B-26

待ち受けiアプリの設計と評価
Design and Evaluation of a Stand-by i-application

津田 雅之 * 杉村 利明 * 倉掛 正治 *
Masayuki Tsuda Toshiaki Sugikimura Shoji Kurakake

1. はじめに

弊社携帯電話504iシリーズで実装し、サービスを開始したiアプリ待受画面(以下、待ち受けiアプリと呼ぶ)の設計と評価について述べる。

待ち受けiアプリとは、Java言語でプログラムされた弊社携帯電話向けアプリケーションプログラム[1]であり、携帯電話のトップ画面である待受画面上で実行される常駐型プログラムのことである。その開発においては、通話、ブラウザ、メールといった電話機ネイティブ機能とJavaの実行環境を共存させた上で、携帯電話でのJavaによる情報処理機能を高めることを目的としている。待ち受けiアプリの実現により、電話の待ち受け状態を利用した自動的な情報の取得やユーザーへのリアルタイムなイベント通知、さらには携帯電話上で情報の統合が可能となり、情報処理端末としての携帯電話の役割が高まることを我々は期待している。

待ち受けiアプリを実現する上での課題は、電話機ネイティブ機能とJava実行環境との共存である。具体的には、

・キー入力の配送制御
・省電力の実現

の2点を解決する必要がある。キー入力の配送制御とは、Javaと電話機能へのキー入力の競合に起因した課題である。キー入力が電話機ネイティブ機能のみに配送されるのであれば、Java実行環境でのキー操作に制限がかかり、結果的にアプリケーションに制限をかけることになる。しかし、待ち受けiアプリにのみキー入力が配送されるのであれば、従来の待受画面のように電話機ネイティブ機能をワンタッチで呼び出すことが困難となり、操作性を落してしまう。省電力の実現も待ち受けiアプリを実現する上で重要な課題となる。従来の待受画面への静的な画像の表示と異なり、待ち受けiアプリでは常時Javaを実行するため電池の消耗が激しくなる。待ち受けiアプリにより数時間しか電池がもたないのであれば、ユーザーに待ち受けiアプリを使ってもらうことは難しい。

これら課題に対して、我々は、待ち受けiアプリに活性化状態(キーがJava側に配送される状態)、非活性化状態(キーが電話機能に配送される状態)、休眠状態(省電力の実現のためにVMを停止している状態)を定義し、アプリケーション自身もしくはユーザーの操作でこれら状態間の遷移が行えるように待ち受けiアプリのクラスを設計した。

以下、2章で状態とその遷移、およびそれら状態遷移を反映した待ち受けiアプリのクラス設計について述べ、3章で上記課題に対する非活性、休眠状態導入の効果についての評価結果を示す。

*(株)NTTドコモ マルチメディア研究所, NTT DoCoMo, Inc. Multimedia Laboratories

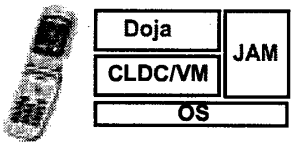

図1: An overview of a java system in a cellular phone

2. 待ち受けiアプリの設計

2.1 iアプリの基本ライフサイクル

504i以前ではiアプリを待受画面に設定することはできず、iアプリのライフサイクルもスタート状態、終了状態、活性化状態、サスペンド状態の4つの状態から構成されていた。iアプリは起動されるとすぐに活性化状態に遷移し、音声着信などによる割込みでサスペンド状態に遷移する以外は終了状態まで活性化状態で実行し続ける。キー入力はすべてJava側に配送される。

図1に示すように弊社携帯電話に実装されているJava実行環境は、Javaのコアクラスである CLDC(Connected Limited Device Configuration) と弊社が開発したiアプリ用クラス(本報告ではDoJaと呼ぶ)および実行環境を制御するアプリケーションマネジャー(以下JAMと呼ぶ)から構成される[2]。iアプリのライフサイクルはDoJaの中で弊社が独自に規定している。

2.2 状態の追加

Java実行環境と電話機ネイティブ機能の共存で課題となる「キー入力の配送制御」と「省電力」を実現するために、待ち受けiアプリに非活性化状態と休眠状態をiアプリのライフサイクルの中に追加し、待ち受けiアプリのライフサイクルを新たに設計した。以下にライフサイクルのキーとなる活性化状態、非活性化状態、休眠状態の各状態を示す。

・活性化状態

待ち受けiアプリが通常のiアプリと同様に実行され、キー入力はすべてJava側に配送されて待ち受けiアプリ側で処理される状態。例えば、この状態でコンポーネントのListBoxでアイテムの入力待ちである場合には、ユーザーの数字キーの押下はアプリケーションに通知され、押下した数字に対応するアイテムが選択される。

・非活性化状態

ユーザー操作によるキー入力は電話機ネイティブ機能側に配送され、Java側に通知されない状態。例えば、UI的にはあってはならない場合であるが、この状態でコンポーネントのListBoxでアイテムの入力待ちである場合、ユーザーが数字キーを押下すると、即アプリケーションはサスペンドされて電話の発信待ちとなる。

・休眠状態

Javaの実行環境をJAMも含めて停止する状態。休眠

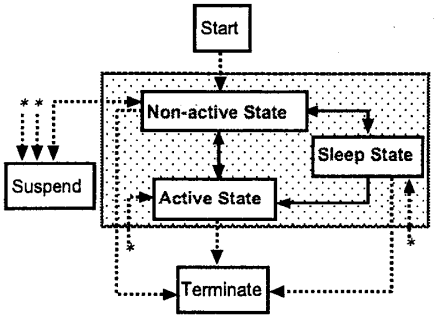

図 2: State transition of a Stand-by i-application

状態に遷移する際、すべての処理を一時停止する。休眠状態から非活性化、活性化状態に遷移した際には一時停止中の処理を自動的に再開される。

2.3 状態遷移

図 2に示すように前節の述べた状態間を遷移しながら待ち受けiアプリは実行される。各遷移において、Java実行環境の設計・実装者が独自の判断でアプリの実行とは関係なく状態を遷移させてしまっては、待ち受けiアプリは開発者にとって非常に開発しづらいものとなる。また、ユーザーにとっても使いづらいものとなる。これを回避するために、我々は状態遷移をプログラム側やユーザー側に可能な限り任せることにした。具体的には、メソッドやイベントの通知により遷移や遷移状態をプログラム側で実行もしくは把握できるようにクラスを設計した。以下に状態遷移の概要を示す。

・休眠状態、非活性化状態から活性化状態への遷移

ユーザー操作が必ず反映できるようにユーザーが特定のキーを押下することで自動的に活性化状態への遷移させる。遷移した結果はイベントとしてアプリケーションに通知される。

・非活性化状態、活性化状態から休眠状態への遷移

非活性化状態から休眠状態へはメソッドを呼ぶことで遷移させる。なお、ユーザーを困惑させるプログラムを禁止するために活性化状態から休眠状態への遷移は禁止している。

・活性化状態から非活性化状態への遷移

アプリケーションがメソッドを呼ぶことで活性化状態から非活性化状態へ遷移させる。

・休眠状態から非活性化状態への遷移

2つ折りの携帯電話の場合には電話を開いたことやメソッドを通じて指定した時間の経過を契機に休眠状態から非活性化状態に遷移させる。また、メソッドの指定により電話機ネイティブ機能側で時計が1分経過した場合に休眠状態から非活性化状態に遷移させる。遷移した結果はイベントとしてアプリケーションに通知される。

2.4 クラス設計

上述したライフサイクルをもったクラスとして待ち受けiアプリクラスを設計した。クラス設計においては、待ち受けiアプリがiアプリのライフサイクルを拡張していることやプログラミングスタイルの統一性という観点から、待ち受けiアプリのクラス (MApplication クラス) はiアプリのクラス (IApplication クラス) を継承させることにした。また、DoJa での基本ポリシーとして1アプリケーションしか実行させないため、MApplication クラスでも IApplication クラス同様に、そのインスタンスの生成は Java 実行環境側でしか許していない。

前節で示した状態遷移を実現するために MApplication クラスには以下のメソッドを定義した。

- deactivate():非活性化状態へ遷移させるメソッド
- sleep() :休眠状態へ遷移させるメソッド
- setWakeUpTimer():休眠状態から非活性状態へ遷移させるタイマーを設定するメソッド
- getWakeUpTimer():タイマーの残り時間の取得するメソッド
- resetWakeUpTimer():タイマを解除するメソッド
- setClockTick():電話機ネイティブ側の時計イベントの通知を有効にするメソッド
- processSystemEvent():各種イベントのハンドラー

3. 評価実験

本報告では紙面の制限から休眠状態での省電力効果の実験結果のみを示す。非活性化状態での操作性の評価結果は当日報告する。

3.1 実験プログラム

休眠状態追加による省電力効果を調べるために描画プログラムを作成し、(1) 活性化状態での無限ループでの実行 (通常のiアプリとして実行) と (2)4 分ごとに休眠状態から非活性化状態に遷移させての実行 (待ち受けiアプリとして実行) の 2 種類を行い、電池のもち時間を比較した。(1) では 1 分ごとに (2) では 4 分ごとに経過時間を携帯電話内のログ領域に保存させることで、時間の計測を行った。実験には F504i、SO504i を用いた。

3.2 実験結果と評価

通常のiアプリとして実行した場合には、F504i では約 6 時間、SO は約 15 時間で電池切れとなってしまう。一方、待ち受けiアプリとして実行した場合では、F504i では約 6 時間、SO では約 15 時間を経過しても電池切れとはならずに動作し続けている。このことから休眠状態を導入することで待ち受けiアプリにおいて省電力を実現できたものと考えられる。

4. おわりに

iアプリのライフサイクルに新たな状態 (非活性化状態、休眠状態) を追加することで Java 実行環境と電話機ネイティブ機能との共存時の課題を解決して、待ち受けiアプリを実現した。そして、状態遷移がアプリ側で制御もしくは把握可能な待ち受けiアプリクラスを設計し、弊社 504i シリースに実装した。評価実験の結果から休眠状態を導入したことで省電力の効果が得られたことがわかった。

参考文献

[1] NTT ドコモ, “i アプリコンテンツ開発ガイド for 504i”,2002.5

[2] 津田, 杉村, “携帯電話向けコンテンツ記述言語の動向”, 電子情報通信学会会誌, vol.84,pp.363-369, 2001.11