携帯電話を用いた視覚障害者のための生活支援システムの提案

守屋, 志保 / MORIYA, Shiho

(発行年 / Year)
2008-03-24
(学位授与年月日 / Date of Granted)
2008-03-24
(学位名 / Degree Name)
修士(理学)
(学位授与機関 / Degree Grantor)
法政大学 (Hosei University)

携帯電話を用いた視覚障害者のための生活支援システムの提案

Support System for the Blind using Mobile Phone with RFID

Shiho Moriya

Graduate School of Computer and Information Sciences,
Hosei University,
Koganei-shi, Tokyo 184-8584, Japan
shiho.moriya.cv@gs-cis.hosei.ac.jp

Abstract

In recent decades, life support system for blind has not been standardized yet. Further, the number of the existing systems is also very small. Even if it exists, it's very poor. We aimed at this problem. A cellular phone is used as a handheld unit in this system. Doe to it's widely use and popularity. Application and RFID reader are loaded into a cellular phone. Loaded reader reads a RFID tag. And the information stocked in a tag is acquired. Sound associated to the acquired information is played. The blind hear the sound, and know what situation are in now. Doja is used as a programming language. When connecting to a network, Doja reduces time to wait. Data is sent and received to store in a scratchpad. One of data corresponds to One separated area. Once data is stored in a scratchpad, the probability that application connects to a network next time might become short. To achieve this, two IDs are used to express an object. Consequently, time of connecting to a network becames minimal. It is achieved that the user hears sound quickly when reading a tag. This system can achieve to make the life of the blind convenient and comfortable.

Supervisor: Runhe Huang, Professor

1. 序論

今日、「ユニバーサルデザイン」「バリアフリー」などという単語をよく耳にする。「ユニバーサルデザイン」（Universal Design、UD と略記することもある）とは、文化・言語の違い、老若男女といった差異、障害・能力の如何を問わずに利用することができる施設・製品・情報の設計（デザイン）をいう。一方「バリアフリー」(Barrier free)とは、広義の対象者としては障害者を含む高齢者等の社会生活弱者、狭義の対象者としては障害者が社会生活に参加する上で生活の支障となる物理的な障害（障碍）や精神的な障壁を取り除くための施策、若しくは具体的に障害を取り除いた状態をいう。一般的には障害者が利用する上での障壁が取り除かれた状態として広く使われている。

世界的な傾向としてより多くの人に使いやすいものを、という概念が広まりつつあるが、この傾向が情報科学の分野でも見られているのだろうか。あるいは見られていたとしても、有効なシステムが実現されているのだろうか。実際のところ、情報弱者が使いやすいデザインは広くは普及していない。多くのシステムは、健常者向け、コアユーザー向けとして発展を続けている。情報技術を活用できる層と情報弱者の間に社会的・経済的格差が生じ、あるいは格差が拡大していく現象を「デジタルデバイド」という。2000 年 7 月の九州・沖縄サミットではデジタルデバイドが議題に挙げられ、情報弱者への支援とデジタルデバイドの克服が重要課題とされたが、未だにこの動きは活発にはなってはいない。

本論文では前述のようなデジタルデバイドの克服の一環としてのシステムを提案する。

1.1. 視覚障害者について

現在、厚生労働省による統計[1]では視覚障害者は全国で３０万人ほどいるとされている。この数字は日本の人口の約 0.2%であり、視覚障害者は生活上で多くの支障を抱えている。身体障害は、障害の程度に応じて 1 級から 6 級までである。視覚障害は「視力障害(１)」と「視野障害（２）」とに区分して認定する。重複する場合、重複障害認定の原則に基づき認定する。以下に視覚障害者の定義を示す。

1 級

両眼の視力（万国式試視力表によって測ったものをいい、 屈折異常のある者についてについては、矯正視力について測ったものをいう。以下同じ）の和が 0.01 以下のもの。

2 級

(1)両眼の視力の和が 0.02 以上 0.04 以下のもの。

(2)両眼の視野がそれぞれ 10 度以内でかつ両眼による視野について視能率による損失率が 95%以上のもの。

3 級

(1)両眼の視力の和が 0.05 以上 0.08 以下のもの。

(2)両眼の視野がそれぞれ10度以内でかつ両眼による視野について視能率による損失率が90%以上のもの。

4級

(1)両眼の視力の和が0.09以上0.12以下のもの。

(2)両眼の視野がそれぞれ10度以内のもの 。

5級

(1)両眼の視力の和が0.13以上0.2以下のもの。

(2)両眼による視野の2分の1以上がかけているもの。

6級

一眼の視力が0.02以下、他眼の視力が0.6以下のもので、両眼の視力の和が0.2を超えるもの。

これらの視覚障害の原因は、最も多いもので糖尿病である。次いで、緑内障などが続く。先天的、後天的かどうかはこの等級の認定には影響を及ぼさない。よってどんな人間でも視覚障害者になるリスクを負っている。

視覚障害者が抱える問題は多岐に渡る。人間の感覚の中で、得る情報の最も多くを占めるとされるものが視覚であり、これに頼ることができない視覚障害者は生活においてさまざまな工夫を強いられる事は明らかである。生活においての工夫により、広く使われるようになった道具は多くある。白杖、点字ブロック、点字、PCの音声読み上げソフト、シャンプーのボトルデザイン、そして最近では音声を再生する案内板が駅構内や公的機関に設置されている事もしばしばある。本論では、視覚障害者の生活を情報技術を用いて統合的に支援するシステムを提唱する。

1.2. 国内における視覚障害者のための取り組み

主に取り組みを行っている団体で有名なものは「日本盲人会連合[2]」である。日本盲人連合会は視覚障害者自身の全国組織である。日本盲人会連合は視覚障害者自身の手で、“自立と社会参加”を実現しようと組織された視覚障害者の全国組織であり、１９４８年（昭和２３年）に結成された。都道府県・政令指定都市における５９の視覚障害者団体の連合体で、国や地方自治体の視覚障害者政策－人権、福祉、教育、職業、環境問題等－の立案・決定に際し、視覚障害者のニーズを反映させるため、陳情や要求運動を行っている。これらのニーズとは情報技術に限ったものではなく、生活の範囲全般にわたるものである。

情報技術に際して述べると、平成12年5月17日に交付された交通バリアフリー法をはじめ特に歩行者ITS（Intelligent Transport system）が、21世紀のバリアフリー道路システムとして歩行空間における歩行者、車椅子使用者、自転車などの安全性、快適性、利便性の向上を実現化させるものとして注目されている。この歩行ITSにおいては、GPSシステム

などが積極的に取り入れられている。GPS を用いたシステムは現在積極的な研究対象とされており、静岡大学の石川 准、兵藤 安昭らによる” GPS による視覚障害者歩行支援システムの開発” [3]においても歩行 ITS が研究されている。このシステムは PC 上に搭載され、自分の歩く予定の目的地までの道筋を GPS によって解析したり、自分が現在いる場所周辺の情報を取得したりするシステムである。加えて、ユーザに GPS 装置をとりつけ、ログを取得することによりユーザが歩いた道筋を PC 上で確認することもできる。これはユーザが移動した後、自分がどこを通ったかを確認し、最適な道筋を自分で認識することを補助する。なお、” GPS による視覚障害者歩行支援システムの開発”は PC 上で動作するシステムであるが、近年携帯電話で実装されるシステムも現れてきている。新潟大学大学院の檜垣 宏行、牧野 秀夫らの” 視覚障害者用音声位置案内システムにおける GPS 携帯電話・PDA の実験と評価”によれば、GPS 機能を搭載した携帯電話をシステムの実行端末として使う事が提案、機能の評価にまで至っている[4][5]。

その他の取り組みとしては、” 自律的移動支援プロジェクト推進委員会”による自立支援プロジェクト[6]等も発足している。これはすべての人が持てる力を発揮し、支え合って構築する「ユニバーサル社会」の実現に向けた取り組みの一環として、社会参画や就労などにあたって必要となる「移動経路」、「交通手段」、「目的地」などの情報について、「いつでも、どこでも、だれでも」がアクセスできる環境をつくっていくための検討を行うことを目的としている委員会である。

このように、情報技術を取り入れたものも取り組みの中には多数発足しているが、規格が統一されておらず普及には至らないものが多数を占めているというのが現状である。なお、総務省による「ユビキタスネットワーク時代における電子タグの高度利活用に関する調査研究会」などで医療・福祉に関するタグの使用が検討されたり[7]、JISC 日本工業標準調査会[8]によって規格統一案が提示されたりしてはいるが、こちらも規格統一までは至っていない。

1.3. 視覚障害者のためのシステム例

現在では財団法人機械システム振興協会[9]により、ＩＣタグ生活支援システムが提案されている。これはタグを製品にとりつけ、利用者が専用端末であるリーダを持ち、タグを読み取った際に音声で製品情報を知らせるというものである。使用機器はスピーカー付 PC、ハンディタイプリーダライター、そしてタグがついている製品である。利用者はハンディタイプリーダライターを手に持ち、読んだタグ情報はリーダから無線 LAN を通じて PC と情報をやりとりする。そして PC に付属しているスピーカーから音声が再生される。よって PC が近くにある環境で使うと想定されている。

しかしこのシステムにおいて音声は PC に格納されており、音声の数には限りがある。そして音声が再生されるまでに５～６秒を要するなどの改善するべき点が多く見られる。

このシステムにおいてメリットとデメリットを挙げると以下のようになる。

メリット

・小さなタグを使うことによってさまざまな物に情報を付加する事ができる

デメリット

- 音声の再生時間が長い（５～６秒）
- 専用の携帯端末を持たなくてはいけない
- 専用の携帯端末のサイズが大きい
- PC が近くにある環境でないと音声を聞くことができない

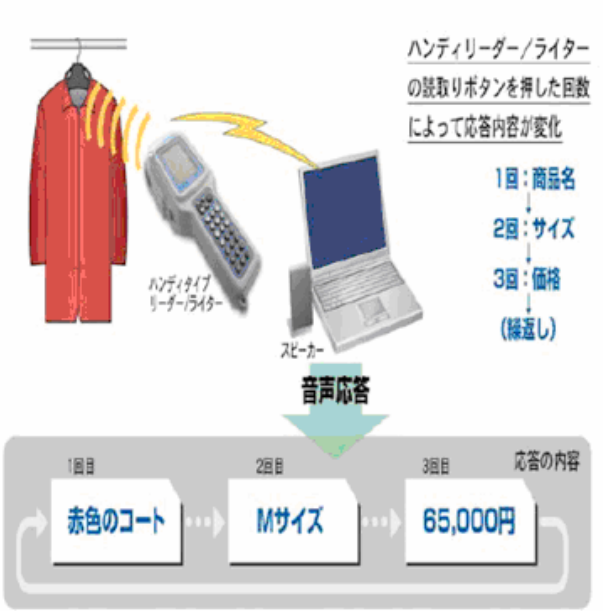


図 1. 機械システム振興会によるシステム概要

また、松下電器産業はＩＣタグを利用した音声レコーダー「物知りトーク」を開発した[10]。このシステムは身のまわりの物の情報を音声で視覚障害者に知らせる。薬局で薬の服用方法を音声レコーダーに録音する。これは IC タグに音声が格納されるわけではない。音声データと IC タグの少量のデータをマッチングさせるというシステムである。薬袋にＩＣタグをとりつけ録音内容を登録すれば、薬を服用する際ＩＣタグに音声レコーダーを近づけてその録音を聞くことが出来る。ＩＣタグから発信される微弱な電波の信号を音声レコーダーがキャッチし、予め録音された音声を再生する仕組みである。タグを対称物に付けて、その前方でリーダを近付けてまず「読む」というボタンを押し、その後「録音」のボタンを押しながら「血圧の薬」などと言うと、「血圧の薬」と記憶される。指を離すと復唱する。また、その他にメモも録音である。「調理方法」や「薬の飲み方」等を説明された時、このリーダに録音しておいて、帰宅後タグに憶え込ませておけば、忘れることを

ふせぐ事ができる。変化する製品の情報を録音することで動的に格納しておくことが利点である。音声の容量は１０秒程度のものを約３００個保存できる。

ただし、松下のシステムにおいて、タグはすべて録音しておかなければならず利用者の負担になってしまう事が予想される。これは改善すべきデメリットであろう。

以下に「物知りトーク」のメリットとデメリットを挙げる。

メリット

・音声を自分で録音することができる

・タグの情報を自分で書き換える事が可能である

デメリット

・録音する音声の件数が多くなると利用者の負担になる

・値段が１セット 59,800＋税　と高い

このように既存のシステムは存在しているが、まだ規格もさまざまであり、統一されていない。またデメリットもとても多く、効率もしくは生産性のよいシステムがまだ存在していない。

またこれらのほかにも白杖とタグを組み合わせた歩行支援システム[11][12]や、北区／板橋区における地域資源活用型産業活性化プロジェクト[13]（KICC プロジェクト）がある。後者は北区・板橋区の中小企業の産業活性化の目的で、都、北区、板橋区が全面的に協調しあっている。このプロジェクトにおいては、視覚障害者移動支援のためのしくみづくりがなされており、主に屋外を歩行する視覚障害者にとってのバリアを軽減することがねらいとされている。

2. 関連技術

システムを構築するにあたって、既存の RFID を使うという姿勢を取り入れた。これはさまざまな物に取り付けるのには最適な為である。その他には携帯端末、音声再生端末として携帯電話、プログラム言語としては java を使用する。Java の大半は J2ME を使用している。なお、シミュレーションの便宜上、一部 J2SE をとりいれた。本章では RFID を含め、上記の使用する関連技術について述べる。

2.1. RFID

RFID とは、Radio Frequency Identification の略語で、電波による自動認識技術の総称である。RFID は対象物の認識（通信）に電波を使うため、直接（光学的には）見えない対象や移動する対象の認識などが可能である[14]。IC タグは RFID の一種であり、情報記憶媒体（メモリ）として IC チップを使うことにより、多くの種類のシステムの提案が可

能になる。システムにおいては IC チップに格納したデータをもとに、さまざまな機能を実装する。使用に際しての構造としては、IC タグリーダから発射される電波によって微量な電力が回路内に発生し、その電力で情報を処理し、リーダに送信する。大抵の場合、使用できる電波出力の関係などから、IC タグと IC タグリーダを近づける必要はあるが、必ずしも接触する必要はない。RFID にはパッシブタグ（受動タグ）とアクティブタグ（能動タグ）の 2 種類がある。

パッシブタグとは、タグリーダからの電波をエネルギー源として動作する RF タグで、電池を内蔵する必要がない。アクティブタグに比べてパッシブタグの受信距離は比較的短くなるが、安価に出来き、ほぼ恒久的に作動することから、広く普及されるであろうと予測される。タグリーダ側は、比較的強めの電波を供給し、タグからの非常に微弱な反射波を受信・解読できる必要がある。現在では、このパッシブタグが非常に安価（10 円以下）に生産できる見込みが出てきた。

アクティブタグは、電池を内蔵したタグである。自ら電波を発するので、通信距離が長い（10-100 メートル以上）。またセンサーを内蔵して、自発的にその変化を通知することが出来るので、センサーネットワークとしての用途が期待されている。

本システムでは、さまざまなものに取り付けるため、量が多くなってもコストがかさまないパッシブタグを使用する。

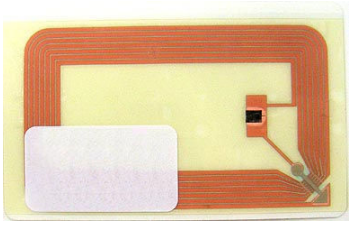


図 2. RFID の一例

2.2. J2ME

Sun Microsystems 社のプログラミング言語「Java 2」の機能セットの一つで、家電製品や携帯情報端末、携帯電話などの組み込み機器向けの機能をまとめたもの。「CDC」(Connected Device Configuration)と「CLDC」(Connected Limited Device Configuration)という 2 つの想定環境(コンフィギュレーション)に分かれている。前者はカーナビや高性能 PDA といった 32 ビット CPU と十分なメモリを持った環境を想定している。後者は携帯電話やネットワーク家電、通常の PDA などの、低速な CPU と少ないメモリからなる環境を対象としている。本システムで使用するのは CLDC コンフィギュレーションである。このコンフィギュレーションに、後述のプロファイルを付加させて使用する。

J2ME はサーバ向けの J2EE、パソコン向けの J2SE とは違い、仮想マシン(実行環境)を核に、最小限のコア API(中核機能)だけを付加し、デバイス(機器)の種類ごとに定義された「プロファイル」と呼ばれる API やクラスライブラリを更に付加していくことで機能を補うという方式を採用している。プロファイルの例としては、携帯電話や通信機能を持った携帯情報端末向けに定められた CLDC 用の MIDP や NTTdocomo 製端末用の Doja などがあり、これらは Sun Community Process を通して業界ごとに定義されている。また、CLDC が共通で使われていたとしても実行する機種によって依存する機能が非常に多い。よって注意が必要とされる。J2ME で使用されるコンフィギュレーション、およびプロファイルの構造を図 3 に示す。

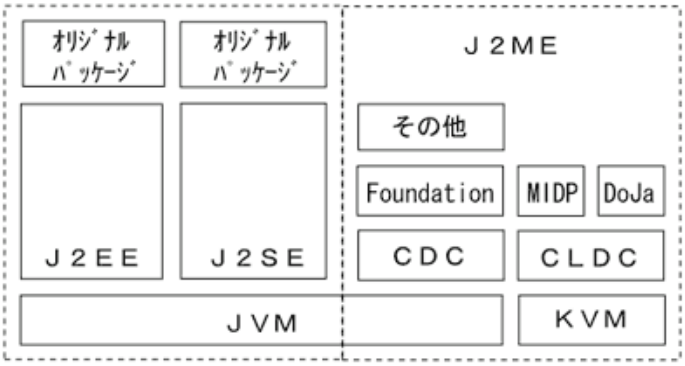


図 3. J2ME の構造

3. マイアプローチ

3.1. 現在の技術、システムの問題点

1.3 であげたように、視覚障害者のニーズに十分に対応するシステムは現在あまり普及していないのが現状である。GPS を用いた移動支援システムは多く提唱されているが、その他の分野では情報技術が未開拓である。タグを用いたシステムにおいても、ネットワークが十分に活用されていない。

タグを用いたシステムにおいても、情報は PC とハンディツールのみでのやりとりにとどまり、タグがネットワークシステムを介して使用されていない事がわかる。またローカルでやりとりしているにもかかわらず時間も多くかかる。加えて、タグと紐付けされている音声はローカルの PC 上にあるため、音声情報の更新は不可能であり、ユーザ間での共有などは実質されていない。ハンディツールにおいても独自の規格で開発されており、片手がふさがる大きさのためユーザがわざわざこれを持ち歩くことにより不便を生じる可能性が高い。

3.2. システム概要

私たちは、これまで述べてきた視覚障害者の生活の不便さと情報技術の浸透の少なさに着目した。視覚障害者の生活に密接にかかわるシステムを提案する。あらゆるものに RFID タグをつけ、タグ情報に対応した音声情報をネットワークを介して端末に保存し、再生するというシステムである。端末には携帯電話を用いることにより、ユーザに余計な持ち物を持たせない事とした。

端末に保存した音声情報は新たに音声セットをダウンロード、上書きをされるまでは永続的に保持される。これにより端末がネットワークを用いて音声セットを頻繁にダウンロードする必要が無くなる。ダウンロードする回数の少なさは、端末の電池消費量節約、音声の再生時間の短縮に貢献する。システムを大まかに表すと図 4 のようになる。

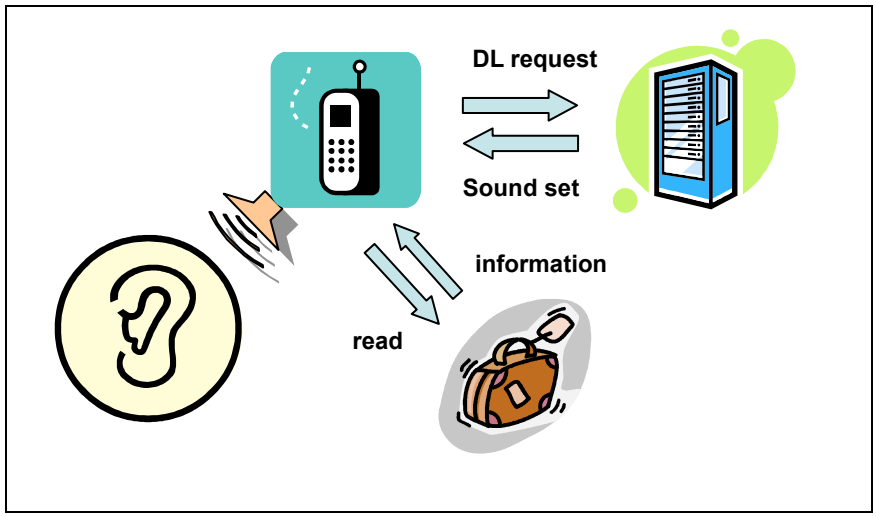


図 4. 提案するシステム概要

3.2.1. システム構築

本システムは、携帯電話、音声セットのストレージサーバ、RFID タグで構成される。携帯電話は RFID のリーダの機能と処理プログラムの格納、ダウンロードした音声セットの格納を行う。ストレージサーバにはさまざまな場所に対応した音声セットが格納されている。この音声セットはプログラム中で必要に応じてダウンロードされ、携帯電話に格納される。またサーバには音声セットは自由に追加可能である。

RFID には使用される場面をカテゴリ分けした ID（Place_ID　以下 P_ID と記す）とカテゴリされた場面においての製品唯一の ID（Object_ID　以下 O_ID と記す）が格納されている。各 ID は int 型で表現される。

システムのフローを以下に示す

① ユーザは携帯電話を用いて、物についた RFID タグを読み取る。携帯電話に搭載されたアプリケーションは RFID に格納された P_ID と O_ID を取得する

② アプリケーション中に保持している P_ID が読み込んだ P_ID と異なる場合、サーバから携帯電話に P_ID に対応する音声セットをダウンロードする。読み込んだ P_ID とダウンロードした音声セットは保持される。（P_ID が同じものであったばあい、新たな音声セットはダウンロードはされず、保持されている音声セットを使用する）

③ O_ID に応じた１つのファイルを音声セットから探し出し、音声を再生する

P_ID という概念を採用した理由として、タグを読み込む際に毎回ネットワークに接続して音声を再生する事がシステムの負荷となる事があげられる。ネットワークに接続する事は、すなわちアプリケーションにおいて処理時間が多くかかる事を意味する。よって、ネットワーク接続回数を最小限にする事を目的として、P_ID という概念を用いた。人が携帯電話を用いてタグを読み込む際には、移動しているとしても速度は遅く、または静止している状態であるかもしれない。よって物体が存在する区画を一区切りとし、その一区切りは機械的ではなく生活上の区切りとした。これによって、ユーザが家にいたならば、家にいる間はネットワーク接続は一回ですみ、たとえ家からどこかへ向かうとしても、頻繁にネットワーキングをする事はまれな事となる。

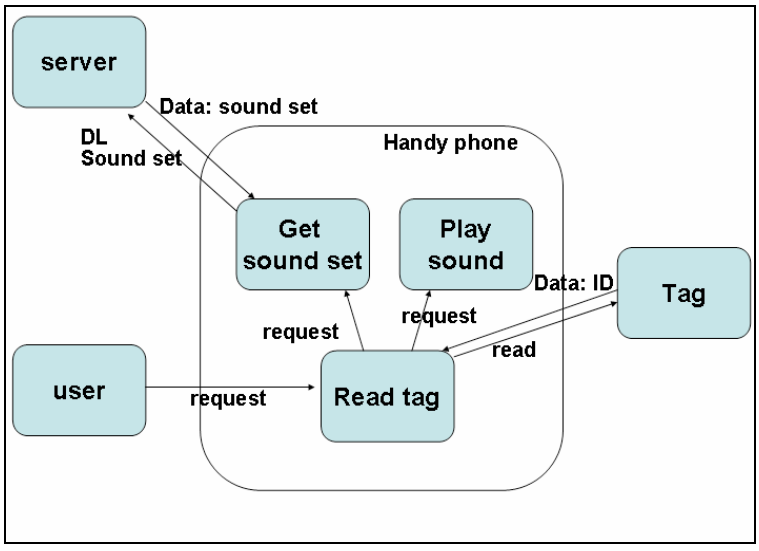


図 5. システムのユースケース図

図 5 にシステムのユースケース図を示した。大別して機能は３つあり、“read tag”、“play sound”、 “get sound set”となる。ユーザはまず“read tag”機能にアクセス

し、タグの情報を読み込む。“read tag”機能は読み込んだ P_ID が保持している P_ID と異なれば“get sound set”機能にリクエストを送る。“get sound set”機能はリクエストを受信すると、P_ID に対応した音声セットをダウンロードする。その後、“read tag”機能は“play sound”機能にリクエストをおくり、“play sound”機能が音声を再生する。

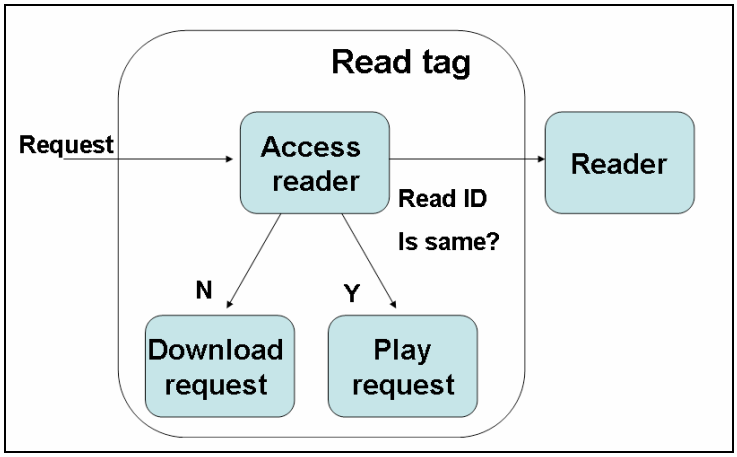


図 6. read tag 機能のユースケース図

“read tag”機能についての詳細を図 6 に表した。読んだ P_ID が保持しているものと違っているならばダウンロードリクエストを送る。同一ならばプレイリクエストを送る。

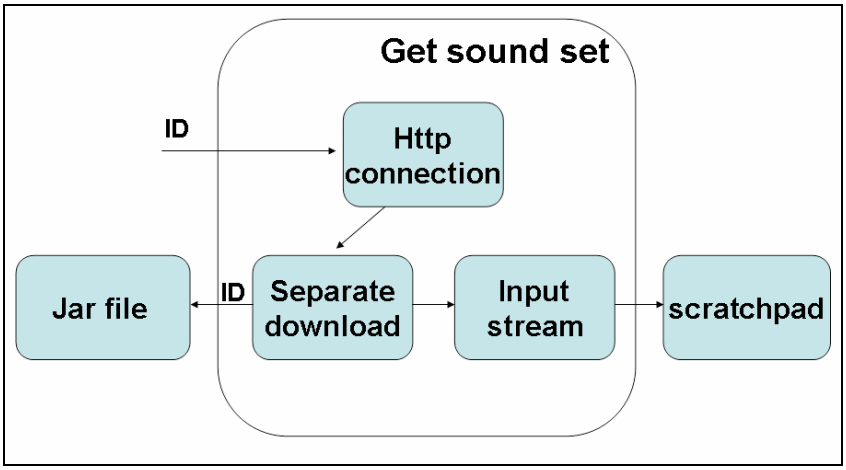


図 7. get sound set 機能のユースケース図

次に“get sound set”機能についての詳細を図 7 に表す。“get sound set”機能はリクエストを受信すると、P_ID に対応した音声セットをダウンロードする。この場合、一つの音声セットを一回の接続ではダウンロードできないため、何回かに分けてダウンロードする。その際に inputStream クラスを用いてスクラッチパッドに書き込んでいく。

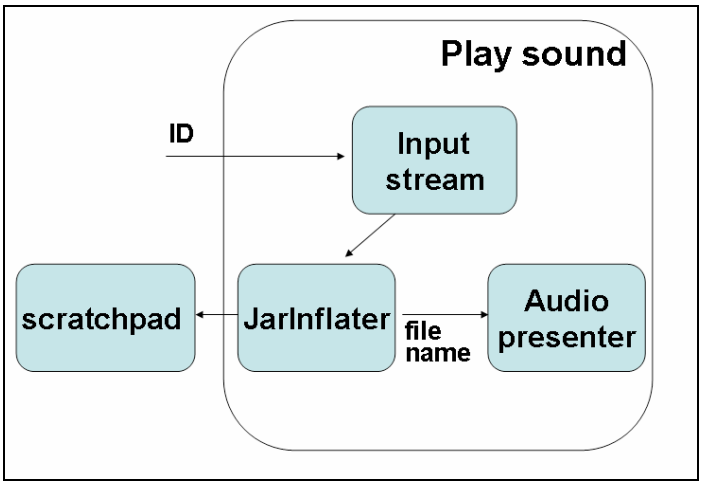


図 8. play sound 機能のユースケース図

最後に“get sound set”機能についての詳細を図 8 に表す。読み込んだ 0_ID とスクラッチパッドのデータからファイルの名前を割り出し、AudioPresenter によって音声が再生される。

なお、本システムは携帯電話にリーダライタが搭載されているという過程で提案するが、シミュレーションは PC 上で行うため RFID リーダを PC につなぎ、リーダアプリケーションとメインアプリケーションで通信することにより再現した。これを図 9 に示す。

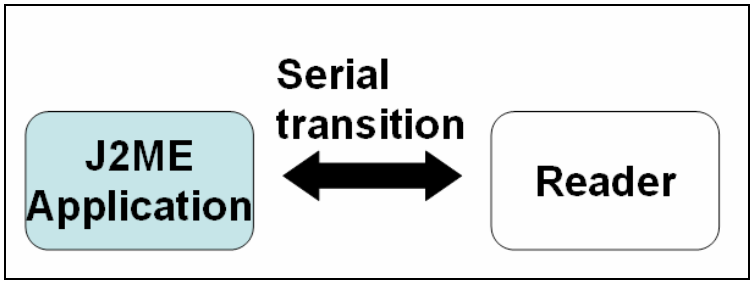


図 9. 提案する通信形態

MIDP においてはソケット通信が可能なため、リーダ機能を持つように見せたローカルホストサーバ、クライアントアプリケーション間はソケットでシリアル転送の代用をした。Doja においてはソケット通信が不可能であるため、PHP サーバを用いてリーダ機能のついたローカルホストサーバと PHP サーバは標準入出力を用いた。PHP サーバとクライアントアプリケーション間は HTTP 通信を用いて実現した。図 10、図 11 に構成の詳細を示す。

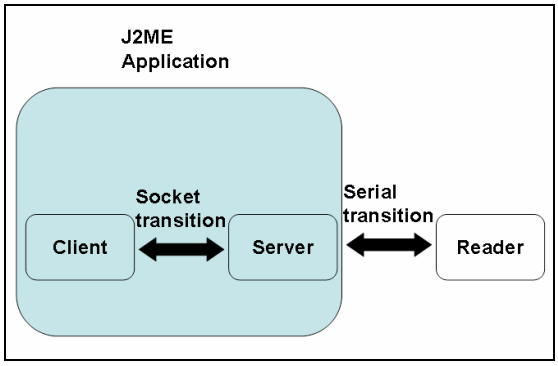


図 10. MIDP におけるシステム構成

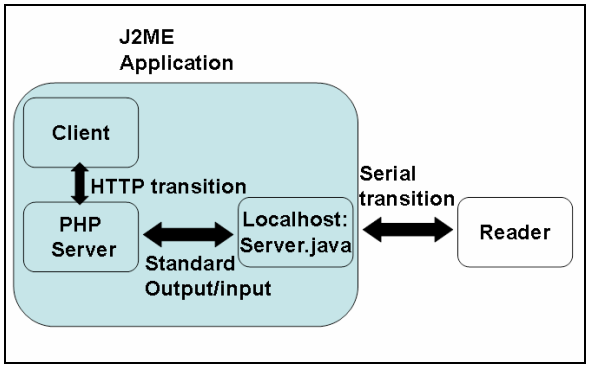


図 11. Doja におけるシステム構成

3.2.2. メリットとデメリット

本システムのメリットとして、ユーザが普段持っているとする携帯電話にリーダを搭載されていることをまず挙げる。詳細は後述する。まずソフト面でのメリットを詳細に挙げる。ソフト面ではメリットがユーザビリティ、スケーラビリティに大別される。以下に各面でのメリットをあげる。

・ ユーザビリティ

ユーザビリティに関しては２つメリットがある。

１つめにソフトキーの押下のみでタグを読むことが出来るため、視覚障害者にとって使いやすいインターフェースである。

２つめに１回音声セットをダウンロードすると、その周辺で読み込むであろうタグはひとまとまりにされているので、頻繁にダウンロードをする必要がない。そのため音声の再生時間が短くてすむ。

・ スケーラビリティ

スケーラビリティに関しても２つメリットがある。

１つめに格納されているデータがネットワーク上にあるため、製品情報などを追加する必要がある時に変更が容易である。

２つめにオブジェクト指向とされている JAVA が使用されているため、機能の追加、改変が容易である。

ハード面に関して言及すると、ユーザが普段から携帯電話を持ち歩いていることから、２点メリットがある。１つめは携帯電話であればユーザがキーの位置なども把握できており使い慣れているため、わざわざ新しく機械の使い方を覚える必要がない。２つめは Doja には赤外線機能、および Bluetooth 機能が搭載されているため、拡張すれば携帯電話から製

品情報を読み取るだけでなく、家電製品の操作も可能である。これは今後の課題として取り扱う。

デメリットとしては携帯電話を端末として使用している点から、使用する言語は J2ME であり、実装できる機能が制限される点があげられる。J2ME にもプロファイルがさまざまなものがあるが、今回は Doja を主に使用した。それはファイルをまとめて音声セットとする際に必要となるクラス（jarinflater）が Doja にのみ搭載されていたためである。また、通信に関しても厳しい制約があるため、大きいファイルを一度にダウンロードする、または本アプリケーションがアップされている以外のサーバへの接続などは不可能である。

3.3. ユーザーインターフェース

Doja の場合のユーザーインターフェースを以下に示す。図 12 における①のボタンが左ソフトキー、②のボタンが右ソフトキーである。アプリケーション再生中において使うキーは左ソフトキー、右ソフトキーのみである。左ソフトキーを押下するとアプリケーションが終了し、右ソフトキーを押下するとタグが読み込まれ音声が再生される。

MIDP の場合も同様である。キーによる動作は同様である。①が左ソフトキー、②が右ソフトキーである。右ソフトキーを押下するとタグを読み、対応する音声を再生する。左ソフトキーを押下するとアプリケーションが終了する。これを図 13 に表す。

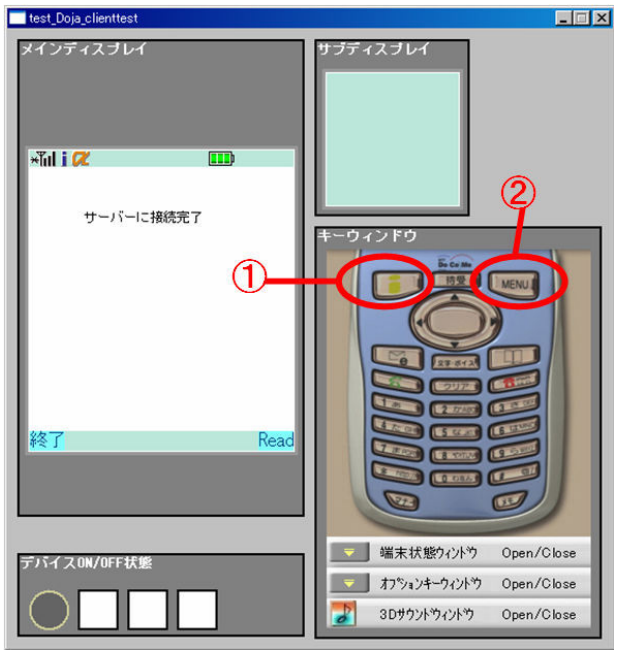


図 12. Doja によるアプリケーション UI

図 13. MIDP によるアプリケーション UI

3.4. ケーススタディ

ここに具体的なケーススタディを挙げる。ユーザは８０代女性、携帯電話を所持している。生活範囲はおもに家、家の周辺、公民館、最寄り駅、スーパーマーケットとする。タグはあらゆるものについていると仮定する。

以下の表に各施設内でのタグ ID とタグによって再生される音声を示した。

表 1. 自宅の音声セット

P_ID	O_ID	再生される音声
0	1	赤色のセーターです。長袖です
0	2	黒色のTシャツです。半袖です
0	3	流水レバーは右手後方です。
0	4	冷蔵庫です、上の段は冷凍庫です
0	5	白い靴です
0	6	歯磨き粉です
0	7	洗顔フォームです
…	…	
0	255	自宅の鍵です

表 2. 自宅周辺の音声セット

P_ID	O_ID	再生される音声
1	1	松永さんの家です
1	2	○○胃腸科へは直進100mです
1	3	花屋です
1	4	公衆電話です
1	5	信号機のボタンです
1	6	AED装置です
1	7	駐車場です
…	…	
1	255	自宅です

表 3. 公民館の音声セット

P_ID	O_ID	再生される音声
2	1	公民館玄関です
2	2	トイレ男性は右、女性は左です
2	3	前方にトイレ操作パネルがあります
2	4	多目的ホールです
2	5	右は○○室、左は××室です
2	6	左の蛇口は熱湯が出ます
2	7	洗顔フォームです
…	…	
2	255	階段は14段です

表 4. 最寄駅の音声セット

P_ID	O_ID	再生される音声
3	1	改札です
3	2	手前から1，2，3，4番線です
3	3	1，2番線ホームは中央線です
3	4	3，4番線ホームは山の手線です
3	5	3番線は内回り山の手線です
3	6	新宿方面です
3	7	終電は12時43分です
…	…	
3	255	ベンチは5メートル先にあります

表 1 から表 4 に示したように、音声情報は晴眼者ではあたりまえに得ることが出来る情報であるが、視覚障害者などにとっては得ることが難しい情報を格納する。特に駅などの案内は音声で行われているが、どこから「３番線に電車が参ります」と音が鳴っているかは判別が難しい。この RFID に対応付けさせることにより、RFID が存在している側のホームが音声と対応している電車であることが視覚障害者にも容易に分かるようになる。

4. 実装

今回の実装では携帯電話を想定しているため、J2ME での実装となる。シミュレーションは Eclipse へプラグインをインストールして行う事が可能である。MIDP、Doja ともに Eclipse 上でシミュレーションを行った。J2ME で実装を行うにあたり、Yu Feng, Dr. Jun Zhu らによる著書”J2ME ワイヤレス Java プログラミング“[15]を参考にした。

4.1 MIDP

MIDP は docomo 以外の携帯キャリア用の java の API である[16]。バージョンは 2.0 を用いた。J2ME の構造内ではプロファイルにあたる。MIDP においては通信方法はソケット通信を用いる。なお MIDP では利用できるファイル形式は MIDI、WAVE 形式と Tone Sequence である。使用するファイルは java ファイルが存在するディレクトリと同階層の res フォルダに格納する必要がある[17]。保存領域としてレコードストアと呼ばれる領域がある。これは半永久的にデータが保存できる。レコードストアの領域はアプリケーションごとに用意されており、他のアプリケーションのレコードストアを参照することも可能である。また、レコードストアは複数の「レコード」で構成されており、各レコードは「レコード ID」を持ち、これによりレコードの識別を行う。

レコードストアを操作するには「javax.microedition.rms.RecordStore」クラスを使い、レコードストアの接続、レコードの書き込み・読み込み、レコードストアの切断、の順に処理を行う。

4.1.1　音声の再生

Oblect クラスの getClass()によって Class オブジェクトを生成後、Class クラスのメソッド GetResourceAsStream()でファイルの内容を含んだ InputStream オブジェクトを作成する。音声の再生には Manager クラスの createPlayer メソッドを用いてファイルを Player オブジェクトに変換する事が必要である。Player の状態を prefetch()状態から start()状態に遷移させる事により音声の再生がスタートする。

図 14 に player クラスの状態遷移の様子を示した。メディアリソースを取得した playrer クラスは REALIZED 状態になるため、createrPlayer メソッドを用いてファイルを player クラスに変換した時点では REALIZED 状態に遷移している事になる。なお、prefetch()メソッドを経ないで start()メソッドを使用することはできない。start()が呼ばれた後はクラスの状態は CLOSED となり、一度 CLOSED 状態になるとその状態から遷移することはない。

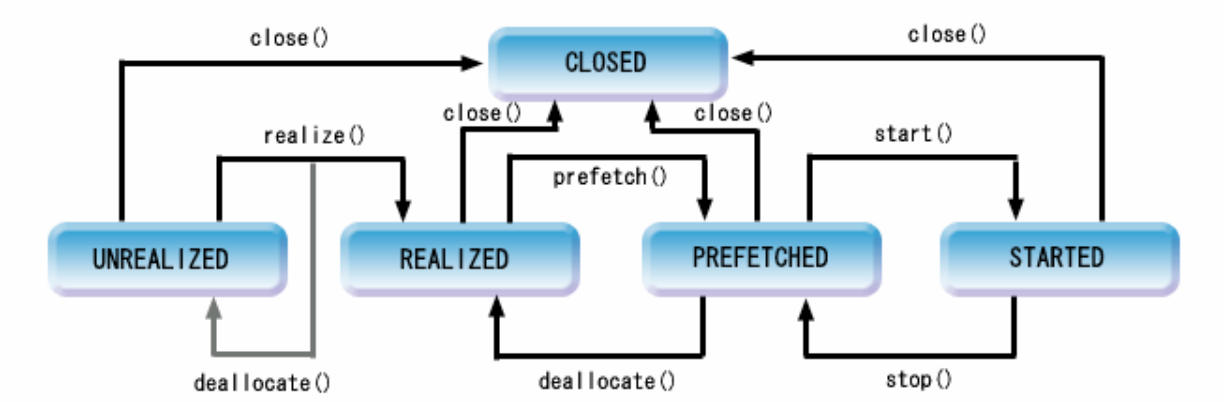


図 14. player クラスの状態遷移図

4.1.2　リーダ用サーバ

クライアントからの要求を受けて PC につながれたリーダにアクセスするために PC 上で実行されている。クライアントアプリケーションとのやりとりはソケット通信を用いる。ServerSocket クラスが実装されている。なお、シミュレーションの時点で必要とされる機能であるため、サーバのみ J2SE での実装となっている。ServerSocket クラスは J2ME では実装されていないクラスである。

サーバが起動されるとポートを開き、accept()メソッドにより接続要求を待機する。接続要求を受信すると、次にクライアントからのリード命令を待機する状態に遷移する。クライアントからリード命令を受信するとリーダに接続した後、取得したタグの ID を出力用 DataOutputStream によって送信する。

4.1.3　リーダ用クライアント

PC につながれたリーダにアクセスするために、PC 上で実行されているサーバに要求を投げる。クライアントは J2ME での実装である。通信はソケット通信を用いる。双方向通信が可能な StreamConnection、受信用に InputStream、送信用に OutputStream を用いる。OutputStream にはリード要求の送信のみに使用される。InputStream にはリーダから読み込んだ int が入るため、これを dataInputStream クラスの readInt メソッドで取得する。取得するのは音声セットに対応する P_ID、各音声セット中に唯一のオブジェクトに対応する O_ID の二つの int データである。

リーダ用クライアントは J2ME であるためシミュレーションは WTK を用いて行う。

4.1.4　ネットワークアクセス

得られた ID を元に、HTTP 通信によってネットワーク上にある音声ファイルにアクセスする。音声ファイルはネットワーク上に P_ID をもとにフォルダ毎に管理されており、場所は”P_ID/O_ID.wav”に位置する。音声は 4.1.1 で述べたように player クラスに変換された後再生される。使用するメソッド、createPlayer(InputStream stream, String type)の type 部分の指定はメディアデータの型を特定する。この type の事を Content-Type と呼ぶ。

いくつかの一般的な Content-Type を下に示す。

- WAV オーディオ・ファイル : audio/x-wav
- AU オーディオ・ファイル : audio/basic
- MP3 オーディオ・ファイル : audio/mpeg
- 標準 MIDI ファイル : audio/midi
- トーン・シーケンス : audio/x-tone-seq

本システムにおいては WAV オーディオ・ファイルを使用するため String は"audio/x-wav"となる。

4.2 Doja

本章では Doja での実装方法を掲載する。Doja は docomo 用の java API である[18]。バージョンは Doja 5.0 である。なお、Doja においてはシリアル通信はサポートされていないため代わりに HTTP 通信を用いる。Doja が動作する携帯電話上においてはファイルという概念を持たないため、データの永続的な保存には特別な領域が確保される。この領域を「スクラッチパッド」と呼ぶ。この保存領域はバイト配列で構成されており、Generic Connection

で得た InputStream、OutputStream を使って読み込み／書き込みが可能である。スクラッチパッドで使用できる領域サイズは、JAM ファイルの SPsize で指定されたバイト数によって設定され、実機では JAM ファイルを参照して、各アプリケーションにスクラッチパッド用の領域を確保する。スクラッチパッドの容量はアプリケーションサイズとあわせて 1024KB までの範囲で設定することができる[18]。そのサイズを超えるとエラーとなり実行およびダウンロードに失敗する。

HTTP 通信においても制約が厳しく、一度の接続では 150KB までしか取得できない。そのため大きなファイルなどは分割してダウンロードする必要がある。

MIDP との大きな違いとして、jarinflater クラスの存在があげられる。なぜなら MIDP には jarinflater クラスに相当するクラスがない。このクラスは jar ファイルを扱うクラスであり、圧縮された複数のファイルを扱う際に利便性が高い。詳しい取り扱い方法については 4.2.4 にて述べる。

HTTP 通信をする際には jam ファイルという設定ファイルに記述が必要となり、変数 UseNetwork に文字列 http を指定する必要がある。また同じ jam ファイルに PackageURL や AppSize という変数が存在し、これらは通信を行なうネットワークの指定やアプリケーションのサイズの記述部分である。Eclipse では設定画面でこれらの操作を行うことが可能である。図 15 に設定画面を示す。ADF 設定において UseNetwork 欄にチェックをし、AppSize 欄にはアプリケーションのバイト数を指定する。他の項目も必要に応じて変更が可能である。この設定画面により jam ファイルを操作することができる。

図 15. eclipse におけるスクラッチパッド、HTTP 通信の設定画面

4.2.1 音声の再生

音声の再生には MediaManager クラス、MediaSound クラスとと AudioPresenter クラスが必要になる。最初に MediaManager クラスの getSound()メソッドの引数にリソースイメージを入れ、MediaSound クラスを取得する。引数のロケーション指定については、最低限のチェ

ック形式のチェック、アクセス可能な URL かどうかのチェックなどが行われる。リソースオブジェクトは指定されたロケーションを保持する。この MediaSound はインターフェースであり、メディアリソースを音声として扱う場合に使用する。次に MediaSound クラスの状態を use()メソッドにより使用可能な状態にする。最後に MediaSound クラスを引数にして AudioPresenter に setSound()メソッドを使用した後、play()メソッドによって再生させる。AudioPresenter は音声メディアデータの再生オブジェクトを定義する。 AudioPresenter クラスは、音を出すメディアデータを再生するためのプレゼンタである。リスナーを登録することによって、演奏の状態の通知を受けることができる。 端末で再生できないようなメディアデータをセットした場合の振舞いは機種依存で、 UIException が発生する場合があある。 属性の設定は、setAttribute メソッドを使用する。 setData メソッドと setSound() メソッドは、1 つのプレゼンタオブジェクトに対して排他的に使用する。setData メソッドでデータを設定してから setSound メソッドでサウンドデータを設定したり、その逆をしたりすることはできない。同じメソッドを複数回呼出した場合は、最後に設定したデータが有効になる。

4.2.2 リーダ用 PHP サーバ

ソケット通信が Doja ではサポートされていないため、PHP で接続をはかる。PHP サーバがリーダ用のサーバアプリケーションを立ち上げる。なお PHP 用パッケージは XAMPP を使用した。XAMPP とは、ウェブアプリケーションの実行に必要なフリーソフトウェアをパッケージとしてまとめたものである。主として開発用ではあるが、実運用環境として使われることもある。Apache(Web サーバ)、MySQL(SQL データベースサーバー)と Web プログラミング言語である PHP や同目的で使われる Perl の 4 つの主要ソフトウェアと管理ツールなど、いくつかの補助的なソフトウェアが含まれている。名前の 2 文字目以降の AMPP は主要 4 ソフトウェアの頭文字である。元々は対応 OS は Linux のみであり、その頭文字 L を付け LAMPP と称したが、後に複数の OS に対応したため L を X に変え XAMPP となった。現在、Windows、Linux、Mac OS X、Solaris で利用可能である。本来、前述の複数のソフトウェアを個別にインストールする必要があり、非常に手間がかかるが、XAMPP は一括してインストールするだけですぐに開発や運用が開始できるため、広く普及している。

この PHP サーバはクライアントが web ページ(サーバ)にアクセス要求をしてきた際に、PC 内でリーダ用サーバを立ち上げ、タグの ID を取得する。取得した ID を動的に web ページとして表示したものをクライアントが読み込み、アプリケーション内で使用する。

図 16 にクライアントアプリケーションからではなく、ブラウザからサーバにアクセスした際の挙動を示す。これは PHP サーバがリーダ用サーバを実行し、タグの ID２つを取得している。0 が P_ID であり、1 が O_ID である。二つの int は改行コードで区切られているが、

ブラウザで表示されているため改行されない。(ブラウザでの改行は
によって表すためである)

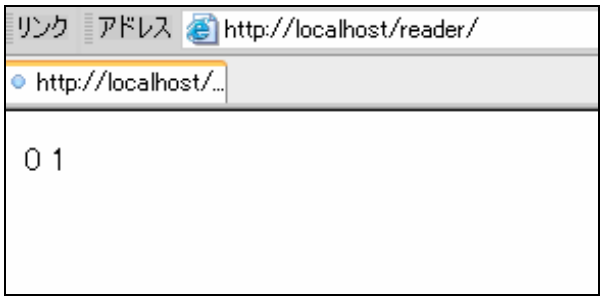


図 16. サーバを単独で動かした際の挙動

4.2.3 リーダ用クライアント

PC につながれたリーダにアクセスするために localhost 上に存在する PHP サーバへの要求をなげる。Connector クラスの open()メソッドを使用する際の引数はパラメータ:
name - 接続先 URL
mode - アクセスモード
timeouts - 呼び出し側がタイムアウト例外を要求していることを示すフラグ
が必要となるがこれらには"http://localhost/reader/index.php", Connector.READ, true を指定する。PHP に対しての通信は HTTP プロトコルを用いる。このプログラムの挙動を図 17 に表す。

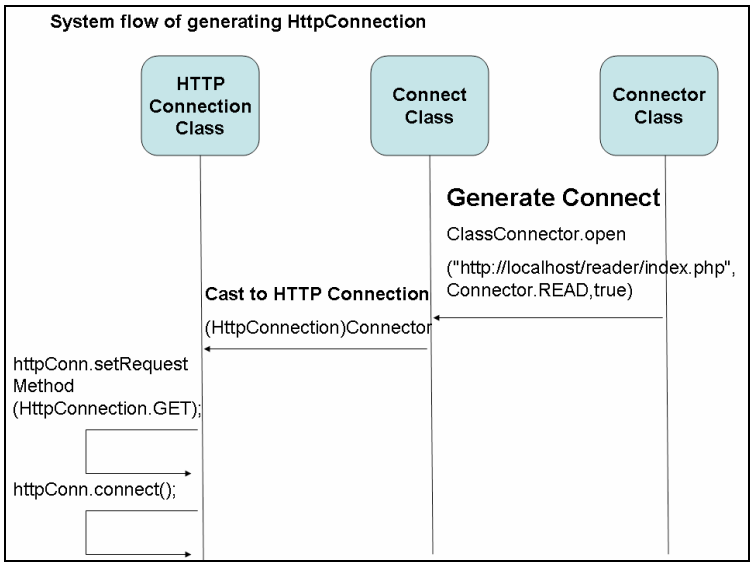


図 17. HTTPConnecion 生成のシステムフロー

open()メソッドを使用して生成された Connection クラスを HttpConnection にキャストした後、InputStream によってデータを取得する。図 16 に示したようなデータを BufferdReader クラスによって扱う。扱う int 型データ２つは改行文字で区切られているため、readLine()メソッドによって１つずつに区切ることが可能である。図 17 で HTTPConnection を生成した後のプログラムの挙動を図 18 で表す。

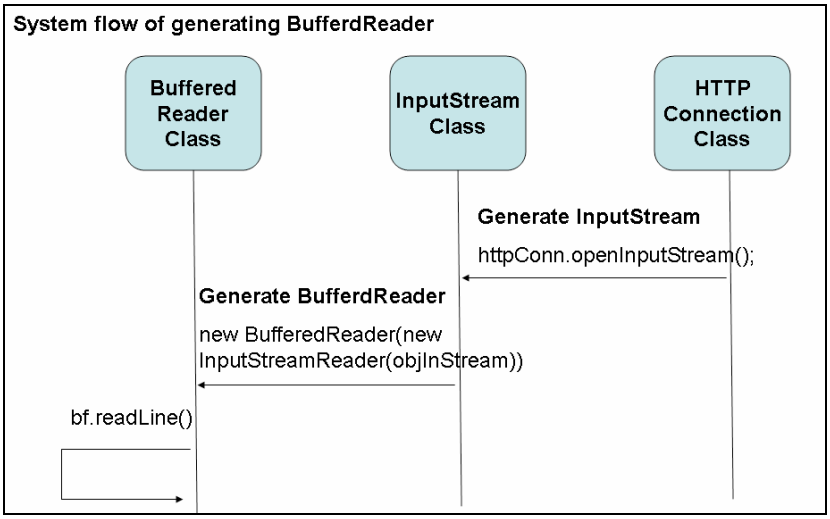


図 18. BufferedReader 生成のシステムフロー

4.2.4 jarinflater

jar ファイルを扱う上で必須となるクラスである。API によると"Jar 形式のファイルイメージからそのエントリを伸長して取り出すためのクラス"であり、jarInflater クラス自体を jar ファイルのイメージとして扱う事ができる。

Jarinflater オブジェクトは Jar 形式のファイルを InputStream に変換した後、またはバイト配列に変換した後、コンストラクタを呼び出す事により生成できる。ファイルの内容を取得するには、getInputStream メソッドの引数にファイル名を指定する。

なお、InputStream を指定してコンストラクタを呼び出す場合、そのストリームからは 1 つの JAR ファイルが過不足なく読み出せる必要がある。 一部が欠落していたり、また末尾などに余分なデータが付加されて読み出されたりする場合はフォーマット違反と見なされ例外が発生することがある。特に ScratchPad に結び付けられたストリームをコンストラクタ に指定する場合、その ScratchPad をオープンする際には引数に pos オプションと length オプションを必ず指定し、適切なデータが読み出せるようにする必要がある。なぜなら jarinflater はどこまでが jar ファイルかを判別する能力がないためである。length オプションを指定することにより、jar ファイルの区切りをクラスに対して示す。

5. 実験と結果

5.1. J2ME におけるプロファイルの違い

MIDP と Doja の違いをここで比較しておく。
MIDP：RMS 機能はあるが、圧縮ファイルを扱うクラスが存在しない
Doja ：スクラッチパッドが存在し、jarinflater と呼ばれる圧縮ファイルを扱うクラスが実装されている。
そのため MIDP では直接ネットワークに毎回接続し、１つのファイルを再生させる形をとった。一方 Doja では複数ファイルを jar ファイルに圧縮し、1 回の通信で取得できるため、毎回接続する方法と、必要に応じて接続する方法の２つの形を実装した。
本章ではそれぞれの音声の再生時間について測定した。

5.2. ストア法の違い

MIDP においては、オブジェクトが存在する範囲を区切り、各場所に P_ID を振り分けていた。この P_ID をネットワーク上のフォルダに見立てる。そして各場所に O_ID が 0 ～255 まで存在するため、この O_ID をフォルダ中のファイルに例える。MIDP のアプリケーションは P_ID の番号のフォルダにアクセスし、そのフォルダ内から対応する O_ID を再生する形になる。このストア法を以下に表す。

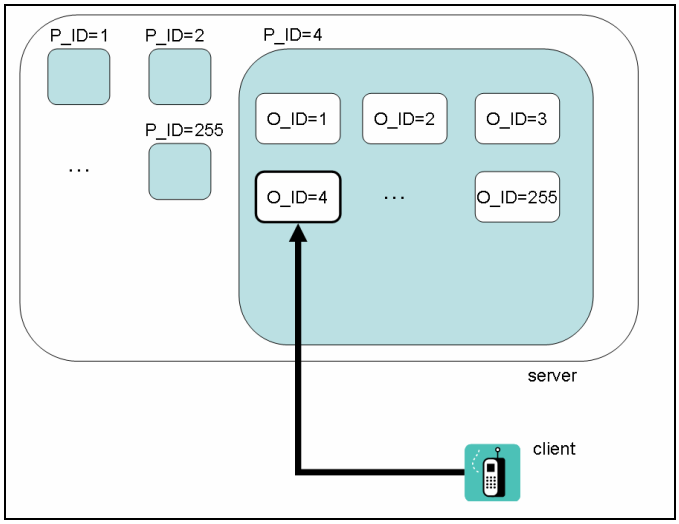


図 19. MIDP におけるサウンドファイルアクセス方法

次に、Doja でのストア法は各 P_ID に対応する jar ファイルがネットワーク上に存在し、アプリケーションは P_ID に対応する jar ファイルをダウンロードし、スクラッチパッドに

保存する。その後スクラッチパッドにある jar ファイル内から O_ID に対応するファイルを取得し、再生する。なお、P_ID がアプリケーション内に保存していた ID と一緒の場合、（すなわちユーザが区切られている場所から他の ID の場所へ移動しなかった場合）ネットワーク接続は行われず、スクラッチパッドへのアクセスのみでサウンドファイルの再生が行われる。

なお、各言語での処理能力の比較のため、Doja においても MIDP で実装されたストア法(図19)を用いて実験を行った。

5.3. 実験結果

最初に MIDP においてネットワークに毎回接続した際のタグ読み込みから再生までの所要時間を測定した。タグを読み込むためにソフトキーを押下した瞬間の時刻を Date クラスのデフォルトコンストラクタで取得し、その後音声が再生された瞬間の時刻も Date クラスによって取得する。単位は ms である。

ネットワークにアップしてある各フォルダのファイル数は 1、50、100、200、255 として、それぞれ差異が生じるかを比較した。再生するファイルは 001.wav、050.wav、099.wav、255.wav とし、ファイル名によっても時間に差異があるかを測定する。以下の表にファイルを１５回再生させ、平均をとった実験の結果を示す。

表 5. MIDP ファイル数 1 の場合の再生時間

再生ファイル＼再生回数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ave
001.wav	235	188	203	203	203	203	187	203	219	204	203	188	203	219	203	204
															単位	ms

表 6. MIDP ファイル数 10 の場合の再生時間

再生ファイル＼再生回数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ave
001.wav	204	219	188	203	250	203	188	187	188	203	188	203	203	203	234	204
															単位	ms

表 7. MIDP ファイル数 50 の場合の再生時間

再生ファイル＼再生回数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ave
001.wav	187	235	203	187	219	203	235	218	203	203	203	203	203	188	203	206
050.wav	203	188	234	250	203	203	203	203	203	203	219	188	203	203	187	206
															単位	ms

表 8. MIDP ファイル数 100 の場合の再生時間

再生ファイル＼再生回数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ave
001.wav	203	203	218	203	203	219	203	234	235	203	218	203	219	204	204	211
050.wav	219	203	203	203	203	219	203	204	203	203	203	188	188	219	219	205
099.wav	219	203	203	234	235	187	265	203	234	219	250	219	234	234	219	224
															単位	ms

表 9. MIDP ファイル数 200 の場合の再生時間

再生ファイル＼再生回数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ave
001.wav	203	218	203	203	203	187	187	219	203	234	203	219	234	188	250	210
050.wav	188	218	219	235	234	172	250	219	234	219	219	266	203	203	203	219
099.wav	203	203	219	203	187	219	188	219	219	203	203	188	219	234	234	209
															単位	ms

表 10. MIDP ファイル数 255 の場合の再生時間

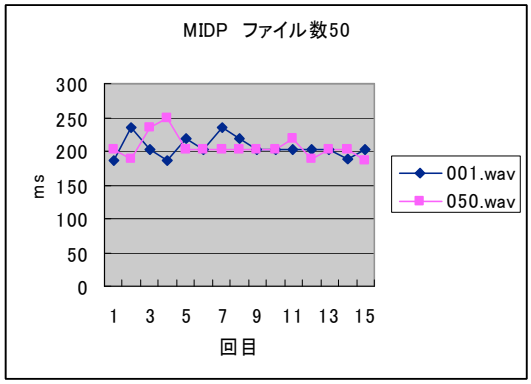
再生ファイル＼再生回数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ave
001.wav	219	219	219	218	203	187	235	203	219	219	219	250	218	203	235	218
050.wav	203	203	188	219	203	203	203	203	219	219	219	203	219	203	203	207
099.wav	188	219	203	203	219	203	203	203	203	219	250	203	219	203	203	209
255.wav	203	203	203	188	219	204	203	188	234	235	203	203	250	187	234	210
															単位	ms

表 11. ファイル数に対する再生所要時間

ファイル数	再生所要時間
1	204
10	204
50	207
100	214
200	214
255	211
	単位 ms

表 5～10 の結果を元に、ファイル数 1～255 におけるサウンドファイルの平均再生時間を計算した結果、ファイル数 1 で 204ms、ファイル数 10 で 204ms、ファイル数 50 で 207ms、ファイル数 100 で 214ms、ファイル数 200 で 214ms、ファイル数 255 においては 211sm となった。この結果を表 11 に表す。これらの数値からファイル数が増えるにつれて、再生時間が若干増えていることが読みとれる。しかし最大値と最小値の差は 10ms ととても小さい。

なお表 7.～表 10.の結果を図 20～23 においてグラフで表した。



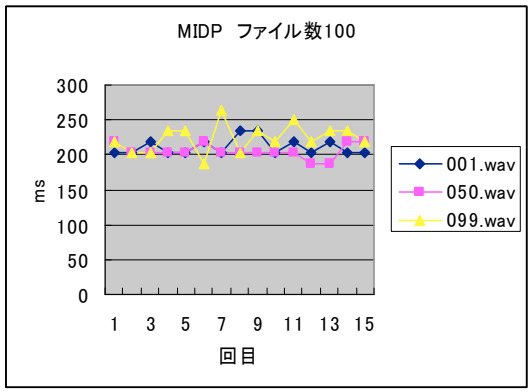


図 20. ファイル数 50 の音声再生所要時間　　**図 21. ファイル数 100 の音声再生所要時間**

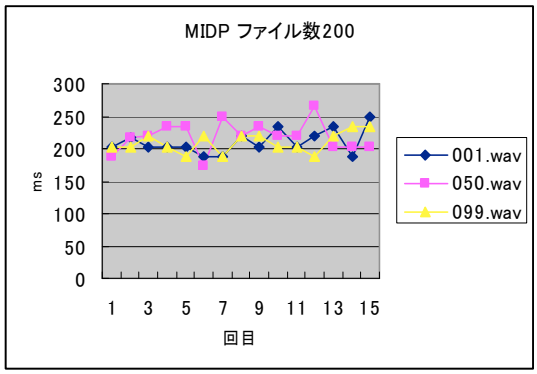


図 22. ファイル数 200 の音声再生所要時間

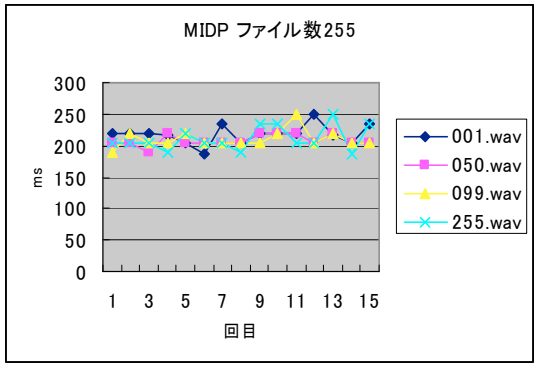


図 23. ファイル数 255 の音声再生所要時間

図 20～23 より、ファイル名 001.wav、050.wav、099.wav、255.wav　により再生所要時間の違いは見られない。平均値は 205ms～220ms 間で推移している事がグラフ、および表から読み取れる。ファイル数が 255 の表 10 に関して言えば、001.wav で平均 218ms、050.wav で平均 207ms、099.wav で 209ms、255.wav で 210ms と、001.wav が最も再生所要時間が遅くなる結果となった。また、これらの時間に法則性は見られない。

次に Doja においてネットワークに毎回接続した際のタグ読み込みから再生までの所要時間を測定した。なお、Doja においては PHP サーバを使いリーダに接続しているため、ボタン押下後から、音声再生までの時間を計るのは実験として適切でない。PHP サーバはシミュレーションの便宜上組み込んであるためである。よって、実験においては実機により近い時間を測定するため、PHP サーバにアクセスし、タグを HTTP 上に表示するまでの時間を 0 として計算した。この時間を 0 にする事により、Doja シミュレータがリーダに直接 HTTP でアクセスしているとした仮定で実験を行うことができる。以下の実験方法は MIDP と同様である。

ネットワークにアップしてある各フォルダのファイル数は 1、50、100、200、255 として、それぞれ差異が生じるかを比較した。再生するファイルは 001.wav、050.wav、099.wav、255.wav とし、ファイル名によっても時間に差異があるかを測定する。以下の表にファイルを１５回再生させ、平均をとった実験の結果を示す。

表 12. Doja ファイル数 1 の場合の再生時間

再生ファイル＼再生回数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ave
001.wav	718	609	609	610	704	688	688	703	687	688	609	610	687	610	609	655
															単位	ms

表 13. Doja ファイル数 10 の場合の再生時間

再生ファイル＼再生回数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ave
001.wav	610	609	610	703	672	703	703	703	703	688	610	609	687	609	609	655
															単位	ms

表 14. Doja ファイル数 50 の場合の再生時間

再生ファイル＼再生回数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ave
001.wav	672	609	609	610	703	703	687	688	688	610	609	609	703	609	610	648
050.wav	610	609	703	687	688	703	703	703	609	609	609	610	704	609	609	651
															単位	ms

表 15. Doja ファイル数 100 の場合の再生時間

再生ファイル＼再生回数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ave
001.wav	656	609	610	609	703	703	703	703	609	609	687	609	609	625	609	644
050.wav	703	609	609	813	688	687	688	734	703	704	703	609	609	813	609	685
099.wav	719	610	609	610	672	672	672	703	609	610	609	609	687	609	609	641
															単位	ms

表 16. Doja ファイル数 200 の場合の再生時間

再生ファイル＼再生回数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ave
001.wav	719	610	609	610	703	703	703	688	704	703	610	609	609	609	609	653
050.wav	719	609	609	610	703	703	688	688	672	671	688	687	609	610	609	658
099.wav	656	610	609	610	703	703	703	703	703	703	703	703	688	609	609	668
															単位	ms

表 17. Doja ファイル数 255 の場合の再生時間

再生ファイル＼再生回数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ave
001.wav	672	609	610	610	703	688	609	609	609	609	703	610	609	657	609	634
050.wav	703	610	609	703	704	610	610	609	610	703	609	765	704	703	687	663
099.wav	703	609	609	609	703	704	688	609	813	610	609	703	687	687	609	663
255.wav	671	609	609	610	703	703	688	688	610	609	687	609	609	734	610	650
															単位	ms

表 18. ファイル数に対する再生所要時間

ファイル数	再生所要時間
1	649
10	654
50	649
100	657
200	660
255	653
	単位 ms

表 12.～表 17.の結果を元に、ファイル数 1～255 におけるサウンドファイルの平均再生時間を計算した結果、ファイル数 1 で 649ms、ファイル数 10 で 654ms、ファイル数 50 で 649ms、ファイル数 100 で 657ms、ファイル数 200 で 660ms、ファイル数 255 においては 653sm となった。この結果を表 18 に表す。これらの数値からファイル数が増えるにつれて、ばらつきはあるが再生時間が若干増えてゆく傾向にある事が読みとれる。しかし最大値と最小値の差はこちらも 11ms ととても小さい。

なお表 12.～表 17.の結果を図 24～27 においてグラフで表した。

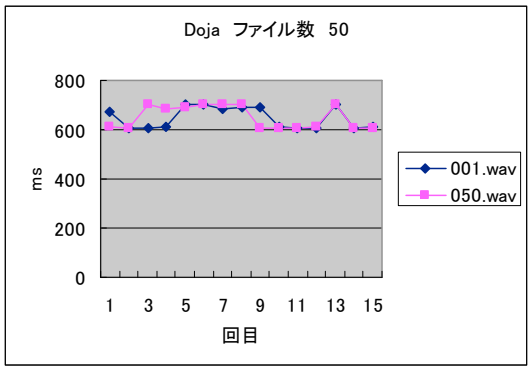


図 24. ファイル数 50 の音声再生所要時間

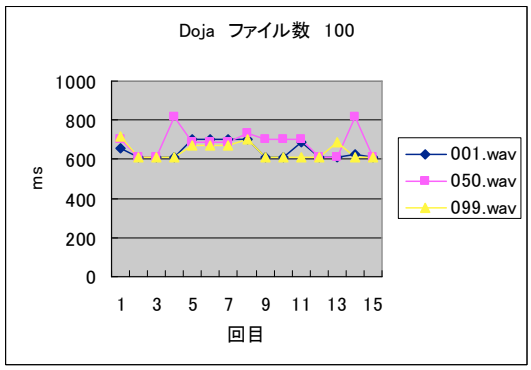


図 25. ファイル数 100 の音声再生所要時間

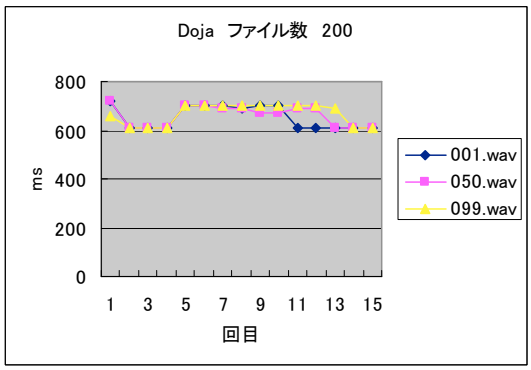


図 26. ファイル数 200 の音声再生所要時間

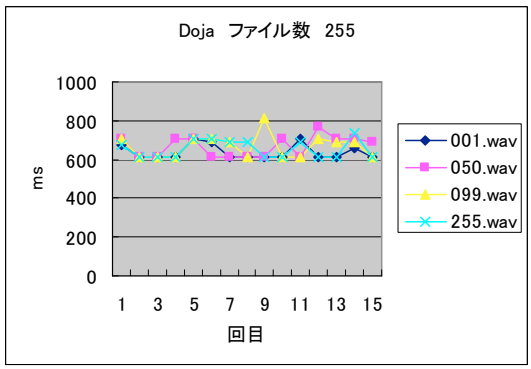


図 27. ファイル数 255 の音声再生所要時間

図 24～27 より、ファイル名 001.wav、050.wav、099.wav、255.wav　により再生所要時間の違いは見られない。平均値は 600ms～700ms 間で推移している事がグラフ、および表から読み取れる。

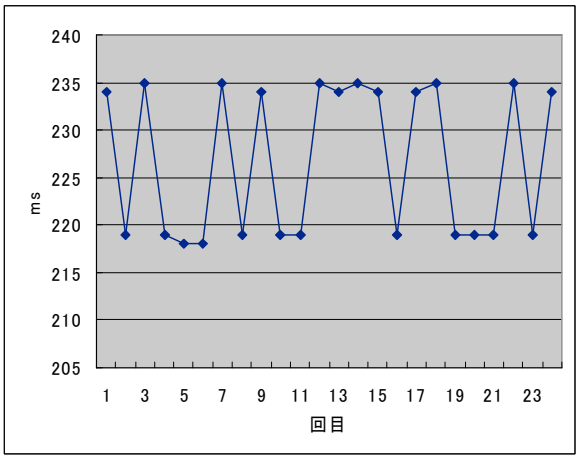


図 28. ローカルデータの音声再生所要時間

最後に音声セットをダウンロードして、ローカルデータにアクセスした際にかかった時間を図 28 に示した。　図 28 に示されたグラフを参照すると、ローカルデータにアクセスする際の音声再生所要時間は 220～235ms であるとされる。（値は中間値ではなく約 220ms または 235ms である）ローカルデータである音声セットをダウンロードするのにかかる時間は約 4000ms である。

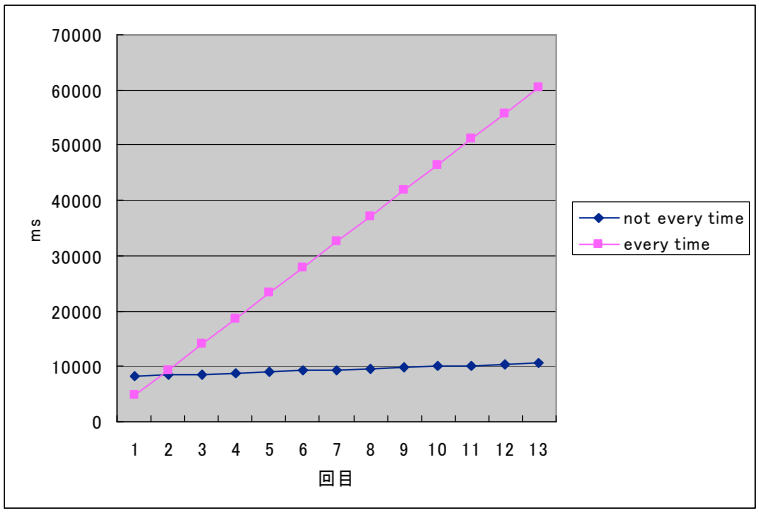


図 29. 音声再生所要時間の比較（接続時間=4000ms）

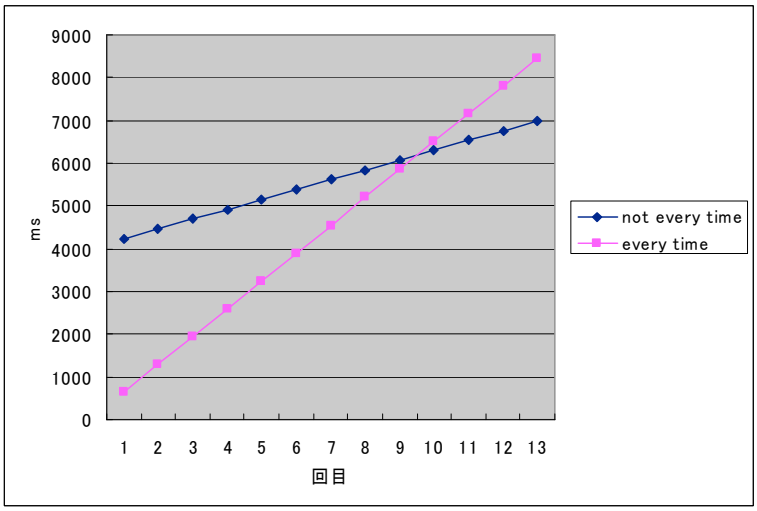


図 30. 音声再生所要時間の比較（接続時間=0ms）

Doja で行った二つの実験を受けて、図 29、図 30 のようなグラフで比較を行った。毎回インターネットに接続して音声を再生した場合の合計した音声再生所要時間と、音声セットを用いて毎回は接続を行わなかった場合の合計した音声再生所要時間の比較である。

図 29 のグラフにおいては、毎回接続時は音声再生所要時間の 650ms に加え、ネットワークに接続するための接続時間 4000ms を含めている。よって原点を通過するピンク色の直線で表されるように、比例的に時間は増える。これを式に表すと式(1)のようになる。

$$time = (c + p)x \quad (1)$$

式(１)において、c は接続時間(この場合は 4000ms)、p は再生にかかる時間(650ms)である。x は再生した回数であり、合計時間は再生した回数に比例する。

しかし、毎回は接続しない場合は、原点を通過しない紺色の直線で表されるように、初回に接続時間 4000ms に加え、ダウンロードに要する時間 4000ms が必要とされるのみで、音声再生が２回目以降になると 230ms ずつ増えてゆくのみである。これを式(2)で表した。

$$time = c + d + px \quad (2)$$

式(2)において、 c は接続時間(この場合は 4000ms)、d は音声セットをダウンロードするのにかかった時間(4000ms)、p は再生にかかる時間(230ms)である。x は再生した回数である。

よって、音声再生が２回目以降になると音声セットを一括ダウンロードする手法の方が所要時間は短くなるという結果になった。

図 30 のグラフにおいては、図 29 のグラフのネットワーク接続時間を０にした場合の音声再生所要時間を表している。式(1)、式(2)における C を０にした式が成り立つ。これは将来携帯電話がネットワークによりスムーズに接続できるようになった場合を仮定している。この場合においても、音声再生が１０回目以降になると音声セットを一括ダウンロードする手法の方が所要時間は短くなるという結果になった。

6. 既存のシステムとの比較

財団法人機械システム振興協会[9]による、ＩＣタグ生活支援システムの音声再生までの時間は５～６秒とされている。しかし本システムでは MIDP、Doja ともにネットワークに毎回接続した際の平均した再生時間までの時間も５秒を超えることはない。平均して 400ms となる。よって音声再生までの時間は 80 パーセント以上短縮されたことになる。

ネットワークに毎回接続しない場合はさらに所要時間が短くなるため、より本システムが有効であることが表すことが出来る。

7. 結論

ネットワークに毎回接続した際はフォルダに格納するファイル数で多少の差異が確認された。表 11 によるとファイル数が大きくなるほど、再生時間が多くなる。この事から単純に推測するとファイルシステムが線形にファイルを探していることを表しているとされる。だが、フォルダ内でいくつかのファイルを読み込んだ際には、ファイルの並び順によって再生時間が左右されることは確認されなかった。この点は線形にファイルを読み込んでい

るとすると矛盾点となる。しかし、ファイル数によって変化する再生時間はごく小さなものであるので誤差の可能性も捨てきれない。

なお、リーダによる ID 読み込みのたびにネットワーク上へアクセスする事はやはり非効率的であった。これの理由としては、ストレージにあるローカルデータを再生する時間の方がネットワークにアクセスして得たファイルを保存した後、データを再生するよりも時間が短いためである。もうひとつはハード面での問題として、ネットワークにアクセスする事自体に時間がかかりすぎるという点があげられる。しかし、このハード面での問題は解決されたと仮定しても、一括で音声セットをダウンロードした方が効率的であることが証明された。

図 30 においては１０回以降であるならば、という制限が生じるが、実際にユーザが１０回以下の音声再生回数で他の場所へ移ってしまうという事がまれなため、効率的なシステムが提案できた事になる。

8. 今後の課題

携帯電話のストレージが今よりも増えることが今後の流れであるため、より多くなったストレージへのアクセス方法などが検討されるとよい。

また使用履歴を反映したストレージアルゴリズムを提案する事により、スムーズになるのではないだろうか。ユーザがよく読みこむタグをローカルに保存しておくことにより、キャッシュシステムに似たストア法を実現することができれば、よりネットワークにアクセスする回数が減ると予想される。最もよく使うファイルほど最も再生時間が短くなれば、全体の再生時間の平均が短くなるためである。

また拡張機能として、赤外線機能を用いた機能をここにあげておく。Bluetooth は有効範囲がひろく、本システムで用いた RFID タグの有効範囲とは大きく異なる。しかし、赤外線機能は範囲が比較的狭く、RFID と有効範囲が似ている。そのため、認識した RFID がついている製品以外の赤外線通信が可能な製品を検出してしまう可能性が Bluetooth に比べて低い。よって赤外線機能のみを拡張機能に用いる技術としてここに挙げる。

拡張機能として音声を再生するフェーズの次に、製品と赤外線通信を使って通信をする機能を図 31 に示した。図 31 において①～③までは本システムで実装されている部分である。赤外線通信で行うのは④の部分である。これには object（製品）側が赤外線通信に対応していること、赤外線通信によっておくられてきたコマンドを実行するための機能がついていることが必須条件となる。携帯電話からは赤外線通信によってコマンドを送信するのみである。ユースケースとして、CD プレイヤーを例にとる。音声によって「携帯電話の１ボタンで再生、２ボタンで一時停止、３ボタンで停止です」などと案内することによりユーザに携帯電話のどのボタンを押せば、製品がどんな挙動をするかを認識させる。ユーザが押したボタンにより赤外線通信で携帯電話は製品にコマンドを送る。すると製品は送

られたコマンドどおりに制御される。このシステムは携帯電話は多くの家電製品のインターフェースになることを目的としている。携帯電話が多くの家電の操作インターフェースになれば、視覚障害者は新たな家電などを買った際にもストレスなく操作することができる。なお赤外線通信を行う API は com.nttdocomo.device パッケージで配布されている。このパッケージ内の IrRemoteControl、IrRemoteControlFrame クラスを使用することにより赤外線機能を実装する。IrRemoteControl クラスは赤外線のリモートコントローラ機能を提供する。IrRemoteControlFrame クラスは IR リモートコントローラから送信されるデータフレームを定義する。

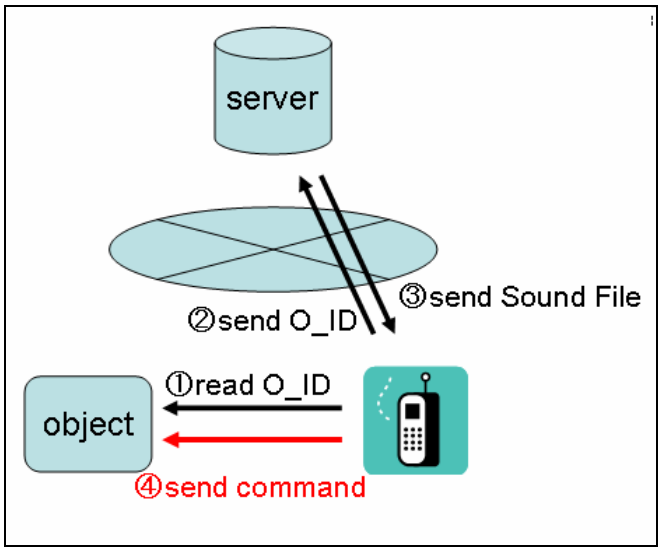


図 31. 赤外線通信を使ったシステムの構成

参考文献

[1] 厚生労働省, 医療
http://www.mhlw.go.jp/bunya/iryou/index.html

[2] 社会福祉法人　日本盲人会連合ホームページ
http://www.normanet.ne.jp/~nichimo/

[3] 石川 准, 兵藤 安昭, *Developing an Accessible GPS System for the Blind,* IEICE technical report, 105(553), pp 51-56, Jan. 2005

[4] 檜垣 宏行, 牧野 秀夫, 渡部 礼二, 鉄本 秀夫, 前田 義, *視覚障害者用音声位置案内システムにおける GPS 携帯電話・PDA の実験と評価,* IEICE technical report. ME and bio cybernetics, 103(327), pp 61-66, Sep. 2003

[5] 檜垣 宏行, 牧野 秀夫, *Speech Guidance System for the Visually Impaired using a GPS Cellular Phone including and Azimuth Sensor,* Proceedings of the IEICE General Conference, 2003 年_基礎・境界, pp.368, Mar. 2003

[6] 自立移動支援プロジェクト, 自律的移動支援プロジェクトのイメージ
http://www.jiritsu-project.jp/material/html/20040318/siryo3.html

[7] 総務省（報道資料）, 「ユビキタスネットワーク時代における電子タグの高度利活用に関する調査研究会」最終報告
http://www.soumu.go.jp/s-news/2004/040330_6.html

[8] JISC 日本工業標準調査会
http://www.jisc.go.jp/

[9] （財）共用品推進機構, 無線 IC チップによる障害者・高齢者支援システム基盤整備に関する調査研究委員会
http://kyoyohin.org/

[10] 野村 富恵, 物知りトーク
http://www002.upp.so-net.ne.jp/nomura/shouhin/shouhin-monoshiri.htm

[11] マイコミジャーナル, ユビキタス ID を利用した歩行支援システムがデモ公開される
http://journal.mycom.co.jp/news/2004/03/24/001.html

[12] 新潟大学, 視覚障害者歩行支援を軸とした蛍光灯通信位置情報提供プラットフォームの開発
http://www.niigata-u.ac.jp/gakugai/st/001/makino.pdf

[13] 東京都立産業技術研究センター（都産技研）, 自律分散手法による視覚障害者移動支援システム技術の開発
http://www.iri-tokyo.jp/research/eval/16before/16b-07.htm

[14] Microsoft, RFID ホーム
http://www.microsoft.com/japan/business/rfid/default.mspx

[15] Yu Feng, Dr. Jun Zhu, *J2ME ワイヤレス Java プログラミング*, May 2002

[16] Sun, MIDP - Download 2.0
http://www.sun.com/software/communitysource/j2me/midp/download20.xml

[17] スパイシーソフト株式会社, アプリ開発講座
http://appget.com/contest/au2007/lecture/index.html

[18] NTT ドコモ, i アプリ ｜ サービス・機能 ｜
http://www.nttdocomo.co.jp/service/imode/make/content/iappli/

付録

Java ソース

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Date;

import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.StringItem;
import javax.microedition.lcdui.TextField;
import javax.microedition.media.Player;
import javax.microedition.midlet.MIDlet;
import javax.microedition.media.*;

public class MIDPMain extends MIDlet implements CommandListener {

        private Form mainForm;

        private TextField inputTextBox, inputTextBox2;

        private StringItem outputLabel;

        private Command endCmd = new Command("終了", Command.EXIT, -1);

        private Command readTag = new Command("read tag", Command.ITEM, 1);

        private Date start, end;

        private String connectString = "socket://192.168.123.21:5000";

        private int count = 0;

        private int total = 0;

        // 双方向通信が可能なStreamConnection
        private StreamConnection streamConnection = null;

        // リクエストの送信用にOutputstramを利用
        private OutputStream outputStream = null;

        private DataOutputStream dataOutputStream = null;

        // リクエストの受信用にOutputstramを利用
        private InputStream inputStream = null;

        private DataInputStream dataInputStream = null;

        private int O_ID;

        private int P_ID;

        private InputStream in;

        Player[] player = new Player[2];

        // コンストラクタで初期化
        public MIDPMain() {

                mainForm = new Form("MIDletSample");
                inputTextBox = new TextField("P_ID", "", 200, TextField.LAYOUT_LEFT);
                inputTextBox2 = new TextField("O_ID", "", 200,

                TextField.LAYOUT_RIGHT);
                outputLabel = new StringItem("", "");
                mainForm.addCommand(endCmd);
```

```
		mainForm.addCommand(readTag);
		mainForm.append(inputTextBox);
		mainForm.append(inputTextBox2);
		mainForm.append(outputLabel);
		mainForm.setCommandListener(this);
		;
	}

	// コンストラクタの次にまず実行されるのがstartapp
	protected void startApp() {
		Display.getDisplay(this).setCurrent(mainForm);

		try {
			// リモートサーバーとのソケット接続を確立
			streamConnection = (StreamConnection) Connector.open(connectString);

			// ソケット接続上にDataOutputStreamを作成
			outputStream = streamConnection.openOutputStream();
			dataOutputStream = new DataOutputStream(outputStream);

			// ソケット接続上にDataInputStreamを作成
			inputStream = streamConnection.openInputStream();
			dataInputStream = new DataInputStream(inputStream);

		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		outputLabel.setText("サーバーにアクセス完了");

	}
	public void hyouji(int P_ID, int O_ID){
		inputTextBox.setString(Integer.toString(P_ID));
		inputTextBox2.setString(Integer.toString(O_ID));

	}

	// ボタンを押されたときの挙動
	public void commandAction(Command c, Displayable d) {
		if (c == endCmd) {
			notifyDestroyed();
		} else if (c == readTag) {// readTagをおしたら実行
			try {
				start = new Date();
				dataOutputStream.writeInt(1);// serverに”1” （←読め）を送信
P_ID
=

				// dataInputStream.readInt();
				P_ID = dataInputStream.readInt();
				O_ID = dataInputStream.readInt();

				// outputLabel.setText("P_ID:" + P_ID + " O_ID:" + O_ID);
//
//
//				inputTextBox.setString(Integer.toString(P_ID));
//				inputTextBox2.setString(Integer.toString(O_ID));
				hyouji(P_ID,O_ID);

			} catch (IOException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}

			try {

				// ////////////////
				// webに音声アクセス//
				// ////////////////

				 Player p;

				 if (O_ID == 1) {

				 p = Manager
```

```
.createPlayer("http://rhlab.cis.k.hosei.ac.jp/moriya/sound/000/001.wav");
                                }
                                else if (O_ID == 2) {

                                p = Manager

.createPlayer("http://rhlab.cis.k.hosei.ac.jp/moriya/sound/000/050.wav");
                                }
                                else if (O_ID == 3) {

                                        p = Manager

.createPlayer("http://rhlab.cis.k.hosei.ac.jp/moriya/sound/000/099.wav");
                                }
                                else if (O_ID == 4) {

                                        p = Manager

.createPlayer("http://rhlab.cis.k.hosei.ac.jp/moriya/sound/000/255.wav");
                                } else{
                                System.out.println("hoge");
                                p = Manager.createPlayer("hoge");


                                }
                                p.start();
                                end = new Date();
                                count++;
                                long time = end.getTime() - start.getTime();
                                total = (int) (total + time);
//                              System.out.println("P_ID : "+P_ID);
//                              System.out.println("O_ID: "+O_ID);
//                              System.out.println("time : "+time);
//                              System.out.println("count: "+count);
//                              System.out.println("ave : "+total/count);
                                System.out.println(time+"¥t");

                               // ////////////////////
                               // ローカルで音声アクセス//
                               // ////////////////////
                               // 音声再生の方法１ 例外が発生
                               // Player p;
                               // try {
                               // System.out.println("resource:/001.wav");
                               // System.out.println("try7");
                               //
                               // p = Manager.createPlayer("resource://res/001.wav");
                               // p.start();
                               //
                               // } catch (Exception e1) {
                               // // TODO Auto-generated catch block
                               // // e1.printStackTrace();
                               // System.out.println(e1.toString());
                               //
                               // }
                               // 音声再生の方法２ この方法で再生する
//
//                             if (O_ID == 1) {
//                                     in = getClass().getResourceAsStream("/001.wav");
//                             }
//                             if (O_ID == 2) {
//                                     in = getClass().getResourceAsStream("/002.wav");
//                             }
//
//                             player[1] = Manager.createPlayer(in, "audio/x-wav");
//                             player[1].prefetch();
//                             player[1].setMediaTime(0);
//                             player[1].start();
//
//                             System.out.println("0");
//                             end = new Date();
                               count++;
```

```
//                              //再生にかかった時間＝再生した時間-タグ読み時間
//                              long time = end.getTime() - start.getTime();
//                              //再生トータル回数
//                              total = (int) (total + time);
//                               System.out.println("P_ID : "+P_ID);
//                               System.out.println("O_ID: "+O_ID);
//                              System.out.println("time : " + time);   //再生にかかった時間を表
示
//                              System.out.println("count: " + count);  //トータル回数を表示
//                              System.out.println("ave : " + total / count);   //再生平均時間を
表示
//
//                              while (player[1].getState() == Player.STARTED) {
//                              }
//                              in.close();
                                if(count == 17){
                                        System.out.println("end");
                                }


                        } catch (Exception e) {
                                System.out.println(e.toString());
                        }

                }

        }

        protected void pauseApp() {
        }

        protected void destroyApp(boolean unconditional) {
        }
}
```

```
import java.awt.*;
import java.net.*;
import java.util.Enumeration;
import java.util.TooManyListenersException;
import java.io.*;

import javax.comm.CommPortIdentifier;
import javax.comm.PortInUseException;
import javax.comm.SerialPort;
import javax.comm.UnsupportedCommOperationException;

import com.sun.corba.se.pept.transport.Connection;

import rhlab.io.ExtraDataOutputStream;
import rhlab.rfid.tagit.TagITPacketData;
import rhlab.rfid.tagit.TagITRfidReader;
import rhlab.util.BitConverter;

public class MIDP_Server extends Frame {
	private ServerSocket server;
	private Socket s;

	private DataInputStream input;// 携帯からの命令ストリーム
	private DataOutputStream output;// 携帯へタグNoを返す出力ストリーム

	public boolean done;

	TextArea textarea = new TextArea("", 15, 30);

	TagITRfidReader reader = null;

	public static void main(String[] args) throws Exception {
		MIDP_Server serv = new MIDP_Server();
	}

	private void initReader() throws IOException, TooManyListenersException,
			UnsupportedCommOperationException, PortInUseException {
		// test引用

		ExtraDataOutputStream out;
		Enumeration portList = CommPortIdentifier.getPortIdentifiers();
		while (portList.hasMoreElements()) {
			CommPortIdentifier portId = (CommPortIdentifier) portList
					.nextElement();
			if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
				if (portId.getName().equals("COM1")) {
					reader = new TagITRfidReader((SerialPort) portId.open(
							"RfidApp", 2000));
					break;
				}
			}
		}

	}

	public MIDP_Server() throws IOException, TooManyListenersException,
			UnsupportedCommOperationException, PortInUseException {

		super("Server");
		server = new ServerSocket(5000, 1);

		setSize(270, 350);
		setLayout(new FlowLayout());
		add(textarea);
		initReader();
		TagITPacketData packet;
		setVisible(true);
		done = false;
		s = server.accept();
		try {
			input = new DataInputStream(s.getInputStream());//入力用
			output = new DataOutputStream(s.getOutputStream());//出力用
```

```
                        ;//サーバーの待機開始
                        //接続すると次へすすむ↓
                        textarea.append("接続完了"); //stramが１であることをダイアログに表示

                        while (!done) {// フラグがdoneでは無い間
                         if(input.readInt()==1){//もし受信した命令ストリームが１（読め）だったら
                                textarea.append("１でした¥n"); //stramが１であることをダイアログ
に表示

                                // 読む
                                packet = reader.readSingleBlock(0, null, 7);
                                textarea.append("tag:"
                                                + BitConverter.toUnsignedInt(packet.getData(),
3, 1));//3ブロック目の長さ１ビットのデータを読んでいる
                                output.writeInt(BitConverter.toUnsignedInt(packet.getData(), 3,
1));
                                output.writeInt(BitConverter.toUnsignedInt(packet.getData(), 2,
1));
                                textarea.append("読んだ値を送信しました¥n");


                                 }

                        }
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        }

        public boolean handleEvent(Event ev) {
                if (ev.id == Event.WINDOW_DESTROY)
                        System.exit(0);
                return false;
        }

}
```

```
import com.nttdocomo.ui.*;

/**
 * Sampleクラス
 */
public class SoundTest extends IApplication {

	/**
	 * 一番最初に呼び出される
	 */
	public void start() {
		mainCanvas canvas = new mainCanvas();
		Display.setCurrent(canvas);
	}
}

/**
 * mainCanvasクラス
 */
class mainCanvas extends Canvas {

	/** サウンドプレイヤー */
	AudioPresenter ap;

	/** 音の用意 */
	MediaSound sound;

	/** サウンド再生中はtrueのフラグ */
	boolean sound_flag;

	/**
	 * コンストラクタ
	 */
	public mainCanvas() {

		// プレイヤーの取得
		ap = AudioPresenter.getAudioPresenter();

		// 音データの取得
		sound = MediaManager.getSound("resource:///sample4.mld");
		// System.out.println("web?");
		// sound =
		//
MediaManager.getSound("http://rhlab.cis.k.hosei.ac.jp/moriya/sound/000/sample2.mld");

		// 音を使える状態にする
		try {
			sound.use();
		} catch (Exception e) {
			// エラーの場合
			System.out.println("Error:" + e);
		}

		// 音データをセット
		ap.setSound(sound);

		// 音の再生
		ap.play();
	}

	/**
	 * paintメソッド
	 */
	public void paint(Graphics g) {
		// 青色をセット
		g.setColor(Graphics.getColorOfName(Graphics.BLUE));
		// 文字列の描画
		g.drawString("音の再生", 0, 30);
	}

}
```

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.util.Date;

import javax.microedition.io.Connector;
import javax.microedition.io.ContentConnection;
import javax.microedition.io.StreamConnection;

import com.nttdocomo.system.StoreException;
import com.nttdocomo.ui.AudioPresenter;
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Image;
import com.nttdocomo.ui.MediaManager;
import com.nttdocomo.ui.MediaSound;
import com.nttdocomo.ui.Panel;
import com.nttdocomo.ui.Frame;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.util.JarInflater;
import com.nttdocomo.util.Phone;
import com.nttdocomo.opt.ui.*;
import com.nttdocomo.ui.*;
import com.nttdocomo.io.*;

public class DojaMain extends IApplication {
        DojaCanvas c = new DojaCanvas();

        public void start() {
                Display.setCurrent(c);
                c.exe();
        }
}

class DojaCanvas extends Canvas {
        HttpConnection httpConn; // HTTP接続オブジェクト

        InputStream objInStream; // 入力オブジェクト

        OutputStream objOutStream; // 出力オブジェクト

        final int ERROR_CODE = -1;

        int O_ID = ERROR_CODE;

        int P_ID = ERROR_CODE;

        private Date start, end,pre,post;


        int storedO_ID = ERROR_CODE;

        int storedP_ID = ERROR_CODE;

        int length,filelength;

        /** サウンドプレイヤー */
        AudioPresenter ap;

        /** 音の用意 */
        MediaSound sound;

        void exe() {
                setSoftLabel(SOFT_KEY_1, "終了");
                setSoftLabel(SOFT_KEY_2, "Read");
        }

        void draw(Graphics g) {
                g.drawString("サーバーに接続完了", 50, 50);
```

```
	}

	public void processEvent(int type, int param) {
		if (type == Display.KEY_PRESSED_EVENT) {
			// 終了
			if (param == Display.KEY_SOFT1) {
				System.exit(0);
			}
			// タグを読む
			else if (param == Display.KEY_SOFT2) {
				start= new Date();
				this.connect();

			}
		}
	}

	public void paint(Graphics arg0) {
		// TODO Auto-generated method stub

	}

	void connect() {

		try {

			// HTTP 接続 (HttpConnection) オブジェクトを取得する。
			// System.out.println("local");
			pre = new Date();

			httpConn = (HttpConnection) Connector.open(
					"http://localhost/reader/index.php", Connector.READ,
true);

			// リクエストメソッドを設定する。
			httpConn.setRequestMethod(HttpConnection.GET);

			// リモート資源に接続する。
			httpConn.connect();
			draw(this.getGraphics());
			post = new Date();

			// データをInputStreamに
			objInStream = httpConn.openInputStream();
			BufferedReader bf = new BufferedReader(new InputStreamReader(
					objInStream));
			P_ID = Integer.parseInt(bf.readLine());
			O_ID = Integer.parseInt(bf.readLine());
//			System.out.println("P_ID : " + P_ID);
//			System.out.println("O_ID : " + O_ID);

			// 入力ストリームをクローズする。
			objInStream.close();
			// HTTP 接続をクローズする。
			httpConn.close();

			if (P_ID != storedP_ID) {

				connectjarfile();
				playjarfile(O_ID);
				storedP_ID = P_ID;
			} else {
				playjarfile(O_ID);

			}

		} catch (IOException e) {
			// TODO Auto-generated catch block
			System.out.println(e.toString());
			// e.printStackTrace();
		}

	}
```

```
	// この中で描画をしない方がいい

	public void connectjarfile() {
		HttpConnection c;
		InputStream in;
		// OutputStream out;
		byte[] data;

		/** サウンド再生中はtrueのフラグ */
		boolean sound_flag;
		int jarsize;

		// TODO Auto-generated method stub
		try {

			c = (HttpConnection) Connector.open(
					"http://localhost/sound/jarsampleL.jar", Connector.READ,
					true);
			c.setRequestMethod(HttpConnection.GET);

			c.connect();
			filelength = (int) ((ContentConnection) c).getLength();
			System.out.println("filelength:" + filelength);
			c.close();
			System.out.println("150kb");

			for (int pos = 0; pos < filelength; pos=pos+130000) {
				c = (HttpConnection) Connector.open(
						"http://localhost/sound/jarDL.php?P_ID=" + P_ID
+
"&pos="
+ pos, Connector.READ, true);
				c.setRequestMethod(HttpConnection.GET);
				c.connect();


				in = c.openInputStream();

//				length = (int) c.getLength();

//				System.out.println("c.getlength:"+length);
				byte[] buf = new byte[130000];
//				System.out.println("buflength:"+buf.length);
				in.read(buf);
				in.close();
				c.close();
//				System.out.println("pos:"+pos);

				OutputStream out = Connector
						.openOutputStream("scratchpad:///0;pos=" + pos);

				out.write(buf);
				out.close();
			}
		} catch (Exception e) {
			e.printStackTrace();
		}

	}

	public void playjarfile(int O_ID) {

		JarInflater ji;


		try {
			InputStream in = Connector
					.openInputStream("scratchpad:///0;pos=0,length=" +
filelength);
			ji = new JarInflater(in);

			// ji = new JarInflater(Connector
```

```
                        // .openInputStream("scratchpad:///0;pos=0,length=" + length));

                        // ZIPファイル内のサウンドをとりだす

                        // プレイヤーの取得
                        ap = AudioPresenter.getAudioPresenter();
                        // 音データの取得
                        sound = MediaManager.getSound(ji.getInputStream("jarsample/" + O_ID
                                        + ".mld"));
                        // 音を使える状態にする
                        try {
                                sound.use();
                        } catch (Exception e) {
                                // エラーの場合
                                System.out.println("Error:" + e);
                        }
                        // 音データをセット
                        ap.setSound(sound);
                        // 音の再生
                        ap.play();

                        end = new Date();
                        long time = (end.getTime() - post.getTime())+(pre.getTime() -
start.getTime());
                        System.out.println(time+"¥t");

                        // ZIPファイルを閉じる
                        ji.close();
                        in.close();
                        System.out.println("playjarfile 終了");
                } catch (Exception e) {
                        e.printStackTrace();
                }

        }

}
```

```
//Doja用のサーバー
import java.awt.*;
import java.net.*;
import java.util.Enumeration;
import java.util.TooManyListenersException;
import java.io.*;

import javax.comm.CommPortIdentifier;
import javax.comm.PortInUseException;
import javax.comm.SerialPort;
import javax.comm.UnsupportedCommOperationException;

import com.sun.corba.se.pept.transport.Connection;

import rhlab.io.ExtraDataOutputStream;
import rhlab.rfid.tagit.TagITPacketData;
import rhlab.rfid.tagit.TagITRfidReader;
import rhlab.util.BitConverter;

public class DojaServerTest {
	public static void main(String[] args) throws Exception {
		process();
	}

	private static void process() throws IOException,
			TooManyListenersException, UnsupportedCommOperationException,
			PortInUseException {
		TagITRfidReader reader = null;
		ExtraDataOutputStream out;
		Enumeration portList = CommPortIdentifier.getPortIdentifiers();
		while (portList.hasMoreElements()) {
			CommPortIdentifier portId = (CommPortIdentifier) portList
					.nextElement();
			if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
				if (portId.getName().equals("COM1")) {
					reader = new TagITRfidReader((SerialPort) portId.open(
							"RfidApp", 2000));
					break;
				}
			}
		}
		TagITPacketData packet;
		// 読む
		packet = reader.readSingleBlock(0, null, 7);
		System.out.println(BitConverter.toUnsignedInt(packet.getData(), 3, 1));
		System.out.println(BitConverter.toUnsignedInt(packet.getData(), 2, 1));

		System.exit(0);
	}
}
```

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;


import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;

import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Image;
import com.nttdocomo.ui.Panel;
import com.nttdocomo.ui.Frame;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.util.Phone;
import com.nttdocomo.opt.ui.*;
import com.nttdocomo.ui.*;
import com.nttdocomo.io.*;



public class DojaClientTest extends IApplication {
    DojaCanvas c = new DojaCanvas();

          public void start() {
            Display.setCurrent( c );
            c.exe();
          }
        }

class DojaCanvas extends Canvas {
        HttpConnection  httpConn;               //HTTP接続オブジェクト
        InputStream             objInStream;    //入力オブジェクト
        OutputStream    objOutStream;   //出力オブジェクト
        private int O_ID, P_ID;

          void exe() {
                setSoftLabel(SOFT_KEY_1,"終了");
                setSoftLabel(SOFT_KEY_2,"Read");
          }
          void draw(Graphics g){
                  g.drawString("サーバーに接続完了",50,50);
          }

          void connect(){

                        try {

                                // HTTP 接続 (HttpConnection) オブジェクトを取得する。
//                              System.out.println("local");
                                httpConn =
(HttpConnection)Connector.open("http://localhost/reader/index.php",Connector.READ,true);

                                // リクエストメソッドを設定する。
                                httpConn.setRequestMethod(HttpConnection.GET);

                                // リモート資源に接続する。
                                httpConn.connect();
                                draw(this.getGraphics());


                                //データをInputStreamに
                                objInStream = httpConn.openInputStream();
                                BufferedReader bf = new BufferedReader(new
InputStreamReader(objInStream));
//                              P_ID =Integer.parseInt(bf.readLine());
//                              O_ID =Integer.parseInt(bf.readLine());
//                              System.out.println("P_ID : "+P_ID);
//                              System.out.println("O_ID : "+O_ID);
```

```
                                System.out.println("P_ID: "+bf.readLine());
                                System.out.println("O_ID: "+bf.readLine());



                                // 入力ストリームをクローズする。
                                objInStream.close();
                                // HTTP 接続をクローズする。
                                httpConn.close();


                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                System.out.println(e.toString());
//                              e.printStackTrace();
                        }

          }
            public void processEvent(int type,int param) {
                if (type==Display.KEY_PRESSED_EVENT) {
                    //終了
                    if (param==Display.KEY_SOFT1) {
                        System.exit(0);
                    }
                    //タグを読む
                    else if (param==Display.KEY_SOFT2) {

                            this.connect();

                    }
                }
            }

          //この中で描画をしない方がいい
        public void paint(Graphics arg0) {
                // TODO Auto-generated method stub

        }
        }
```

```
import java.awt.*;
import java.net.*;
import java.util.Enumeration;
import java.util.TooManyListenersException;
import java.io.*;

import javax.comm.CommPortIdentifier;
import javax.comm.PortInUseException;
import javax.comm.SerialPort;
import javax.comm.UnsupportedCommOperationException;

import com.sun.corba.se.pept.transport.Connection;

import rhlab.io.ExtraDataOutputStream;
import rhlab.rfid.tagit.TagITPacketData;
import rhlab.rfid.tagit.TagITRfidReader;
import rhlab.util.BitConverter;

public class HTTPServer extends Frame {
	private ServerSocket server;
	private Socket s;

	private DataInputStream input;// 携帯からの命令ストリーム
	private DataOutputStream output;// 携帯へタグNoを返す出力ストリーム

	public boolean done;

	TextArea textarea = new TextArea("", 15, 30);

	TagITRfidReader reader = null;

	public static void main(String[] args) throws Exception {
		HTTPServer serv = new HTTPServer();
	}

	private void initReader() throws IOException, TooManyListenersException,
			UnsupportedCommOperationException, PortInUseException {
		// test引用

		ExtraDataOutputStream out;
		Enumeration portList = CommPortIdentifier.getPortIdentifiers();
		while (portList.hasMoreElements()) {
			CommPortIdentifier portId = (CommPortIdentifier) portList
					.nextElement();
			if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
				if (portId.getName().equals("COM1")) {
					reader = new TagITRfidReader((SerialPort) portId.open(
							"RfidApp", 2000));
					break;
				}
			}
		}

	}

	public HTTPServer() throws IOException, TooManyListenersException,
			UnsupportedCommOperationException, PortInUseException {

		super("Server");
		server = new ServerSocket(5000, 1);

		setSize(270, 350);
		setLayout(new FlowLayout());
		add(textarea);
		initReader();
		TagITPacketData packet;
		setVisible(true);
		done = false;
		s = server.accept();
		try {
			input = new DataInputStream(s.getInputStream());//入力用
			output = new DataOutputStream(s.getOutputStream());//出力用
```

```
                        ;//サーバーの待機開始
                        //接続すると次へすすむ↓
                        textarea.append("接続完了"); //stramが１であることをダイアログに表示

                        while (!done) {// フラグがdoneでは無い間
                         if(input.readInt()==1){//もし受信した命令ストリームが１（読め）だったら
                                textarea.append("１でした¥n"); //stramが１であることをダイアログ
に表示

                                // 読む
                                packet = reader.readSingleBlock(0, null, 7);
                                textarea.append("tag:"
                                                + BitConverter.toUnsignedInt(packet.getData(),
3, 1));//3ブロック目の長さ１ビットのデータを読んでいる
                                output.writeInt(BitConverter.toUnsignedInt(packet.getData(), 3,
1));
                                output.writeInt(BitConverter.toUnsignedInt(packet.getData(), 2,
1));
                                textarea.append("読んだ値を送信しました¥n");


                                 }

                        }
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        }

        public boolean handleEvent(Event ev) {
                if (ev.id == Event.WINDOW_DESTROY)
                        System.exit(0);
                return false;
        }

}
```