState(1495):   canAddCall: true
D/InCallControlState(1495):   canSwap: false
D/InCallControlState(1495):   canMerge: false
D/InCallControlState(1495):   bluetoothEnabled: true
D/InCallControlState(1495):   bluetoothIndicatorOn: false
D/InCallControlState(1495):   speakerEnabled: true
D/InCallControlState(1495):   speakerOn: false
D/InCallControlState(1495):   canMute: true
D/InCallControlState(1495):   muteIndicatorOn: false
D/InCallControlState(1495):   dialpadEnabled: true
D/InCallControlState(1495):   dialpadVisible: false
D/InCallControlState(1495):   onHold: false
D/InCallControlState(1495):   canHold: true
D/InCallScreen(1495): updateProviderOverlay: false
D/InCallScreen(1495): updateMenuButtonHint()...
D/PhoneApp(1495): setScreenTimeout(MEDIUM)...
D/PhoneApp(1495): requestWakeState(SLEEP)...
D/PowerManagerService(1398): ignoring user activity while turning off screen
V/BTLD_AG(1634): btapp_ag_cback event=5, handle=1 app_id=0
I/BTL-IFS(1634): send_ctrl_msg: [BTL_IFS CTRL] send BTLIF_CONN
ECT_RSP (SCO) 0 pbytes (hdl 24)
E/ScoSocket(1495): [BTUI] [### SCO ###] onConnected()... mStat
e(4) [1:READY/2:ACCEPT/3:CONNECTING/4:CONNECTED/5:CLOSED]
```

A.7 本研究の実装で通話に応答したが SCO-S リンクが成功しなかった例

```
D/InCallScreen( 1499): ============== dumpBluetoothState() =============
D/InCallScreen( 1499): isBluetoothAvailable()...
```

```
W/BluetoothHeadset( 1499): Proxy not attached to service
D/InCallScreen( 1499):   - headset state = -1
W/BluetoothHeadset( 1499): Proxy not attached to service
D/InCallScreen( 1499):   - headset address: null
D/InCallScreen( 1499):   ==> false
D/InCallScreen( 1499): = isBluetoothAvailable: false
D/InCallScreen( 1499): isBluetoothAudioConnected: ==> isAudioOn = false
D/InCallScreen( 1499): = isBluetoothAudioConnected: false
D/InCallScreen( 1499): isBluetoothAudioConnected: ==> isAudioOn = false
D/InCallScreen( 1499): isBluetoothAudioConnectedOrPending: ==> FALSE
D/InCallScreen( 1499): = isBluetoothAudioConnectedOrPending: false
D/InCallScreen( 1499): = PhoneApp.showBluetoothIndication: true
D/InCallScreen( 1499): =
D/InCallScreen( 1499): = BluetoothHandsfree.isAudioOn: false
W/BluetoothHeadset( 1499): Proxy not attached to service
D/InCallScreen( 1499): = BluetoothHeadset.getCurrentHeadset: null
D/InCallScreen( 1499): syncWithPhoneState: it's ok to be here; update the screen...
```