PHS通信モジュールW-SIMを利用したLinuxベース携帯電話の開発

第2回

PHS携帯電話アプリケーション・ソフトの開発

田中岳彦
河合孝時

今回から2回にわたって，携帯電話のアプリケーション・ソフトウェアの開発について取り上げる．開発環境として，ソフィアシステムズのLinuxベースPHS携帯電話開発キット「Sandgate W-SIM Phone」を利用する．本キットの開発環境の概要，開発できるアプリケーション・ソフトウェア，GTK+を用いて作成したソフトウェアを本開発キット上で動作させるための修正方法，音声端末の開発において考慮すべき事項と対策などを解説する．（編集部）

Sandgate W-SIM Phoneは，W-SIMを利用したソフィアシステムズ製の音声端末向けの開発キットです．本開発キットには，音声端末を実現するためのミドルウェア・ライブラリと評価用サンプル・ソフトウェアが含まれます．OSにはLinuxを採用しています．Linuxの豊富な資産とミドルウェアを組み合わせて利用することで，音声端末向けのソフトウェアを低コストで開発できます．

Sandgate W-SIM Phoneで開発できるアプリケーション

評価用サンプル・ソフトウェアの中には，ソース・ファイルが用意されているものもあり，機能追加や改良を行うことができます．**図1**に本開発キットのソフトウェア構成を示します．

本開発キットを利用することで，W-SIMを利用した音声端末に必要な以下のソフトウェア開発を行うことができます．

1）アプリケーション・ソフトウェアの新規作成
2）評価用サンプル・ソフトウェアの改造と機能追加
3）ミドルウェアの追加
4）デバイス・ドライバの改良と新規作成
5）カーネルの改良

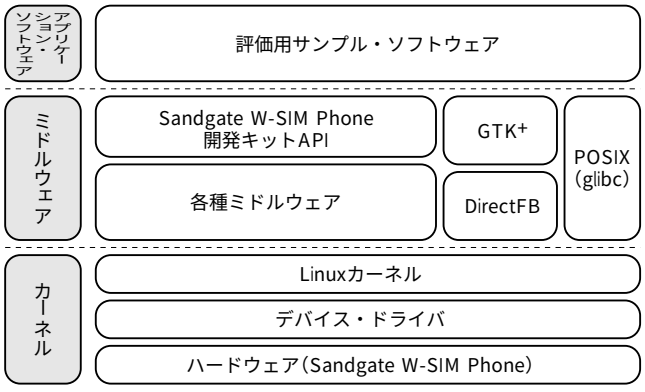

図1　Sandgate W-SIM Phone開発キットのソフトウェア構成

本開発キットには，音声端末を実現するためのミドルウェア・ライブラリと評価用サンプル・ソフトウェアが含まれる．Linuxの豊富な既存資産とミドルウェアを利用して，音声端末向けアプリケーション・ソフトウェアを開発することができる

● ホスト・パソコンとW-SIMで開発環境を構築

本開発キットを利用して音声端末のソフトウェアを開発するには，Linuxをインストールしたホスト・パソコンとW-SIMを用意する必要があります．本開発キットを**図2**のように接続して，ソフトウェアを開発します．

開発に必要な機材を以下に示します．

●本開発キットに付属するもの
1）Sandgate W-SIM Phone実機
2）デバッグ・ボード
3）RS-232-Cクロス・ケーブル
4）LANクロス・ケーブル
5）ACアダプタ
6）USBケーブル

●別途用意するもの
1）ホスト・パソコン
　シリアル・ポート（RS-232-C）×1
　LANカード（実機接続用の専用LANカード）×1

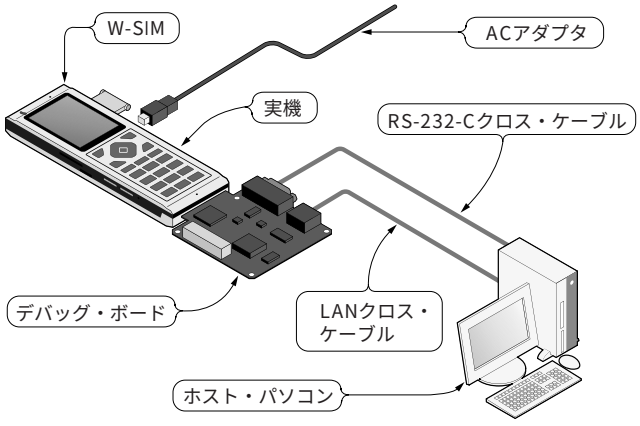

図2　Sandgate W-SIM Phoneとホスト・パソコンの接続

Sandgate W-SIM Phone実機とLinuxをインストールしたホスト・パソコンを，RS-232-Cクロス・ケーブルとLANクロス・ケーブルで接続する．W-SIMは別途用意する必要がある

KEYWORD ── PHS，W-SIM，Linux，Sandgate W-SIM Phone，クロス開発環境，アプリケーション・ソフトウェア，GTK+，待ち受け，ダイヤラ，メーラ，PIM，FEP

Linux（Fedora Core4 フル・インストールを推奨）

2）W-SIM

Linux をインストールしたホスト・パソコンに，クロス・コンパイル環境と実機をホスト・パソコンからネットワーク・ブートする環境（**図3**）を構築します．このネットワーク・ブート環境上で動作させることにより，作成したソフトウェアの動作確認を行います．また，クロス・コンパイル環境で作成したシステムを，実機上のフラッシュ ROM に書き込むことで，通常の音声端末と同じようにスタンドアロン（実機のみの環境）で動作させることもできます．

● GTK+ を利用してソフトウェアを開発できる

本音声端末のアプリケーション・ソフトウェアは，オープン・ソースで広く利用されている GTK+ [注]のインターフェースを利用して GUI（Graphical User Interface）を構築し，ミドルウェア API によりハードウェアの各機能を利用します．通常は C 言語でアプリケーション・ソフトウェアを作成します．アプリケーション・ソフトウェアとミドルウェア API の関係を**図4**に示します．

本開発キットのアプリケーション・ソフトウェアには，以下の特徴があります．

1）GTK+ インターフェースにより，GUI を容易に作成できる
2）ミドルウェア API により，ハードウェアの各機能を容易に利用できる
3）アプリケーション・フレームワークにより，ほかのアプリケーション・ソフトウェアやミドルウェアと容易に連携させることができる
4）Linux の豊富な資産を活用できる

● サンプル・ソフトウェアが用意されている

本開発キットに搭載されている主なサンプル・ソフトウェアについて説明します．

「待ち受け」は，システム起動後の最初の画面です．すべてのアプリケーション・ソフトウェアの中の最下位層に位置します．日時の表示やメイン・メニューへ遷移する機能を提供します．Windows のデスクトップに相当します〔**図5**（**a**）〕．

「ダイヤラ（無線モデム通信ソフトウェア）」は，発信・着信を行います．発信は待ち受けからの数字キーで遷移します〔**図5**（**b**）〕．

「メーラ（メール管理ソフトウェア）」はその名のとおりメールの送受信を行うアプリケーション・ソフトウェアです．メイン・メニューから，あるいは待ち受けからのショートカット・キーによって遷移します〔**図5**（**c**）〕．

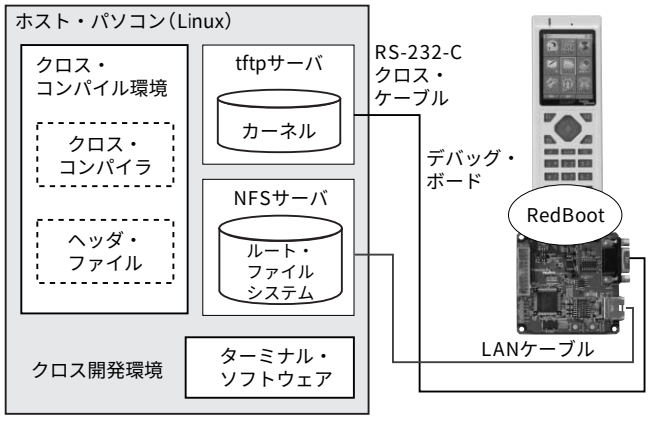

図3　ネットワーク・ブートを利用した開発環境の構成

実際の開発は，ホスト・パソコンと実機を接続し，ネットワーク環境で行う．ホスト・パソコンから TFTP（Trivial File Transfer Protocol）でカーネルをロードし，ルート・ファイル・システムは NFS（Network File System）マウントして使用する．クロス・コンパイル環境は開発キットに含まれる

注：GTK+（GIMP Toolkit）は，GUI ベースのアプリケーション・ソフトウェアを開発するための C 言語ライブラリ．もともと，GIMP（GNU Image Manipulation Program）というフリーの画像編集ソフトウェアを開発するために作られた．ライセンスは GNU LGPL（GNU Lesser General Public License）となっているため，入手や改変，再配布が可能．GTK+ には，ラベルやテキスト・ボックス，チェック・ボックス，メニュー，ツール・バー，ツリー・ビューなど，さまざまな部品（ウィジェット）が標準で用意されている．また，GTK+ の新しいウィジェットも簡単に開発することができる．GTK+ や GIMP の詳細については，以下の Web サイトを参照されたい．

- GTK+（http://www.gtk.org/）
- GIMP（http://www.gimp.org/）

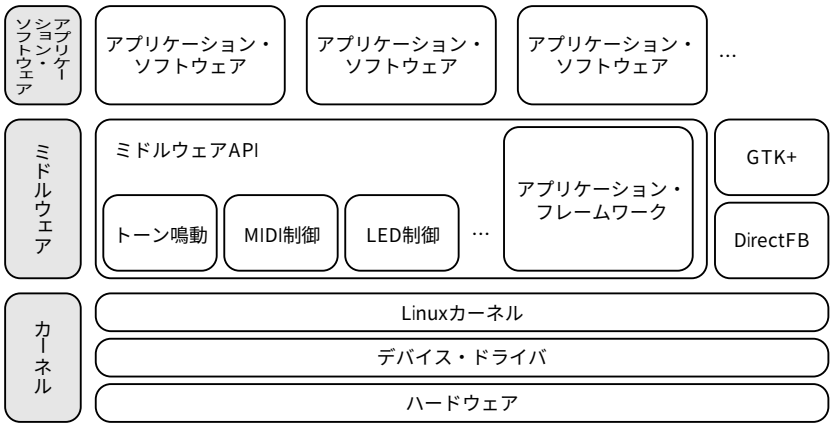

図4
アプリケーション・ソフトウェアとミドルウェア API の関係

アプリケーション・ソフトウェアは，GTK+ の機能を利用して GUI を構築し，ミドルウェア API によりハードウェア機能を利用する．また，アプリケーション・フレームワークの機能を利用して，ミドルウェアやほかのアプリケーション・ソフトウェアとの通信を行う

(**a**) 待ち受け　(**b**) ダイヤラ

(**c**) メーラ　(**d**) PIM

(**e**) Java　(**f**) FEP

図5　サンプル・ソフトウェアの利用画面

本開発キットには，音声端末に必要な基本機能に対応するサンプル・ソフトウェアが用意されている

「PIM（Personal Information Manager）」は，カレンダやスケジュール帳，電話帳の管理を行います．メイン・メニューから，あるいは待ち受けからのショートカット・キーによって遷移します〔**図5**(**d**)〕．

「Java」はJavaVMとして機能し，Javaのアプリケーション・コードを実行する環境を提供します．メイン・メニューから，あるいは待ち受けからのショートカット・キーによって遷移します〔**図5**(**e**)〕．

「FEP（Front-end Processor）」は日本語文字入力を支援します．FEPはメーラ，PIM，Java，機能設定など，さまざまなアプリケーション・ソフトウェアで利用されます〔**図5**(**f**)〕．

2 アプリケーションの種類と開発の流れ

本開発キットに搭載されているソフトウェアはシステム・アプリケーション，ベンダ・アプリケーション，追加アプリケーションの3種類に分類されます．

表1　システム・アプリケーションの一覧

システム・アプリケーション	ソース・ファイル[注]
起動アニメーション	○
待ち受け	×
ダイヤラ	×
発着信履歴	○
機能設定	×
データ・フォルダ	×
追加アプリケーション	○
各種メニュー	○
電源OFF/終了アニメーション	×

注：○は添付，×はバイナリのみ

表2　ベンダ・アプリケーションの一覧

ベンダ・アプリケーション[注]	提供元	ソフトウェア名
メーラ	富士通ビー・エス・シー	Be Star Mail
PIM	富士通ビー・エス・シー	Be Star PIM
Java	Tao Group社	intent
Webブラウザ	Tao Group社	intent
文字入力（エンジン部分）	オムロンソフトウェア	Advanced Wnn Ver1.20

注：ベンダ・アプリケーションのソース・ファイルは添付されない

● 基本機能を実現するためのシステム・アプリケーション

システム・アプリケーションは，音声端末の基本機能を実現するためのソフトウェアで，システムに静的に組み込まれています．ほかのアプリケーション・ソフトウェアが，後述するアプリケーション・フレームワークに起動要求を行うと，アプリケーション・フレームワークから起動されます．

本開発キットには，**表1**のシステム・アプリケーションが搭載されています．

● 目的に応じて搭載するベンダ・アプリケーション

ベンダ・アプリケーションは，音声端末の利用目的に応じてオプションとして搭載されるソフトウェアです．メーラやPIMなどのソフトウェアがこれに該当します．ベンダ・アプリケーションは，システム・アプリケーションと同じようにシステムに静的に組み込まれます．また，アプリケーションの種類別（メーラ機能，Java機能など）に，ほかのソフトウェアとのインターフェースが統一されます．そのため，端末の利用目的に合わせて別のソフトウェアに静的に差し替えることができます．システム・アプリケーションと同じように，ほかのソフトウェアがアプリケーション・フレームワークに起動要求を行うと，アプリケーション・フレームワークから起動されます．

本開発キットには，**表2**のベンダ・アプリケーションが搭載されています．

● 動的にインストールできる追加アプリケーション

追加アプリケーションは，「データフォルダ」からシステムに

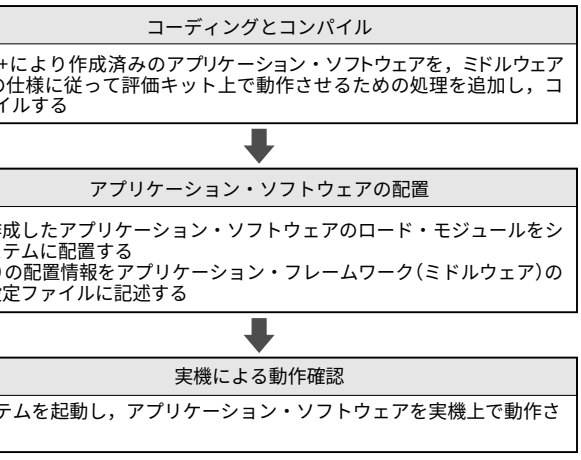

図6　システム・アプリケーションとベンダ・アプリケーションの開発の流れ

システム・アプリケーションとベンダ・アプリケーションは，静的にシステムに配置される

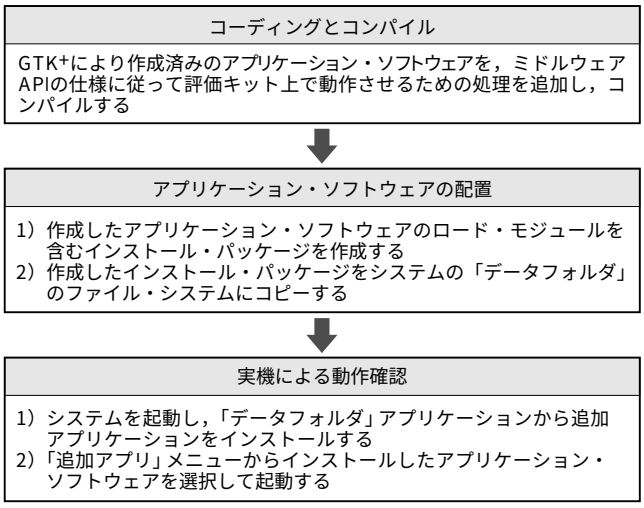

図7　追加アプリケーションの開発の流れ

追加アプリケーションは，「追加アプリ」メニューから動的にインストールできる

動的にインストールできるアプリケーション・ソフトウェアです．インストールしたアプリケーションは，「追加アプリ」メニューから起動することができ，アンインストールすることもできます．追加アプリケーションの機構は，作成したアプリケーション・ソフトウェアをダウンロードなどにより，システムに動的に追加できるしくみとして用意されています．

● アプリケーション・ソフトウェア開発の流れ

GTK+ を利用してすでに作成済みのアプリケーション・ソフトウェアを本開発キット上で動作させる際の手順を中心に，アプリケーション・ソフトウェア開発の流れについて説明します．システム・アプリケーションとベンダ・アプリケーションは，**図6**のような流れで開発します．

一方，追加アプリケーションの場合は，アプリケーションを動的にインストールできるようにするため，**図7**のような流れで開発します．なお，追加アプリケーションはパッケージ形式にする必要がありますが，コーディングおよびコンパイルの時点ではシステム・アプリケーションとの違いはありません．

3 アプリケーション・フレームワークのさまざまな管理機構

アプリケーション・ソフトウェアは，アプリケーション・フレームワークが提供する四つの管理機構（アプリケーション管理，メッセージ通信，ウィンドウ管理，電話機状態管理）のもとで動作し，また，これらの機能を利用できます（**図8**）．アプリケーション・フレームワークとは，メッセージ通信によるプロセス間通信機能，および各種ミドルウェアに対する処理依頼を行う機能などを備えたライブラリ群のことです．

音声端末上のアプリケーション・ソフトウェアの開発は，通常のアプリケーション・ソフトウェアと異なり音声端末ならではの考慮が必要となります．詳細は後述しますが，音声端末ならではの動作も，アプリケーション・フレームワークを利用することで可能となります．

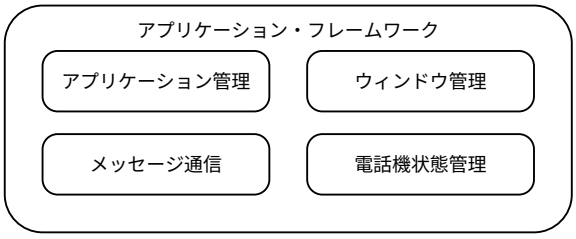

図8　アプリケーション・フレームワークの機能

アプリケーション・フレームワークは，アプリケーション管理，メッセージ送信，ウィンドウ管理，電話機状態管理の四つの機能から構成される

● アプリケーション管理：アプリの起動や終了を管理する

アプリケーション管理は，アプリケーション・フレームワーク上で動作するアプリケーション・ソフトウェアやミドルウェアの起動，および終了を管理します．アプリケーション・ソフトウェアは，それぞれ固有のアプリケーションIDで識別します．そのため，個々のアプリケーション・ソフトウェアには，固有のアプリケーションIDを割り当てる必要があります．このIDを使用してアプリケーション・ソフトウェアの起動，およびアプリケーション・ソフトウェア間の通信が行われます．アプリケーションIDは16ビット（2バイト）で構成され，**表3**に示す区分に従って割り当てを行います．

アプリケーション・ソフトウェアは，初期化するときにアプリケーションIDを指定してアプリケーション・フレームワーク初期化API（`wapc_init`関数）を呼び出し，動作環境の初期化とアプリケーションIDの登録をかならず行う必要があります．

表3　アプリケーションIDの割り当て

アプリケーションID	説　明
0x0000 ～ 0x0FFF	アプリケーション・フレームワーク用
0x1000 ～ 0x1FFF	ミドルウェア用(通信インフラ依存ミドルウェア)
0x2000 ～ 0x3FFF	ミドルウェア用
0x4000 ～ 0x4FFF	システム・アプリケーション用
0x6000 ～ 0x6FFF	ベンダ・アプリケーション用
0x8000 ～ 0x8FFF	追加アプリケーション用

リスト1　ウィンドウの登録

```
ウィンドウの登録
ret = wapc_window_register(window, WINDOW_TYPE_NORMAL, NULL );
if( ret != MSGAPC_OK ){
    printf("APP ERROR WINDOW REGISTER \n");
    return -1;
}

/* wapc_window_register関数の第2引き数で終了タイプ(画面終了時の動作)
を指定します */
```

表4　ウィンドウ終了時の動作

終了タイプ	説　明
WINDOW_TYPE_NORMAL	終了の種別にかかわらず，アプリケーションに終了指示が行われる
WINDOW_TYPE_ABORT	登録したウィンドウが最前面に表示されている場合は，WINDOW_TYPE_NORMALと同じ動作となる． 最前面に表示されていない場合は，登録したウィンドウが最前面に表示された状態で，終了処理が完了する
WINDOW_TYPE_GUARD	電源キー，W-SIM挿抜による終了処理の場合，登録したウィンドウが最前面になった時点で終了処理が完了する
WINDOW_TYPE_BYPASS	終了の種別にかかわらず，登録したウィンドウに対する終了指示は行わない．登録したウィンドウが最前面に表示された状態でも終了処理は完了せず，下側に位置するほかの画面に対する終了処理は継続して行われる

● ウィンドウ管理：画面を制御する

音声端末では，「待ち受け」→「メイン・メニュー」→「設定画面」というように複数のアプリケーション・ソフトウェアを連続起動した状態でクリア・キーが押されると，一つ前の画面に戻ります．一方，電源キーが押された場合は，一度に待ち受けに戻ります．ウィンドウ管理は，このような画面制御を容易に実現するための機能を提供します．

画面をもつアプリケーション・ソフトウェアは，初期化するときにアプリケーション・フレームワークに対して，画面(GTK+のウィンドウ)と画面制御の方法(終了タイプ，詳細は**表4**)を登録します．**リスト1**にコーディング例を示します．

● メッセージ通信：プログラム間通信を提供する

本開発キット上のアプリケーション・ソフトウェアおよびミドルウェアは，アプリケーション・フレームワークが提供するメッセージ通信機構を利用して通信を行います．同期通信型と非同期通信型の二つのメッセージ通信方式がサポートされています．

常駐しないアプリケーション・ソフトウェアの起動は，起動要求オプションを付加したメッセージの送信により行います．起動要求オプションを送信したとき，送信先のアプリケーション・ソフトウェアが起動していない場合は，アプリケーション・フレームワークがアプリケーション・ソフトウェアを起動し，メッセージ送信を行います．

表5　電話機状態の一覧

状態種別	状　態
W-SIM状態	挿入/未挿入
イヤホン状態	挿入/未挿入
通話状態	通話中/非通話中
USB状態	挿入/未挿入
電源断可能状態	電源断可能/電源断不可能
バッテリ状態	残量未取得/残量50%以上/10%未満など
充電状態	未取得/充電中/充電完了/充電なし

● 電話機状態管理：電話機の状態変化を通知する

電話機状態管理は，電話機の状態変化をアプリケーション・ソフトウェアに通知する機能です．バッテリの残量の変化に応じて，待ち受け画面上部の電池残量マークの表示を変更する場合などに使用されます．電話機状態として管理できる状態を**表5**に示します．電話機状態が変化すると，アプリケーション・ソフトウェアに対して，状態変化を通知するメッセージが送信されます．アプリケーション・ソフトウェアから電話機状態を動的に確認することもできます．

4 GTK+アプリケーションを動作させるための修正方法

GTK+アプリケーション・ソフトウェアをアプリケーション・フレームワーク上で動作させるための改造箇所を中心に，音声端末アプリケーション・ソフトウェアの詳細について説明します．

● アプリケーションの初期化前にコールバック関数を登録

GTK+の初期化処理gtk_init関数を呼び出す前にアプリケーション・ソフトウェアの動作環境の初期化とメッセージ到着時のコールバック関数の登録を行います(**図9**)．表示する画面(ウィンドウ)の準備が整ったら，アプリケーション・フレームワークに対してウィンドウ登録を行います．

● 登録したメッセージ処理関数内でメッセージを受信

ほかのアプリケーション・ソフトウェアやミドルウェアからメッセージが到着すると，wapc_init_msg_func_register関数でコールバック関数として登録したメッセージ処理関数が自動的に呼び出されます(**図10**)．このメッセージ処理関数内

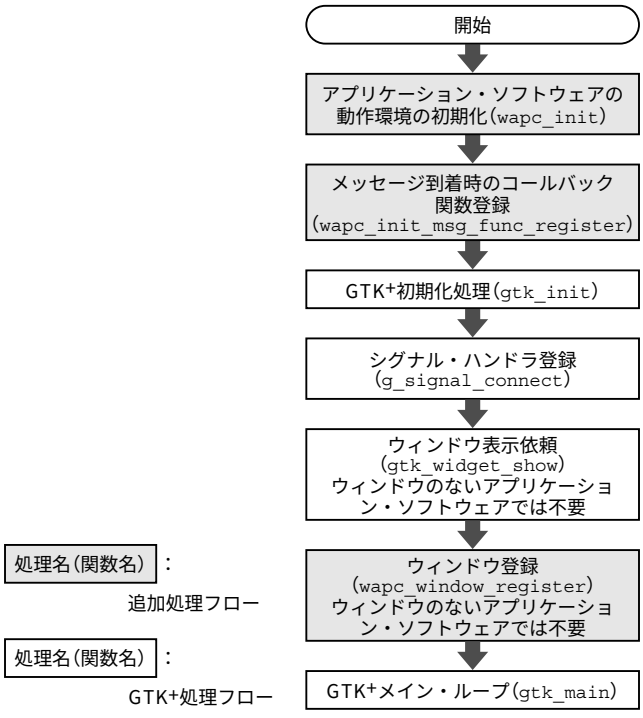

図9　アプリケーションの初期化処理フロー

アプリケーション・フレームワークを使用するアプリケーション・ソフトウェアは，GTK+ 初期化(gtk_init)の前に，フレームワーク登録処理(wapc_init)，およびメッセージ・コールバック関数の登録処理(wapc_init_msg_func_register)を行う

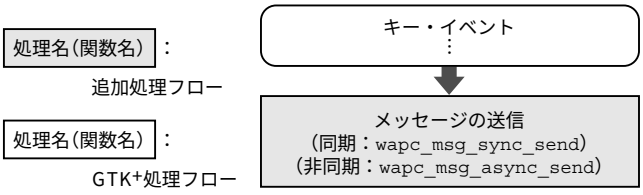

図11　メッセージ送信時の処理フロー

メッセージ送信処理は，通常のアプリケーション・ソフトウェアにおいてはキー・イベントを契機に行う．同期メッセージと非同期メッセージを適宜使い分ける．ほかのアプリケーションを起動する場合は，起動要求付きメッセージを送信する

で到着したメッセージを受信し，メッセージに応じた処理を行います．

● キー・イベント処理関数内でメッセージを送信

メイン・メニューでキー操作に応じてアプリケーション・ソフトウェアを起動する場合，アプリケーション・ソフトウェアは，受け取ったキー・イベントを契機にほかのアプリケーション・ソフトウェアへメッセージを送信します．このとき，GTK+のキー・イベントのコールバック関数内で，アプリケーション・フレームワークのメッセージ送信 API を使用してメッセージ送信を行います(**図11**)．

● 終了処理：メイン関数から復帰した後に動作環境を解放

GTK+ を利用して作成したアプリケーション・ソフトウェア

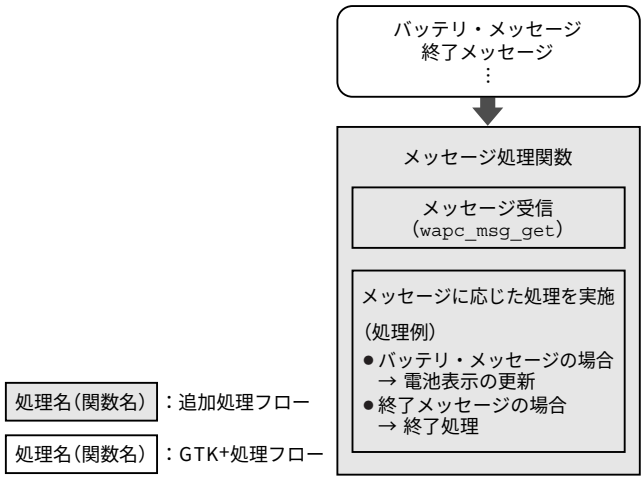

図10　メッセージ受信時の処理フロー

アプリケーション・ソフトウェアがメッセージを受信すると，初期化時に登録したコールバック関数が呼ばれる．アプリケーション・ソフトウェアは，コールバック関数内でメッセージに応じた処理を行う．代表的なメッセージは一括終了要求

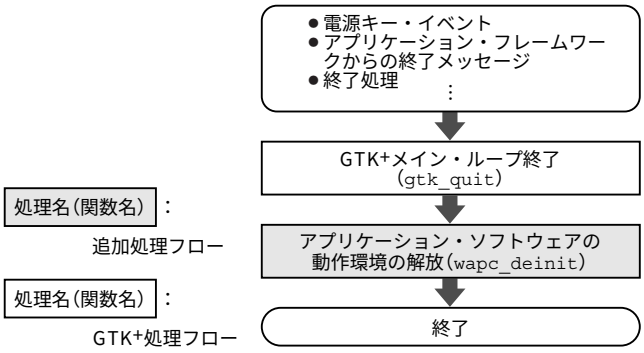

図12　アプリケーション・ソフトウェアの終了処理フロー

各種終了イベントにより，アプリケーション・ソフトウェアは終了処理を行う．アプリケーション・ソフトウェアが終了する際は，GTK+ メイン・ループの終了処理(gtk_main_quit())を行った後，アプリケーション・フレームワークからの削除

の終了処理では，gtk_main 関数から復帰した後に，アプリケーション・ソフトウェアの動作環境の解放を行います(**図12**)．アプリケーション・ソフトウェアの動作環境を解放した後は，メッセージの送受信などのアプリケーション・フレームワークの機能は使用できなくなります．

● 順番に終了要求してアプリケーションを一括終了

電源キーを押したことをアプリケーション・ソフトウェアが検出した場合，アプリケーション・フレームワークに対して，アプリケーション・ソフトウェアの一括終了を要求する必要があります．要求を受け付けたアプリケーション・フレームワークは，最前面のアプリケーション・ソフトウェアから最背面のアプリケーション・ソフトウェアまで，順番に終了要求メッセージを送信します(**図13**)．

アプリケーション・ソフトウェアの一括終了の詳細は，次節

図13　アプリケーション・ソフトウェアの一括終了処理フロー

通常のアプリケーション・ソフトウェアは，電源キーが押された場合に各アプリケーション・ソフトウェアに対して一括終了要求を送信する必要がある．電源キー長押し，W-SIM 挿抜，および低電圧の終了メッセージは，ミドルウェアより送信される

で説明します．

C 音声端末アプリケーションが考慮すべき事項と対策

音声端末向けのアプリケーション・ソフトウェアでは操作性に配慮し，音声端末特有の考慮を行わなければならない部分があります．Linux の通常の GUI 処理とは異なるので注意が必要です．ここでは，音声端末向けのアプリケーション・ソフトウェアを作成する際のアプリケーション・ソフトウェアの終了に関する注意事項を中心に，アプリケーション・フレームワークを使用した対策方法について説明します．

● クリア・キーを押してアプリケーションを終了

クリア・キーの押下をアプリケーションの終了に割り当てた場合，クリア・キーが押されたときに一つ前の画面に戻る必要があります．たとえば，「待ち受け」→「メイン・メニュー」→「設定画面」というように複数の機能を連続して起動した状態でクリア・キーが押されると，設定画面から一つ前の画面（メイン・メニュー）に戻る必要があります．つまり，クリア・キーによる終了では，キー・イベントを取り扱ったアプリケーション・ソフトウェアのみが終了すればよいということになります．

● 電源キーを押してアプリケーションを終了

電源キーによる終了を実行すると，待ち受けまで一度に戻る必要があります．上記と同じ例では，設定画面の電源キーを押すことによって，待ち受けに戻ります．

このように，同じアプリケーション・ソフトウェアの終了でもキーの種類によりその動きは異なります．キーの種類による終了方法をシステム内で統一しておくことで，統一感のあるユーザ・インターフェースにすることができます．

● 電源キーを長押ししてシステムを終了

電源キーを長押しすると，アプリケーション・ソフトウェアを終了し，待ち受けに戻るだけでなく，システム全体を終了させる必要があります．

● バッテリ残量が少なくなったらシステムを終了

バッテリ残量がある一定量を下回り，低電圧状態になった場合は，アプリケーション・ソフトウェアを終了する必要があります．この場合，アプリケーション・ソフトウェアだけでなく，システム全体を終了させます．

これにより，低電圧状態の不安定な動作や突発的な電源断を防ぎます．突発的に電源が断たれると，最悪，データの消失などを引き起こす可能性があります．

● W-SIM を挿抜したらアプリケーションをすべて終了

通常の音声端末では，通信部分を物理的に取り外すことはできません．一方，W-SIM を使用する音声端末では，W-SIM の挿抜が行われると，その時点で起動しているアプリケーション・ソフトウェアをすべて終了し，待ち受けまで戻ります．メーラや Web ブラウザなどにおいて，実際に通信を行っていた場合はもちろんのこと，文字入力中に W-SIM の挿抜が行われても，待ち受けを除くすべてのアプリケーション・ソフトウェアを終了させます．イベントによるアプリケーション・ソフトウェア終了の違いを**図14**に示します．

電源キーの長押しや低電圧，W-SIM の挿抜による終了は，アプリケーション・フレームワークからの終了要求メッセージとして各アプリケーション・ソフトウェアに通知されます．本開発キットでは，これらのメッセージ送信をミドルウェアが行っています．つまり，アプリケーション・ソフトウェアは送信さ

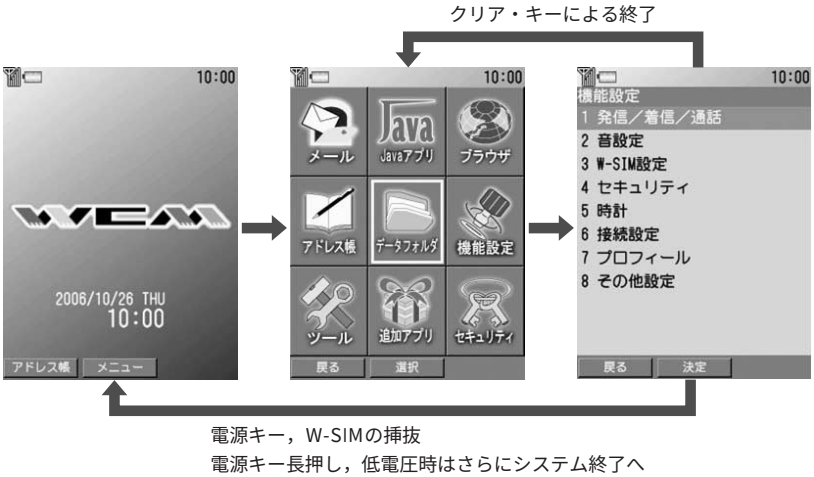

図14
イベントによるアプリケーション・ソフトウェア終了の違い

通常の終了要求イベントのケースである．待ち受け，メイン・メニュー，機能設定と遷移した状態でクリア・キーを押すと，メイン・メニュー（一つ前の画面）に戻る．一方，電源キーを押した場合は待ち受けに戻る

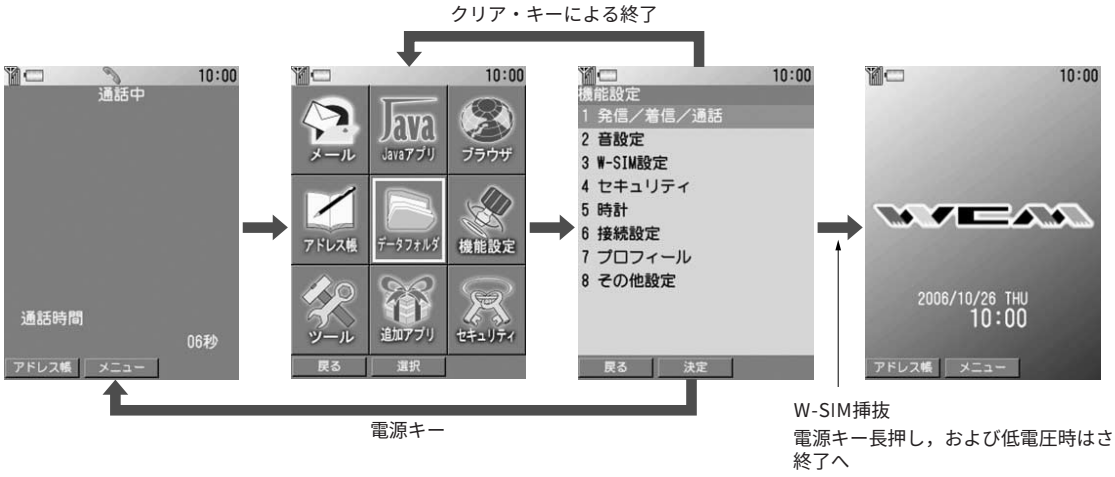

図15　通話状態のアプリケーション・ソフトウェア終了

ダイヤラ動作中（通話中）にメイン・メニュー，機能設定と遷移した状態で電源キーを押すと，ダイヤラの前面に表示されている画面のみが終了し，ダイヤラが表示された時点で終了要求イベントは停止する．一方，電源キー長押し，および低電圧による終了要求イベントでは，ダイヤラも終了し，システム全体が停止する

れたメッセージに従い，正しく終了処理を行えばよいだけです．また，アプリケーション・ソフトウェアの終了前に編集中のデータを保存したい場合は，終了要求メッセージを契機に保存処理を行う必要があります．

一方，電源キーの終了については，電源キーを操作したアプリケーション・ソフトウェアがほかのアプリケーション・ソフトウェアに対して終了要求メッセージを送信する必要があります．これが，前述したアプリケーション・ソフトウェアの一括終了要求となります．この処理を行わない場合は，ほかのアプリケーション・ソフトウェアに終了要求が実行されず，クリア・キー終了と同じ動作となります．

特殊な場面では，終了要求イベントによりアプリケーション・ソフトウェアを終了させたくない場合があります．たとえば，ダイヤラを起動し，通話中にメイン・メニューから機能設定を起動した状態で電源キーを押す場合を考えます．このとき，機能設定とメイン・メニューは終了させますが，ダイヤラを終了させてしまうと通話が切断されてしまいます．また，通話時間なども確認できません．通話状態のアプリケーション・ソフトウェア終了の違いを**図15**に示します．

このような場合を考慮し，ウィンドウごとに終了タイプを指定することが可能となっています．これが，**表4**のウィンドウの終了タイプです．終了タイプは4種類あり，適切なタイプを指定することで，音声端末ならではの動作を実現できます．

● 電話機の状態によって機能を制限

電話機状態とは，電話機に搭載された各種デバイス（ハードウェア）の状態などを意味します．W-SIMを例にあげると，電話機にW-SIMが挿入されているか，未挿入であるかという状態があります．W-SIMが未挿入状態のとき，通信系アプリケーション・ソフトウェア（メーラ，Webブラウザなど）や電話帳

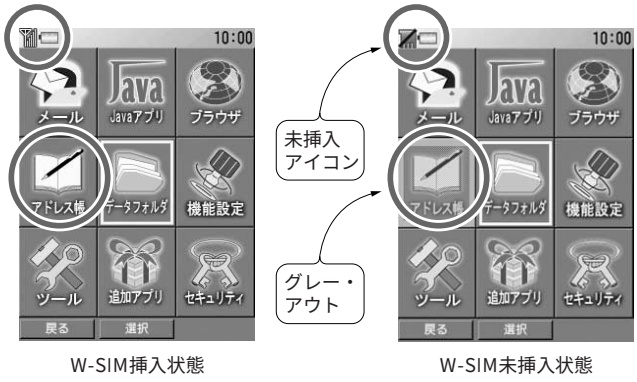

図16　W-SIM挿入状態による表示変更の例

メイン・メニューを起動した際の画面である．W-SIMの挿入状態ではアンテナ・マークが表示され，アドレス帳も選択可能となっている．W-SIMが未挿入の状態では，アンテナ・マーク部分に未挿入マークが表示され，アドレス帳はグレー・アウトにより選択不可となる

を起動するための選択項目をグレー・アウトしたい場合があります．**図16**は，実際に電話機の状態による機能制限を行っているメイン・メニューの例です．

こういった場合は，W-SIM状態の変化を通知するメッセージを受け取り，グレー・アウトするために必要な処理を行います．実際のメッセージのハンドリング処理について説明します．**リスト2**は，実際のコールバック関数のサンプルです．`wapc_msg_get`関数を呼び出してメッセージを取得した後，`msg_id`を参照し，処理を振り分けます．ここに終了要求メッセージ（`MSGAPC_TERMINATE`）が格納されています．さらに，終了理由は，`msg`構造体内の実際のメッセージ内容（`msg`配列）に格納されています．

サンプル・ソースでは，W-SIM挿抜（`TERMINATE_REASON_`

リスト2
メッセージのハンドリング処理

```
static void wapc_msg_func()
{
  int    rc, reason;
  struct msg_t    msg;

  while ( 1 ) {
    rc = wapc_msg_get( &msg );
    if ( rc == MSGAPC_ERROR_NOMSG ) {
      break;
    }

    switch(msg.header.msg_id){
      case(MSGAPC_TERMINATE):

        reason = (int)*msg.msg;

        switch(msg.header.msg_id){
           case(TERMINATE_REASON_WSIM):
             printf("TERMINATE_REASON_WSIM¥n");
             break;
           case(TERMINATE_REASON_POWEROFF):
             printf("TERMINATE_REASON_ POWEROFF ¥n");
             break;
           case(TERMINATE_REASON_LOWBATTERY):
             printf("TERMINATE_REASON_LOWBATTERY¥n");
             break;
           default :
             printf("TERMINATE OTHER REASON¥n");
             break;
        }
        gtk_main_quit();
        break;

      default:
            printf( "msg_func unkown msg(%d)¥n", msg.header.msg_id );
            break;
    }
  }
  return;
}
```


WSIM），電源キー/電源キー長押し（TERMINATE_REASON_POWEROFF），低電圧（TERMINATE_REASON_LOWBATTERY）による終了理由の判定を行っています．終了理由により処理を変更する場合は，case 文の中で処理を行います．そして，gtk_main_quit 関数を用いてアプリケーション・ソフトウェアを終了させます．

通常のアプリケーション・ソフトウェアでは終了理由を考慮する必要がないので，終了要求（MSGAPC_TERMINATE）を受信した時点で終了処理を行います．

たなか・たけひこ，かわい・こうじ
（株）富士通ソフトウェアテクノロジーズ