平成21 年度戦略的基盤技術高度化支援事業

「T-Kernel プラットフォーム ユーザインタフェース系の機能強化ソフトウェア研究開発」

研究開発成果等報告書

平成22年3月

委託者　関東経済産業局
委託先　NECソフト株式会社

目次

第１章　研究開発の概要

１－１　背景と目的

(1)背景

組込 SW では様々な OS(オペレーティングシステム)が使用されているが、組込機器に搭載される OS で、最もシェアが高い OS は RTOS(リアルタイム OS)で、約 35%を占めている。

2009 年時点では、RTOS の中でも iTRON が主で、90%を占めている。iTRON は RTOS の仕様が定められているが、実装方法(ソース・コード)は開発元によって異なる等、差分がある「弱い標準化」の為、2000 年以降、TRON 協会を中心に iTRON の課題を改善した「強い標準化」を目指す T-Kernel が公開されている。

しかし、2009 年時点での T-Kernel の普及率は RTOS が適応されている組込機器の 5%に留まっている。
T-Kernel の普及が進まない要因の一要素に、T-Kernel 上で動作する SW(ソフトウェア) を開発する際、UI(ユーザインターフェース)に関する開発環境が充実してない為、開発元の SW 開発費や開発期間が増大する等の課題がある。

近年登場したiPhoneやAndroid搭載の組込端末では、UIや通信制御に関するアプリケーション(以下アプリと記述)等を効率的に開発する為の様々なミドルウェアがPF(プラットフォーム)として標準で提供されているが、現状のT-Kernelベースの組込SW開発環境においては、前述した充実したSW開発環境が十分に提供されてない。

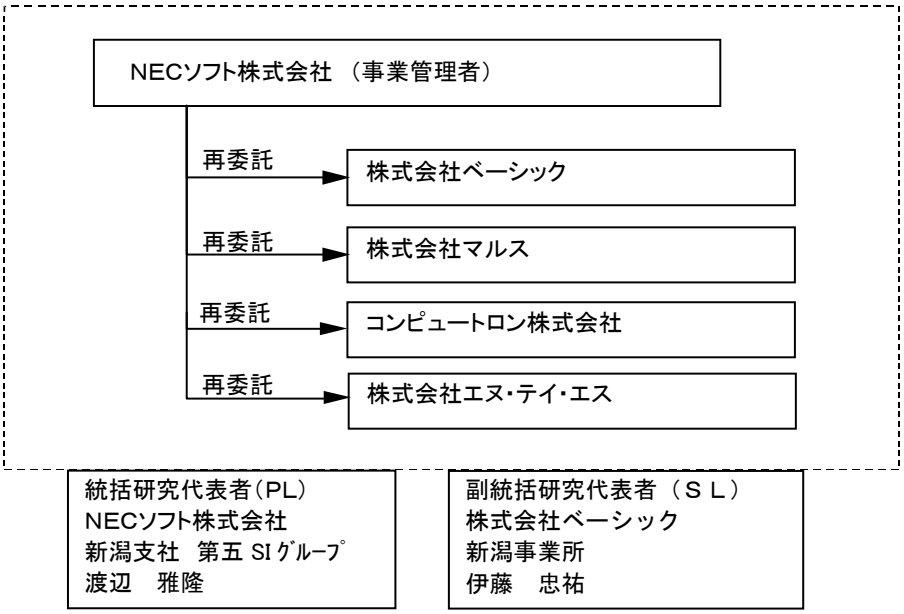
(2)目的

本研究開発の目的は、上述したT-Kernel上のSW開発環境の課題に対して、ユーザインタフェース(UI)系に関連するミドルウェアや関連するプロセス間通信等のミドルウェアを強化する開発を行い、下記効果を得る事を目的とする。

・アプリケーション開発の開発規模の削減、開発期間の短縮
・アプリケーション開発のSW品質の向上
・タッチパネルやキーボタン操作の操作性の向上
・タッチパネルやキーボタン操作の信頼性の向上

１－２ 研究体制

◇研究組織



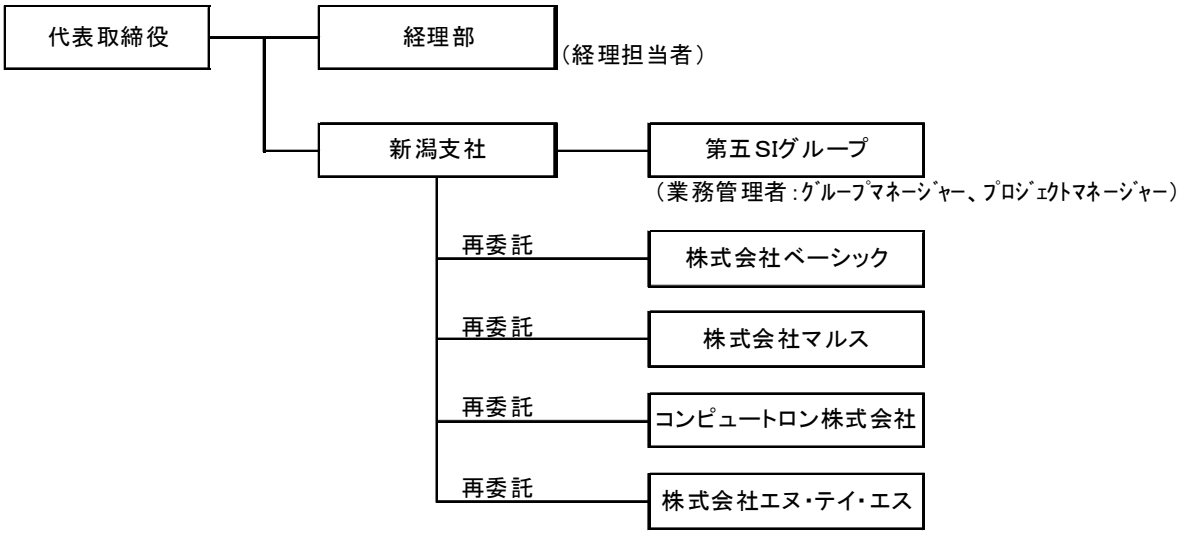
◇研究開発期間

2009 年 11 月中旬～2010 年 3 月上旬

◇管理体制

【責任者】

NECソフト株式会社 新潟支社 第5SIグループ
高野 善之 プロジェクトマネージャー



◇社外の指導・協力者

氏名	所属・役職	備考
星野 雅博	財団法人にいがた産業創造機構 情報戦略ﾌﾟﾛｼﾞｪｸﾄﾏﾈｰｼﾞｬｰ・新事業育成ﾒﾝﾀｰ	アドバイザー

1－3 成果概要

本研究開発では、具体的に下記 5 つのテーマ①②③④⑤を設定して開発を行った。
各テーマと、目的の一覧を以下に示す。

【開発テーマ】
①キーボタンの操作性の向上　（目的:UI 系の機能強化）
②キーボタン操作の信頼性の向上（目的:UI 系の機能強化）
③タッチパネルの操作性の向上　（目的:UI 系の機能強化）
④ミドルウェア開発　（目的:UI 系の機能強化の基盤ミドルの充実＆通信ミドル強化）
⑤アプリケーション開発　（目的:①②③④の動作検証）

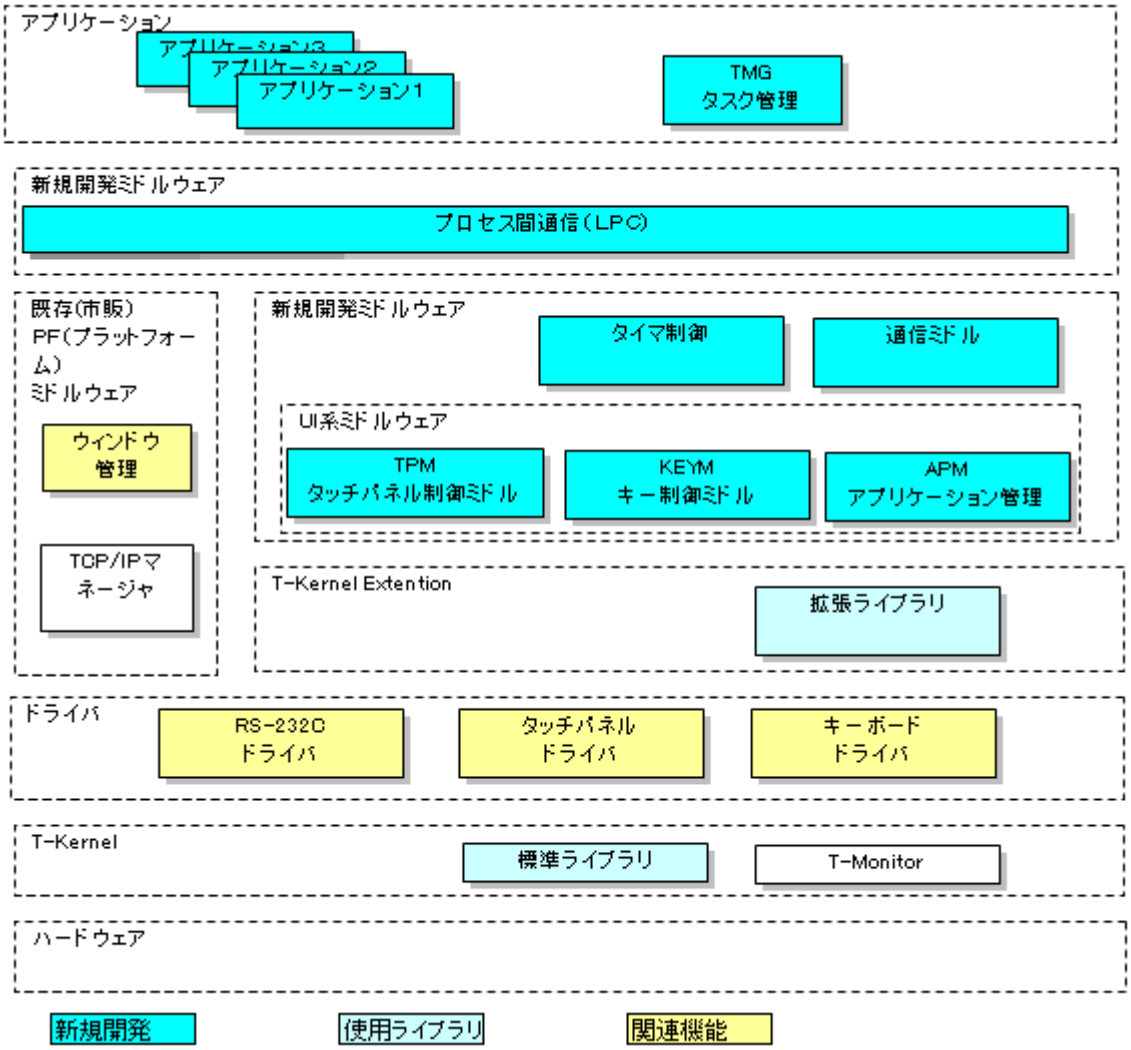
【目的】UI関連ミドルウェアやプロセス間通信ミドルウェアの充実化による、
・アプリケーション開発の開発規模の削減。開発期間の短縮。
・アプリケーション開発のSW開発品質の向上
・タッチパネルやキーボタン操作の操作性の向上
・タッチパネルやキーボタン操作の信頼性の向上

開発テーマの半数がUI改善のため、画面作成等の都合もあり、開発時点(2009 年時点)で提供されているT-Kernel/SE等や市販PFを活用してアプリ開発を行ったが、新規に開発したミドルウェアは、T-Kernel/SE等、公に標準提供されているSWを活用し、特定の市販PFに依存しない開発を行った。

開発テーマ	開発概要	成果 （理由）
①キーボタンの 操作性の向上	・キー操作の充実化 キー長押処理の追加等	・アプリ開発規模の削減。（ミドルによる機能提供） ・操作性の向上 （ミドルによる機能提供） ・操作の均一性の向上 （アプリ実装依存の削減） ・HW 差分のミドル吸収
②キーボタン操作の 信頼性の向上	キー操作時の誤動作の防止機能 ・キー通知ロストの防止 ・キー通知マスク	・信頼性の向上 （キーロスト・マスク機能提供）
③タッチパネル（TP） の操作性の向上	・タッチパネル操作の充実化 （タップ,プレス,フリック,マルチタップ） ・操作確定時の音響効果 ・操作確定時のアイコン表示効果	・アプリ開発規模の削減　（ミドルによる機能提供） ・操作性の向上 （ミドルによる機能提供） ・操作の均一性の向上 （アプリ実装依存の削減） ・HW 差分のミドル吸収 ・イベント通知の効率化 （アプリへの通知削減） ・信頼性の向上 （操作確定の視聴覚認識提供）
④ミドルウェア開発	・プロセス間通信 IF（LPC） ・ネットワーク機器通信 IF（RPC） ・タイマ制御機能	・SW 開発効率の改善 （LPC 適応・煩雑実装の削減） ・SW 開発の品質向上 （LPC 適応・煩雑実装の削減） ・信頼性の向上　（LPC 適応・イベント処理改善） ・ネットワーク通信 SW 開発の効率化 （RPC の適応）
⑤アプリケーション 開発	上記①②③④の機能検証アプリ ・HEMS 関連アプリ ・お薬の時間通知アプリ ・文字入力アプリ、電卓等 ・ビューアアプリ	上記テーマの検証に活用

本研究開発の組込機器の SW 構成、全体アーキテクチャのブロック図を下記に示す。

【全体アーキテクチャ(ブロック図)】



アーキテクチャのブロック図の機能

機能名	略語	機能概要
アプリケーション	アプリ,APL	開発テーマの動作検証を行う画面表示を有する複数のアプリ
タスク管理	TMG	複数のアプリをタッチパネル操作で切替可能とする機能
プロセス間通信	LPC	同期/非同期等の複数機能を1つの I/F で提供する機能
通信ミドル		親子機間通信・親親機間通信の SW 開発を容易化する通信ミドル
タイマ管理		タイマ(時間)の設定/通知を LPC を用いてアプリへ通知する機能
ウィンドウ管理		画面表示や切替等に関する制御機能
タッチパネル制御ミドル	TPM	タッチパネルドライバからデータを取得・加工し、アプリに通知する機能
キー制御ミドル	KEYM	キードライバからデータを取得・加工し、アプリに通知する機能
アプリケーション管理	APM	アプリケーションの起動終了や切替を管理する機能
タッチパネルドライバ		タッチパネルのHWデバイスを制御する機能
キーボードドライバ		キーボタン等のHWデバイスを制御する機能

1－4　当該研究開発の連絡窓口

所属:NECソフト株式会社　新潟支社　第五 SI ｸﾞﾙｰﾌﾟ
氏名:高野善之
電話:025-242-2126
E-mail takano@mxc.nes.nec.co.jp

第2章　本論

2－1 キーボタンの操作性の向上

2-1-1 背景（現状・問題点・目的）

既存 T-Kernel /StandardExtension では、キーボタンを長押操作した場合に、キーイベントについて「押下」「開放」「リピート」が提供されているが、長押と短押の区別がされてない。キーボタンの長押操作をアプリが認識する為には、アプリ側で、短押の「押下」「開放」通知を受けて、タイマ処理を行い、長押しを検知する実装を行う必要があった。その為、キーボタンの入力数が多い場合は、アプリの開発規模増大（費用や期間の増大）に繋がったり、長押しの判定処理がアプリの実装に依存する為、アプリ毎に操作性が異なる等の品質問題に繋がっていた。

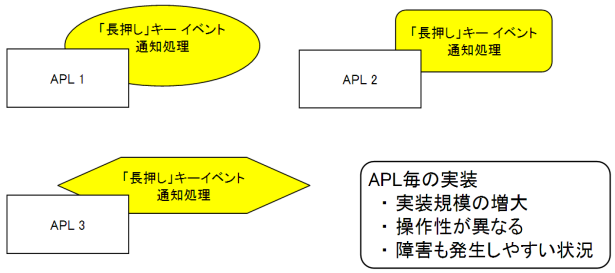
2-1-2 開発目標

キーボタンの長押判定を、アプリ層でなく、ミドル層にて判定し、キー短押/長押しの操作結果を区別してアプリに通知する事で、下記 2 点の改善を行う。

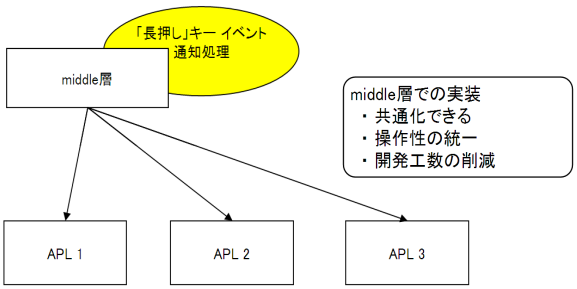
① 組み込み SW の開発効率化（工数・期間等）
② 操作性の統一

改善前後の構成(ブロック図)を以下に示す。

【改善前】



【改善後】



2-1-3 開発手段

上気目標の実現手段として下記3案の差分を確認する為、案1⇒案2⇒案3と段階的に開発を行った。

【案1】キードライバの変更
【案2】T-Kernel/Extension の変更
【案3】キー操作に関する新規ミドルウェアの開発

以下の上記3案の詳細について記載する。

【案1】キードライバの変更
一般的なキードライバは、キーボタンが操作された場合、押下/開放の2種類のデータを、上位層にキーイベントとして通知する。案1は、キードライバにて長押判定を行い、キー長押結果を、押下されたキーと異なる新規キーの押下イベントとしてアプリに通知する。更に、アプリ側にてキーイベントを有効とするタイミングを、押下でなく開放時に変更する。この結果、キーイベンが長押された場合は、元のキーの開放より先に、長押に関するキーが通知される事で、長押を有効として制御する事が可能となる。

【案2】T-Kernel/Extension の変更
T-Kernel/Extension にてキーリピートの判定を行っているのと同様に、キー長押し判定処理を追加し、
キー長押しが有効の場合に新規キーイベントとしてアプリに通知する。

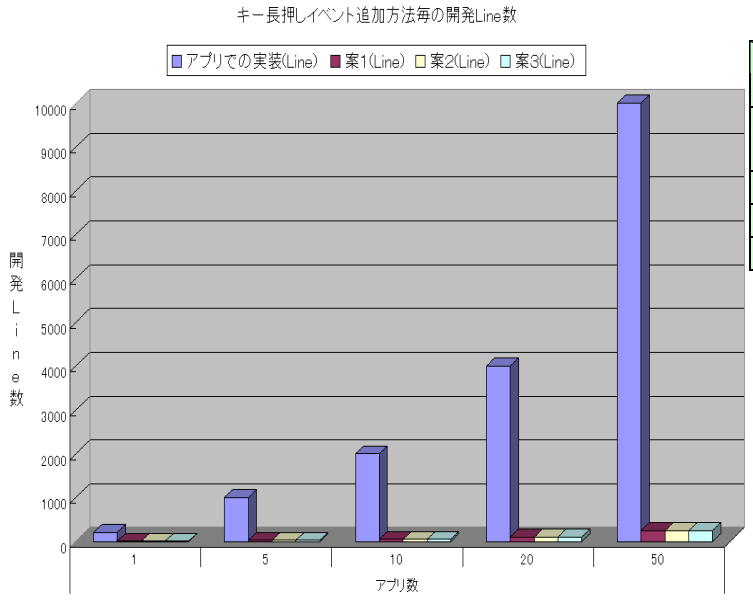
【案3】キー操作に関する新規ミドルウェアの開発
キードライバを制御するミドルウェアとしてキーマネージャ（KEYM）を新設し、KEYM にて、キー操作の長押し判定を行いアプリに通知する。 ※キーボタン操作を一元制御できるので案3がベストな案と考える。

2-1-4 成果/効果

キーイベントの追加をミドル層で実装を行うことにより各アプリの操作性の共通化が行われ、実装についても各アプリで実装を行う際にはそれぞれ 200Line 程度必要であるが、キーイベント受信時に既存のキーイベントと同様にイベント種別を確認するだけで実装できるため 5Line 程度での実装が可能となった。

そのため、ミドル層での実装（案2、案3）を行うことによりアプリ毎に実装をおこなうよりも開発規模について大幅に削減することが可能となった。

また、長短キーイベントの変更についてもキーイベント切り替え IF をコールするのみでキー毎の短押し、長押しの設定の切り替えが可能であり、今後のアプリの拡張において、より柔軟な対応が可能となっている。



	アプリ数				
	1	5	10	20	50
アプリでの実装(Line)	200	1000	2000	4000	10000
案1(Line)	5	25	50	100	250
案2(Line)	5	25	50	100	250
案3(Line)	5	25	50	100	250

	開発方針	開発規模	拡張性	キーロスト防止可否	補足
案1	既存の機能拡張 キードライバ＋ライブラリの改造	中	低	×	組込機器毎に、ドライバ修正が必要。 改造影響範囲も広い。
案2	既存の機能拡張 T-Kernel/Extension の改造	中	低	×	機能拡張時は注意が必要。
案3	新規開発 キー制御ミドルの新設	大	高	○	ドライバ差分や HW 差分を キー制御ミドルで吸収可能となり、 SW 資産の流用が可能となる。

２－２　キーボタン操作の信頼性の向上

2-2-1　背景（現状・問題点・目的）

医療機器や、航空宇宙分野、自動車等、ユーザによる入力操作が、人命に関わる様な処理に繋がる機器においては、ユーザが組込機器の入力 IF(キーボタン等)を操作した結果が、正確/適切に処理できる仕組みが不可欠である。

※キーボタン操作の信頼性の定義について
　画面切替等が起きた場合でもユーザが入力した結果が適切に処理される事。

携帯電話等では待受画面で入力した数字キーが抜けないで電話画面に表示される等、特定の組込機器では、アプリ切替や画面切替においても、ユーザがキーボタン等を操作した結果を、アプリに適切に通知する為の仕組みが入っているが、T-Kernel / StandardExtension 等には、その様な仕組みが十分でない。
キーボタン操作によるアプリ切替時、常駐アプリから非常駐アプリへの画面切替時に、ユーザがキーボタンを操作した結果が非常駐アプリに通知されない等が起こりうる。

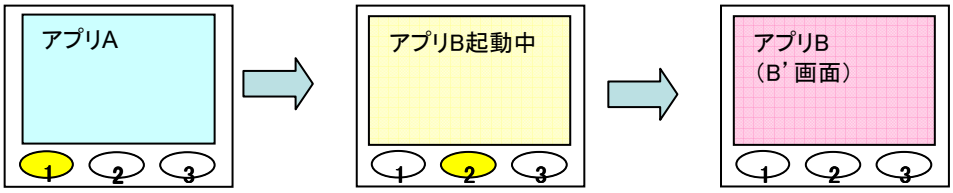
上記の様に、ユーザがキーボタン等を操作した結果が、組込機器上で動作するアプリに正確/適切に通知されない現象を、以降、キーロスト、または単にロストと記述する。

【キーロストの検証結果(T-Kernel/StandardExtension たは、T-Kernel 市販 PF】

キーボタンを連打した場合	キーロスト有無
単一アプリでの画面切替	キーロストしない
常駐アプリ⇒常駐アプリへの切替	キーロストしない
常駐アプリ⇒非常駐アプリへの切替	キーロストする。

※アプリの常駐・非常駐について
　常駐アプリの定義は、組込機器の起動時にアプリのプログラムがメモリ上に常に展開されている状態。
　非常駐アプリの定義は、常時、アプリのプログラムがメモリ上に展開されている訳でなく、必要な場合に、
　メモリ上にプログラムが展開されて動作する。

キーロストが発生しない（期待値）

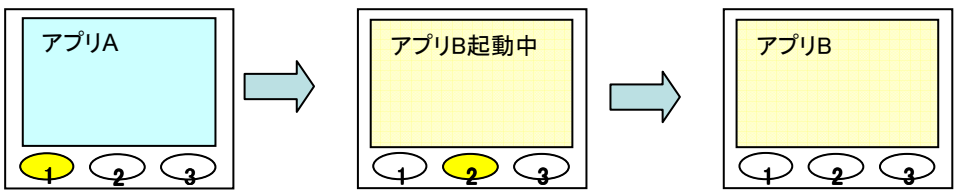


1キー押下
アプリBを起動

2キー押下
アプリB起動中に2キー押下

2キーがアプリBに通知され、
B’画面が表示される。

キーロストが発生する場合



1キー押下
アプリBを起動

2キー押下
アプリB起動中に2キー押下

2キーがアプリBに通知されず、
B’画面が表示されない。

2-2-2 開発目標

上記検証結果に基づき、常駐アプリから非常駐アプリへの切替時に発生するキーロストの対策として、下記2点を開発目標として対応を進めた。

・アプリの状態（常駐や非常駐）に依存せずキーロストが発生しない仕組
・キーイベントの通知をアプリが必要としている場合に通知する仕組

2-2-3 開発手段

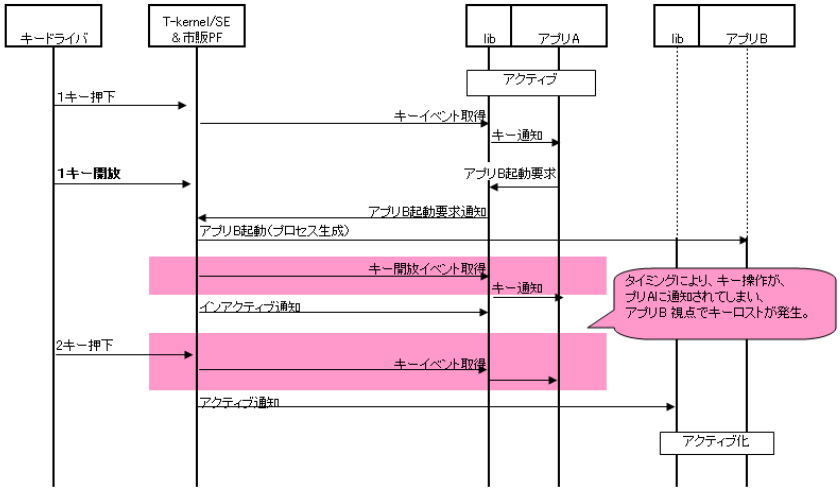
上述の背景や目標、また、既存 SW の改善可否等を考慮した上で、下記2案について、開発を行った。

案1：キー通知可否を APM に確認してアプリに通知する。（既存の SW 資産活用を重視）
案2：キーコールバックの仕組を新設する。（キーロスト対策等、キー通知の最適化を重視）

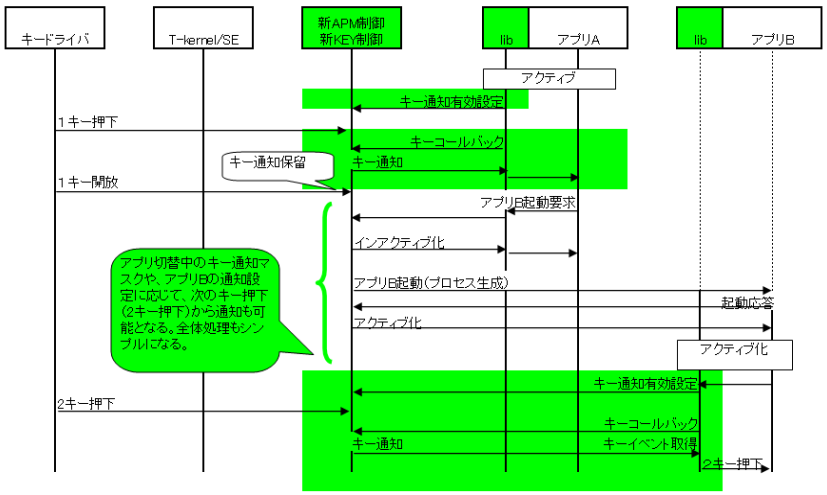
キーロストは、アプリ切替時の制御に関係している為、上記2案に共通する機能として、ミドルウェアにアプリ切替を制御する APM（アプリケーションマネージャー）を新設し、アプリ切替時のキー通知制御を制御する。
案1は、既存の SW 資産の流用率を向上する視点で、APM は既存 PF の補完的な機能として開発し、案2は最適化に必要な機能が入る為、案1と案2の APM の機能は異なる。
また、案2は、ミドルウェアに KEYM(キー制御ミドル)を新設し、キーイベントの通知制御を、細かく制御可能とする。

以下に既存と、案2（最適化）のシーケンス図を示す。

【既存処理】



【案2】キーコールバックの仕組みを新設。（キーイベントをアプリとハンドシェイクしてアプリに通知する）



2-2-4 成果/効果

【成果】
本開発により T-Kernel でもキーロストの防止が可能となり、
組込機器のユーザ IF（キーボタン）操作について、信頼性を向上させることができる。
キーロスト防止の他にも、下記効果を得る事ができる。

		案1		案2	
既存 PF との差分 （開発内容）		差分中	・APM 新規開発 ・LPC 新規開発 ・既存 PF の lib 改造	差分大	・APM の新規開発 ・KEYM の新規開発 ・LPC 開発 ・新規 lib 開発
キー通知の信頼性 （全体視点）		△	中	◎	高
詳細 機能差分	既存 PF の SW 資産活用	○	影響小	×	影響中 （新規 lib 対応等）
	HW 差分吸収	△	改造箇所が複数に分散	○	KEYM で吸収
	キー通知	△	アプリに対してキー押下通知無しのキー開放が通知される場合がある	◎	アプリ設定に応じてキー押下未通知のキー開放をミドルで破棄が可能
	キー通知 （マルチ通知）	×	複数の送信先に同期した通知が困難	○	複数の送信先に同期した通知が可能
	イベント通知 （システム全体視点）	△	既存と新規機能が混在、最適化が困難	◎	最適化可能
	キー通知マスク	△	キーイベント通知先が複数の場合、キーマスクの同期制御が困難	◎	キー通知先が複数でもアクティブアプリを軸にした同期通知制御が可能
	キーロスト	◎	対応可能	◎	対応可能

【補足】
本テーマ(キー通知の信頼性)は、案2を最終目標とし、既存の市販 PF の制限等も考慮した上で、上述しない複数の段階を経た開発を行った。
案1と案2とも、開発難易度が高く、開発規模も大きい為、アーキテクチャと機能設計後、コア部分のプログラム開発と動作検証の確認迄で公募開発期間が満了となったが、他の組込 SW 開発で蓄積した我々のキー操作の信頼性に関するノウハウを、T-Kernel 上でも実現できる事を確認できた。

また、本テーマで開発した APM 機能は、後述のテーマ③、タッチパネルの UI 改善にも活用している。具体的には、APM が起動中のアプリ情報を一括管理することにより、タスクマネージャ（以降 TMG）の制御に活用し、組込端末における、アプリの起動状態・表示状態の参照を実現し、タッチパネル操作の改善にも応用した。

2－3　タッチパネルの操作性の向上

2-3-1 背景（現状・問題点・目的）

近年登場したiPhoneやAndroid搭載の組込端末では、UI（ユーザインターフェース）を向上する為に、タッチパネル対応として、タップ（軽く叩く）、フリック（はじく）等、様々なタッチパネル操作に対応したSW開発環境が標準で提供されている。

しかし、現状のT-Kernelベースの組込SW開発環境においては、前記UIに関する充実したSW開発環境が提供されておらず、更に下記問題もある為、T-KernelベースでのUI向上施策のSW開発を行った。

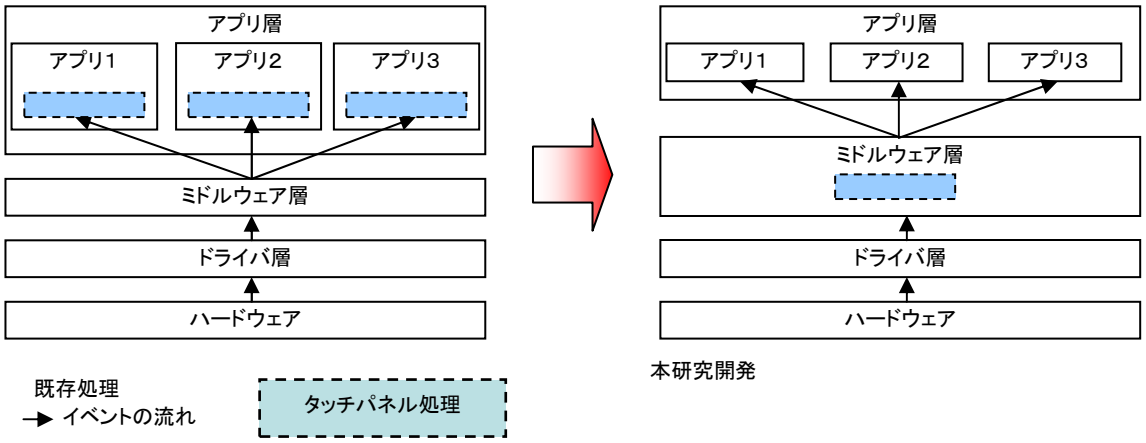
既存の問題点　（T-KernelベースでのタッチパネルSW開発）

・タッチパネル操作を含むアプリ開発にて開発期間や開発工数が増大する
・タッチパネル処理がアプリの実装や処理負荷に依存し、操作結果が均一にならず信頼性に問題がある
・タッチパネル操作が確定したことをユーザが認識しづらい

2-3-2 開発目標

T-Kernelプラットフォームでのタッチパネルに対する課題を解決するため、新規のタッチパネル処理用ミドルウェアを開発する。

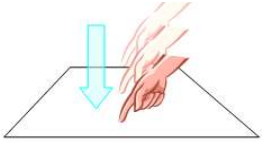
アプリで実装していたタッチパネル処理をミドルウェアで実装する事で、タッチパネル操作を含むアプリ開発において開発負担軽減、操作の統一化、不要イベントの破棄や非同期でのアプリへの通信による信頼性の向上を目標とする。



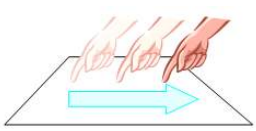
タッチパネルを操作した時に、既存の市販PF等で提供されている、下記3種類のイベントに＋αとして、新たに下記4種類のイベント、タップ（軽く叩く）、プレス（押し続ける）、フリック（はじく）、マルチタッチを、新設する。

■既存の市販PF等で提供されているタッチパネルのイベント

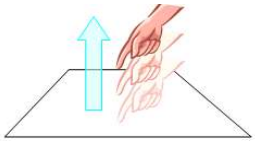
◇タッチ
　タッチパネルを指で触る操作



◇ドラッグ
　タッチしたまま、指を移動する操作

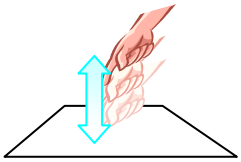


◇リリース
　タッチパネルから指を離す操作

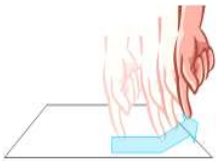


■新規に追加する通知イベント
　タッチパネルの操作について、下記の様に、特定のイベントを通知する操作の総称を、
以降の記述でジェスチャとして表記する。

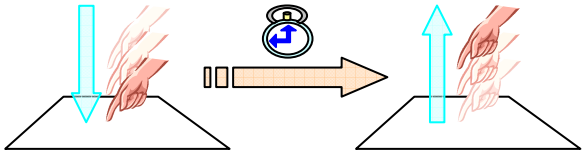
◇タップ
　タッチパネルをタッチして
　リリースする操作



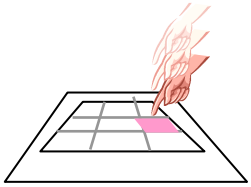
◇フリック
　タッチパネルをタッチし、一定時間内に
　一定方向にドラッグ後、リリースする操作



◇プレス/リピート
　タッチパネルの一点を一定時間以上押し続ける操作



◇マルチタップ
画面領域を、2次元行列で管理し有効領域内
のタップを、他のタップと区別して通知知する



2-3-3　開発手段

2-3-3-1 実現手段

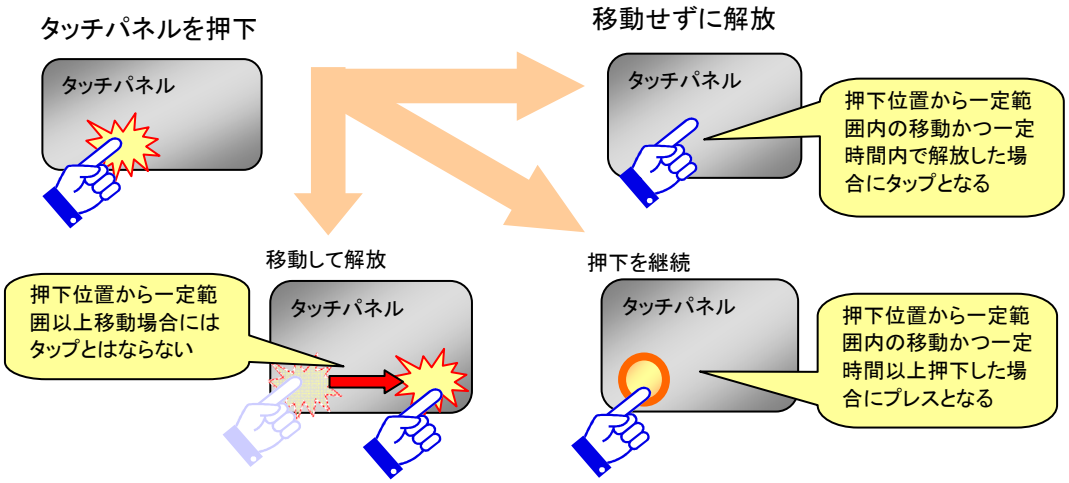
◇概要

　下記アーキテクチャのミドル層に、TPM(タッチパネル制御ミドル)を新設し、タッチパネルの処理は、ドライバから通知されるデータをアプリに直接するのではなく、TPMでタッチパネル操作の結果を特定のデータに加工し、加工後のデータをアプリに通知する。動作に必要な各種の設定値は新規I/Fを作成し動的に変更可能とする。
また、イベントの通知開始／停止も新規I/Fを設ける事により、アプリ側で制御可能とする。

新規に追加したタッチパネル操作の実現手段を以下に示す。

◇タップ

　移動距離が一定範囲内で、一定時間内でタッチした場合にタップとする
　移動距離が一定範囲内で、一定時間以上でタッチした場合をプレスとする
　尚、プレス後もタッチし続けた場合をリピート間隔毎にタップを定期的に通知する

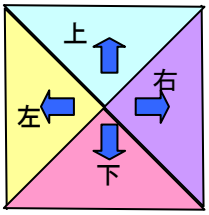


◇フリック

移動距離が一定範囲以上で、一定時間内でタッチした場合にフリックとする
境界線上での判定は左右フリックを優先する

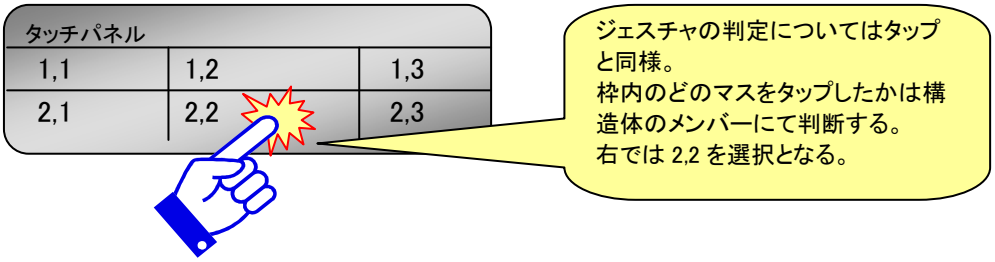
指の移動方向とフリックタイプの関係は以下の通り。

移動角度	フリックタイプ
0～45°	右フリック
46～134°	上フリック
135～225°	左フリック
226～314°	下フリック
315～360°	右フリック

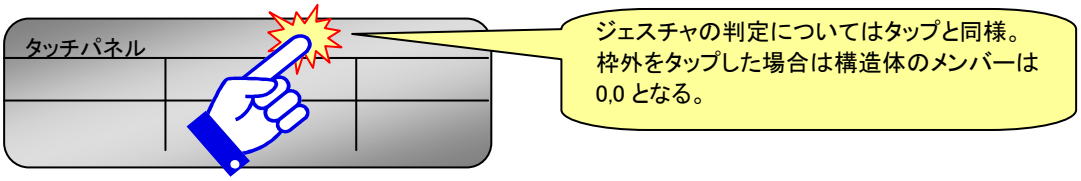


◇マルチタッチ

タッチパネルの画面領域を、2次元行列で管理し、有効領域内のタップを、通常のタップと区別してアプリに通知する。枠内のタップはマルチタッチのイベントに返還して通知し、アプリは、1つのイベント通知で、枠内の何処が押されたかを判断制御できる様にする。



枠外の場合は、枠内と異なるイベントを通知する。



◇ジェスチャ確定とサウンド(鳴動)制御

ユーザ操作に対して、ジェスチャが確定した際にブザー音等を鳴動させ、ユーザがジェスチャ操作が確定した事を認識可能とする。ブザー音を使用する場合は、音階ではなく、鳴動時間と鳴動回数にて各ジェスチャを区別可能とする。

◇ジェスチャ確定とアイコン表示

ポインタアイコンを用いて、タッチパネルを押下した際に触れた位置にアイコンを表示し、解放した際にアイコンを消去する。但し、解放後すぐに消去するとポインタアイコンが一瞬しか表示されず、ユーザが行った操作を認識しにくいため、指を離した1秒後まで表示させておくこととする。
また、ジェスチャ確定時には表示しているアイコン種別を変更する。

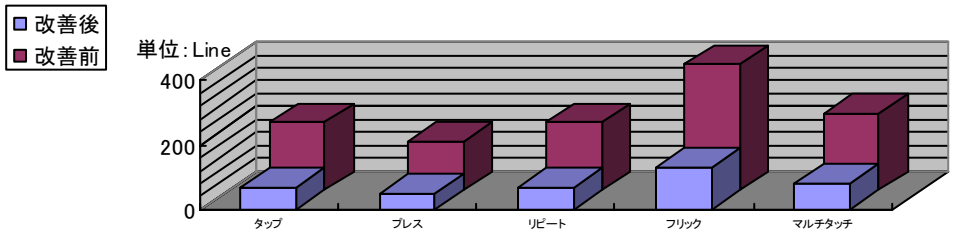
2-3-4 成果/効果

タッチパネル処理をミドルウェアにて実装することにより以下の成果を得ることができた。
・新規イベント提供によるアプリ開発規模の削減（HWのメモリの有効利用の効果もある）
・操作性の向上 （ミドルでタッチパネル操作を一括判定する事よる操作の均一性の確保）
・イベント通知の効率化（ 不要イベントをアプリに通知しない事でシステム全体の効率化）
・信頼性の向上（ジェスチャ確定時の視聴覚によるユーザ認識の向上）

		効果	改善後	従来
全体		◎	・アプリ層の実装削減 ・操作の向上 （操作の均一性の向上） ・イベント通知の効率化 （不要イベントの通知削減） ・信頼性の向上 （ジェスチャ確定の視聴覚認識可）	・アプリ層の実装負荷増大 ・操作が均一になり難い（アプリ実装依存） ・イベント処理増大（バッテリ駆動では不利）
ジェスチャ	タップ	○	・アプリ層の実装削減 ・イベント通知の効率化	・実装負担 ・操作性の押下/開放を基に、アプリがタップ判定（座標や時間）⇒実装負担
	プレス リピート	◎	・アプリ層の実装削減 ・操作性向上 （アプリ負荷依存無） ・イベント通知の効率化	・操作性低 （タイマ判定等ではアプリ負荷の影響を受ける。またはドライバデータにタイムスタンプでアプリ側で判断等、アプリ実装の煩雑要因（品質影響）
	フリック	◎	・アプリ層の実装削減 ・操作性向上 （アプリ負荷依存無） ・他ジェスチャと区別した通知が可	アプリ側での座標トレースやタイマ処理等、アプリ実装の煩雑要因(品質に影響)
	マルチ タッチ	◎	・アプリ層の実装削減 ・操作性向上 （アプリ負荷依存無） ・他ジェスチャと区別が容易。	アプリ側での座標トレースやタイマ処理等、アプリ実装の煩雑要因になっていた
ジェスチャ 以外の効果	効果音	○	・信頼性/操作性の向上 （ジェスチャ確定の聴覚確認が可）	アプリで対応する場合、ジェスチャ確定と鳴動にズレが生じる事がある
	アイコ ン表示	○	・操作性向上 （ジェスチャ確定の視覚確認が可）	アプリで対応する場合、ジェスチャ確定とアイコン表示にズレが生じる事がある
	ロスト 防止	○	・キー操作の信頼性向上で開発した機能との組合わせにより、タッチパネル操作結果についても、操作結果のロスト防止が可能 （信頼性が向上）	イベント通知ロストの防止やイベント通知マスク機能が無い（信頼性低）

※新規イベント提供によるアプリ開発規模の削減
上記表のジェスチャ機能をアプリで実装する場合、アプリ側で、タッチパネルの押下、移動、開放のイベント監視、タイマ監視を行う必要があり、それぞれのアプリで100～300 Line程度の開発規模となる。タッチパネル処理をミドルウェア層にて実装する事で、アプリ側の開発規模を削減できるため、複数アプリを開発する場合は開発工数の削減が可能となる。また、HWのメモリの有効利用等の効果を得る事ができる。

開発規模概算 （単位：Line）

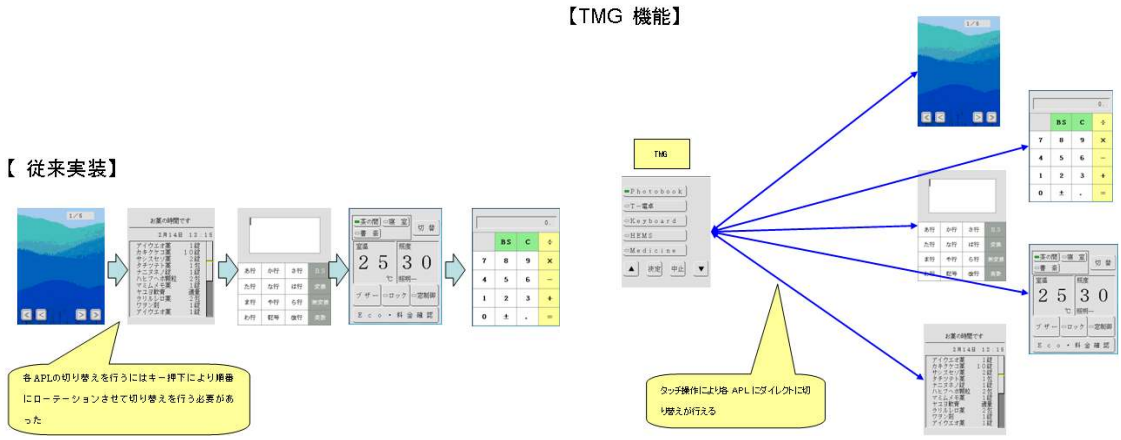


2-3-5 タッチパネルの操作性向上（その2）

タッチパネル操作によるアプリケーション切替機能(TMG)の開発
複数のアプリケーションを搭載する場合は、画面切替（アプリケーション切替）が容易に操作できることも、商品性として重要な課題である為、前述のテーマ②③の開発結果を基に、タッチパネル操作によるアプリケーション切替機能の向上に関する開発を行った。

開発目標/手段

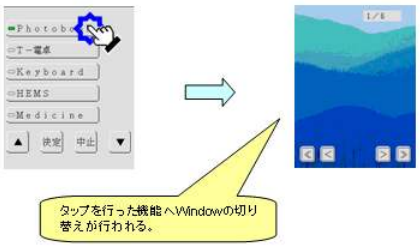
テーマ②で開発を行った APM(アプリケーションマネージャ)と、テーマ③のタッチパネルミドル（TPM）を基盤として、更に、新規にタスクマネージャ(以降 TMG)機能を作成し、ユーザがタッチパネル操作をすることで、任意のアプリに切替可能とする機能を開発した。



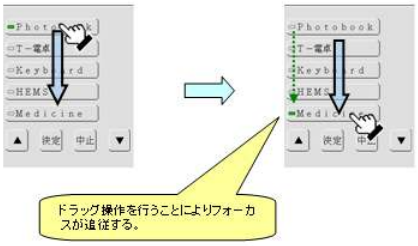
成果/効果

アプリ切替の操作性が向上、画面が不可視状態のアプリに対しても、アプリ切替えが可能となった。
以下にタッチパネル操作のアプリ切替操作向上の具体例を示す。

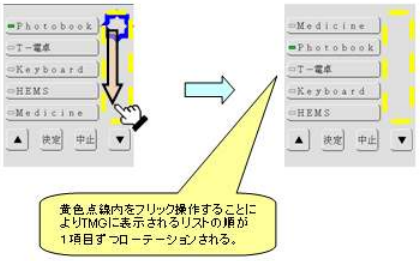
・タップ



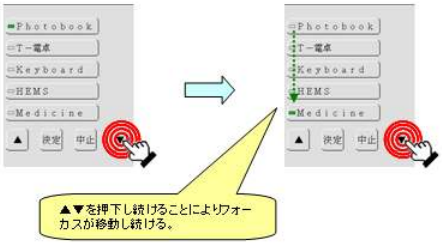
・ドラッグ



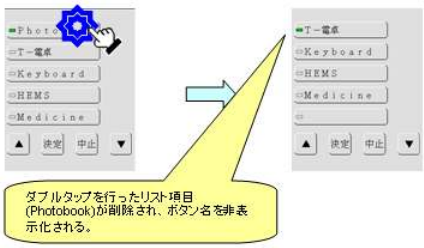
・フリック



・リピート



・ダブルタップ



2－4 ミドルウェア開発

2-4-1 背景(現状・問題点・目的)

T-Kernel ベースにおける既存 SW 開発は、T-Kernel/StandardExtension(標準／拡張ライブラリ)や、複数の市販 PF から提供されている様々な機能を使用した SW 開発が可能である。

様々な機能が提供されている事は、開発手法の多様化に繋がり、プロセス間通信の視点で見た場合、異なる開発環境や異なる開発者が開発した SW に対してプロセス間通信を行う場合に下記課題が存在する。

◇既存の問題点(課題)

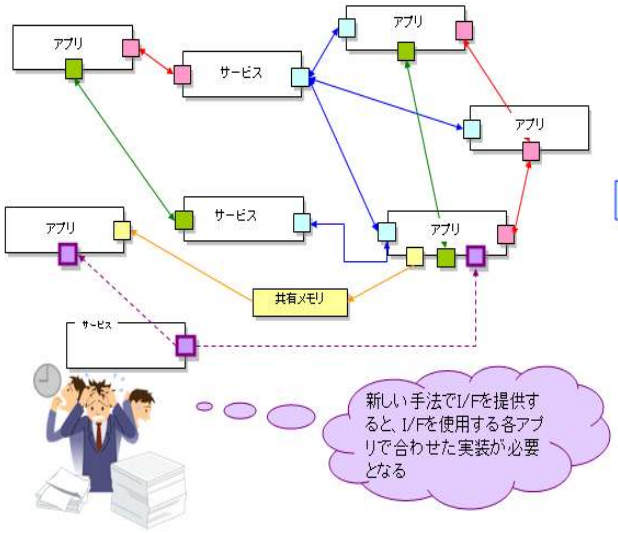
・プロセス間通信の I/F(インターフェース)仕様が多様化しており開発効率が悪い
・プロセス間通信の処理順序に差分があり信頼性が低い
・異なる通信機器との通信に関連する開発効率が悪い

以下に上記課題について補足説明する。

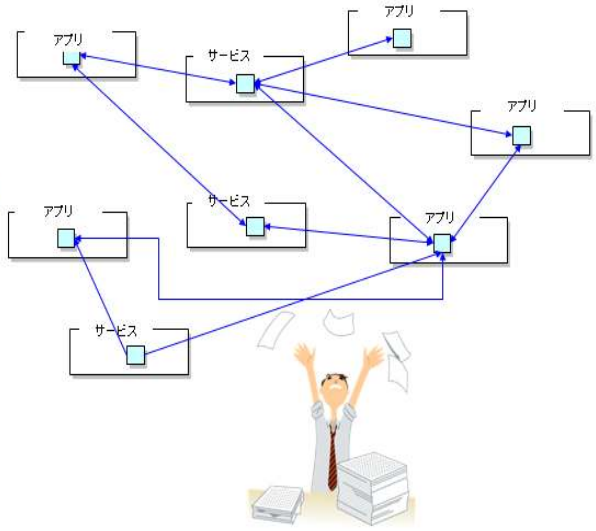
◇プロセス間通信の I/F(インターフェース)仕様が多様化しており開発効率が悪い

異なる開発環境、または異なる手法で開発したプログラムに対してプログラム間通信の開発を行う場合、通信相手との I/F 仕様調整や、新たな開発(追加/改造)が発生する事が多い。

【改善前】



【改善後】

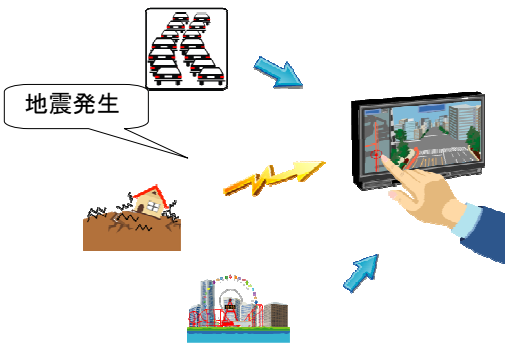


◇処理順序により信頼性があいまい

組込機器に搭載されるアプリは下記の様な、様々な事象を処理している。

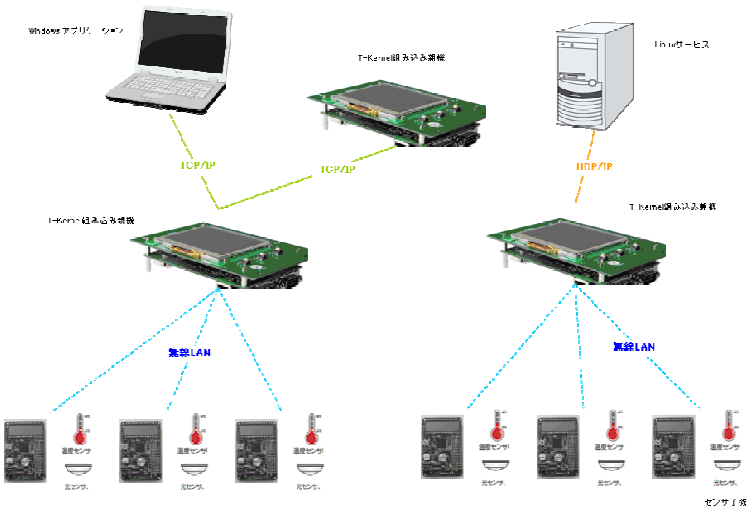
・利用者のキーの操作
・利用者のタッチパネルの操作
・利用者への通知(定期的な情報や緊急なアラームなど)

既存の開発環境(市販PF等)上で動作するアプリは、発生した事象に対し、発生順に処理を行っている為、緊急なアラームなど重要な処理が、他の処理の終了を待って処理される場合が起こりえる。



◇ネットワーク上の機器との通信に関連する開発効率について

異なる通信機器を複数制御するアプリケーションを開発する場合、それぞれの機器を制御するライブラリに合わせた開発が必要となり実装が煩雑となっている。



◇開発目標

・様々な通信手法を一本化したプロセス間通信 I/F を提供する。

⇒システム上の様々な事象通知をイベントドリブン型のプログラミングスタイルで一本化して実装可能とする新規のプロセス間通信 I/F(ミドルウェア)を提供する。

・プロセス間通信の受信側処理に優先通知の仕組みを提供する。

⇒複数の事象が同時に発生した際にプロセス間通信の受信側(アプリ等)が、優先処理したい事象を、前述の新規プロセス間通信の I/F に事前に簡易設定する事で、受信側で煩雑な実装をせずとも優先処理できる仕組みを提供する。

・異なる通信機器を通信プロトコル等を意識せずに制御できる I/F を提供する。

⇒異なる通信機器等を抽象化し、統一した手法で制御できる I/F を提供する事で、対象機器の持つ機能の実行や状態監視などを簡易に制御可能とする仕組みを提供する。

2-4-2 開発手段

通信ミドルウェアとして、1つのコンピュータ上で動作するプロセス同士の通信を対象とした「プロセス間通信」機能と、ネットワーク上に存在する他のコンピュータ上で動作するプログラムとの通信を対象とした「ネットワーク機器通信」機能を提供する。

プロセス間通信とネットワーク上の機器との通信における開発概要を以下に示す。

プロセス間通信

No	開発概要
1	イベントドリブン型でのイベント通知機能の提供
2	処理の同期を伴うイベント通知機能の提供
3	イベント処理の優先処理の提供
4	特定の事象の発生待ち機能の提供
5	イベントループ機能の提供
6	送受信可能なデータサイズの制限を排除
7	プログラムインターフェス間でのコピーを抑止機能の提供
8	イベント通知を利用したタイマ管理機能の提供
9	既存機能からの機能劣化のないことの保障
10	既存機能からの性能差を最小限に抑える(＋30%以内を目標)

ネットワーク上の機器との通信

No	開発概要
1	接続機器の抽象化機能の提供
2	温度、照度センサ子機のファームウェアを提供
3	センサ子機間通信機能の提供
4	親機間通信機能の提供
5	他プラットフォームとの連携の提供

2-4-2-1 プロセス間通信

上位プログラム層（アプリ、サービスなど）に対して、プログラム間で連携処理を行うための機能を開発した。「Local Procedure Call」とは、1つのコンピュータ上で動作するプロセス(タスク含む)が、他のプロセスの機能を呼び出すための手法である。

LPC の主な機能は、イベントと定義する事象通知(以降、イベントと記載する)をトリガーとして、他プロセスへの通知や機能呼び出しを行う。

前述したプロセス間通信の表に記載した、10 通りの機能を考慮し、既存の T-Kernel/StandardExtension 等が提供している I/F(プログラムインターフェース)より使いやすい I/F 機能の開発を行った。

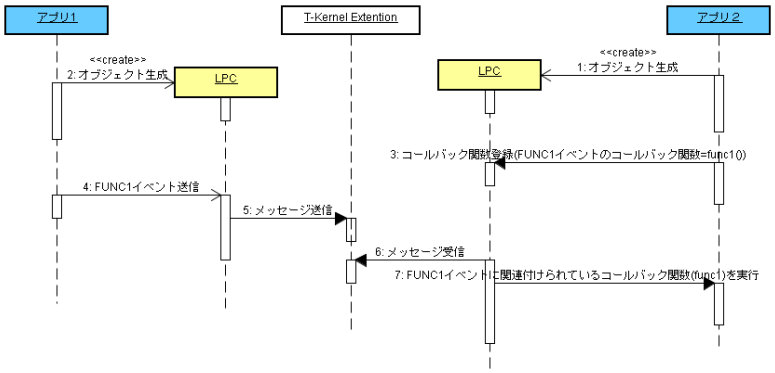
以降で、各機能における実現方法の詳細を以下に示す。

・イベントドリブン型でのイベント通知機能の提供

システムで定義する事象に対するイベントを他のプログラムに通知する機能を提供する。

イベント通知を行うプログラムには、イベントを送信（通知）する機能を提供する。イベント送信を受けるプログラムには、イベントに対するコールバック関数を関連付けることで、イベントを受信した際にコールバック関数が自動的に呼び出される仕組みを提供する。また、送信するイベントには任意のデータを付与することを可能とした。

処理野シーケンスを以下に示す。

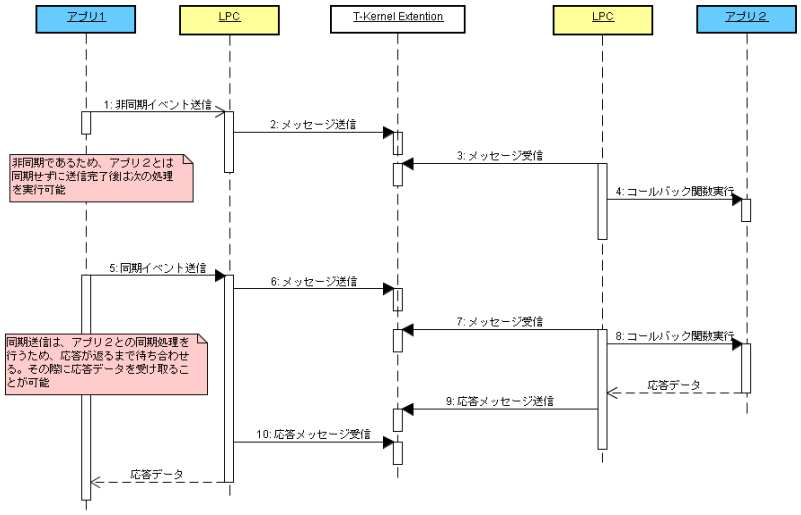


・処理の同期を伴うイベント通知機能の提供

上記のイベント送信の機能に同期処理を持たせることを可能とする。イベント送信先のプログラムのコールバック関数の終了を待ち合わせる仕組みを提供することによりプログラム間で同期処理を行う仕組みを提供する。

また、コールバック関数から出力された応答データをイベント送信元のプログラムへ返却する機能を開発した。

処理のシーケンスを以下に示す。



・イベント処理の優先処理の提供

イベント通知に優先度を付与して優先処理を行う。イベントに付与する優先度については、通知するアプリ側が任意な優先度を指定することを可能とする。

イベントを受信するアプリ側では、同時に複数のイベントが発生している場合は、優先度高いイベントのコールバック関数から順番に処理が実行されるため、処理順序を意識することなく処理の実装を行うことを可能とした。

・特定の事象の発生待ち機能の提供

同期イベントの通知以外の処理の同期手段として、他のプログラムからのイベント通知を待ち合わせる機能を提供する。これにより、他の処理が終了することや異常からの復帰を待つような同期処理の実装が可能となる。

この機能は、既存の標準／拡張ライブラリには存在しない。

・イベントループ機能の提供

アプリで行っているキー操作やタッチパネル操作、ボタン押下などの GUI イベントに対する既存のイベントループへの LPC イベントを組み込んだイベントループ、GUI を持たないサービスプログラムに対する LPC イベントを対象としてイベントループの機能を開発した。

この機能は、既存の標準／拡張ライブラリには存在しない。LPC を使用しない場合は、アプリ側でイベントに対するループ処理を実装する必要がある。

GUI を持つプログラムのイベントループについては、既存ウィンドウ管理ミドルウェアの提供する関数をラップする形で実装する事も可能である。

・送受信可能なデータサイズの制限を排除

既存の標準／拡張ライブラリでもイベントを通知する仕組みは存在しているが、イベントに付与されるデータ、応答するデータのサイズに制限がある。

LPC ではデータのサイズに制限は設けずにシステムリソースが許すサイズまで有効とする。

この機能は、既存の標準／拡張ライブラリには存在しない。制限の範囲外データサイズを通知する場合は、共有メモリやイベントフラグなど他の手法との組み合わせた実装を送信側と受信側で行った。

・プログラムインターフェス間でのコピーを抑止機能の提供

性能向上のためのアプリとミドルウェア間の関数インターフェースでのデータコピーを抑止する機能を開発した。

この機能は、既存の標準／拡張ライブラリには存在しない。コピー抑止機能では、送信を行う前にアプリ側でメモリの事前割り当て API を実行し、その領域を LPC が直接参照や出力を行うことで、不要なデータコピーと領域獲得を減らしている。これにより、送受信するサイズに比例して既存機能と比べても改善効果が高いと考えられる。

・イベント通知を利用したタイマ管理機能の提供

タイマ管理については、標準／拡張ライブラリにて提供されているが、他の事象通知と同様にプロセス間通信のイベントドリブン型の通知で実装することを可能するため、LPC を活用したタイマ管理機能の開発を行った。

・既存機能からの機能劣化のないことの保障

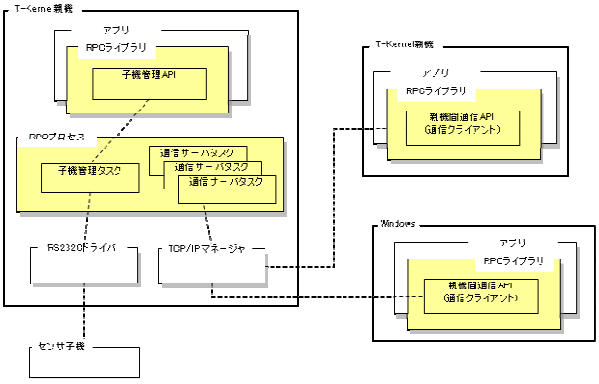
既存の標準／拡張ライブラリからの機能劣化のないことを保障するために以下を考慮して機能を実現した。

- アプリ側でのマルチタスク動作を意識せずに使用可能
- 既存の通信機能に対する大きな性能劣化が見られないこと（＋30%以内を目標）

2-4-2-2 ネットワーク機器通信

上位アプリに対して、ネットワーク接続された機器を抽象化して、機器の持つ機能の実行や状態監視の機能である RPC(Remote Procedure Call)の開発を行った。

「Remote Procedure Call」とは、ネットワーク上に存在する他のコンピュータ上で動作するプログラムの機能を呼び出すための手法である。前述した LPC(Local Procedure Call)が1つのコンピュータ内のプログラムを対象としていることに対し、RPC では、ネットワーク接続されたコンピュータのプログラムが対象となる。



以降に、ネットワーク機器通信の各機能における実現方法の詳細を示す。

・温度、照度センサ子機のファームウェア開発

温度、照度センサを持つ子機で動作するファームウェアの開発を行った。

親機と子機にはそれぞれに無線 LAN モジュールが接続され、無線モジュール間で無線 LAN プロトコルにてネットワーク接続される。今回の研究では、親機側からの温度／照度の取得要求に対して、応答するファームウェアを開発する。本研究開発では、下記機能の対応の為、親機から子機の監視処理のみで開発を行った。

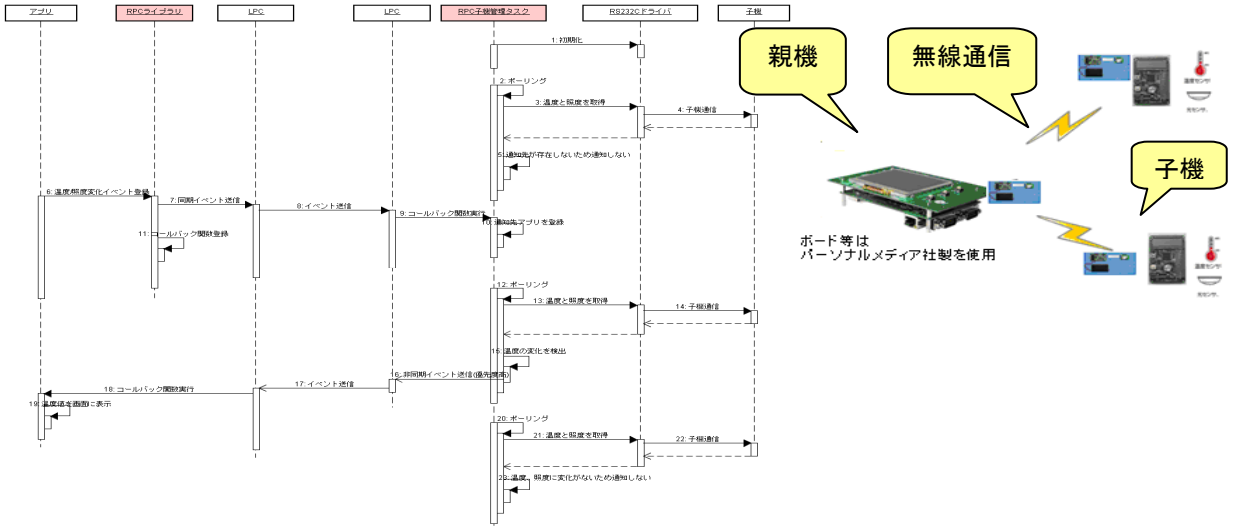
センサ子機ファームウェアの機能概要を以下に示す。

- 温度センサデータ、照度センサデータの取得
- ブザー鳴動
- ディスプレイ表示

・センサ子機間通信機能の提供

親機に接続される無線モジュールのシリアル通信インターフェースを利用して、センサ子機との通信し、アプリに対しては、センサ子機の機能と状態監視として以下機能の開発を行った。

- 温度センサ値、照度センサ値の変化通知
- ブザー鳴動
- 子機のネットワークからの切断通知、再接続通知



・親機間通信機能の提供

親機で動作するアプリやサービスなどのプログラムの持つ機能を他の親機から利用するための機能を提供する。他の親機に対して、LPC のイベント通知を実行する機能を提供することで、親機間での状態監視や事象通知を行うシステムを容易に構築可能とした。

・他プラットフォームとの連携の提供

親機間通信の通信プロトコルを利用した他プラットフォーム向け（今回は、Linux と WindowsC#アプリ））のミドルウェアを提供することで、T-Kernel プラットフォームと他プラットフォームで連携したシステム開発を行いやすくする。

また、通信サーバタスクの処理性能の向上が求められるため、通信サーバタスクの並列処理に取り組んだ。クライアント側からの接続を検出した際に、空いている通信サーバタスクに処理を割り当てた。

2-4-3 成果/効果

通信ミドルウェア開発の成果を以下の観点で検証した。

No	検証内容
1	プロセス間通信に関連する開発効率が改善されること
2	処理順序に関する処理が改善され信頼性が向上すること
3	プロセス間通信の仕組みを活かしてタイマ管理が行えること
4	プロセス間通信の性能について既存機能との性能差が最小限に抑えられていること
5	ネットワーク上の機器との通信に関連する開発効率が改善されること

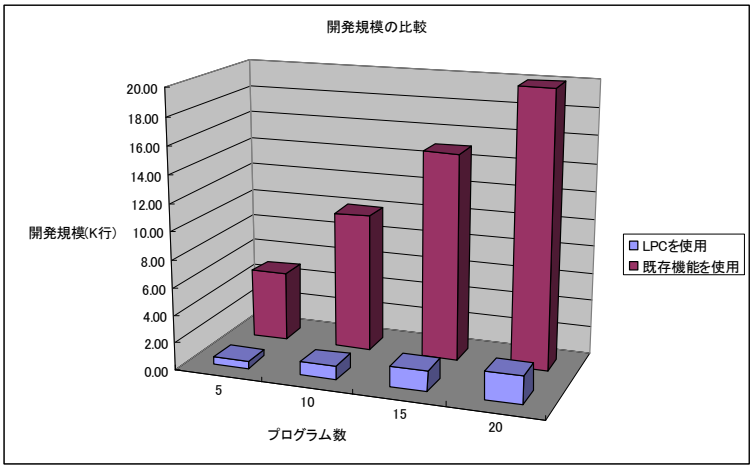
検証の結果から、導入に伴う開発効率の改善を見込むことが可能であり、プロセス間通信においては既存機能と比較しての機能面での向上だけでなく、処理性能についても最小限の差に抑えられている。
そのため、通信ミドルウェアの導入は、SW 開発の効率化へ大きな貢献ができる。

検証結果の詳細を以下に示す。

2-4-3-1 プロセス間通信に関連する開発効率の改善

上位プログラム側でLPCと同様の機能を持たせる場合、プログラム当たり約1.0K行※の実装が必要と考える。
※LPCの開発規模が有効行(空白、コメント行以外)で約3.0K行である。
以下の点を考慮して、LPCの1/3の規模で見積もるが実際はもう少し多いと考える。
- 上位プログラムの内容によっては必要の無い機能が存在する
- LPCを使用しないシステムにおいてもLPCに近い共通化を図ることを想定

LPCを使用した場合は、LPCの提供する関数の呼び出し部分の実装が必要となり、その規模は、プログラム当たり約100行と考えている。LPCを使用した場合と既存機能を使用する場合と比べ、約1/10の規模で開発可能であり、更に上位アプリの数に比例して効率化される規模は大きくなる。



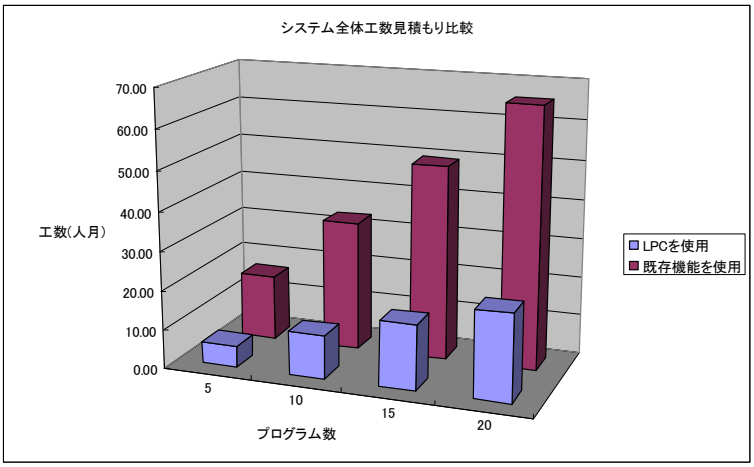
またLPCを使用する場合は、プログラムの開発規模だけでなく、以下の工程での工数削減効果と品質の向上効果も見込まれる。

工程	既存機能の開発	LPC 使用効果
知識習得	複数の手法の理解が必要である	LPC のみの理解となる。既存機能より容易に理解可能である
I/F 設計	複数のプログラム間と情報を共有するための手法や受け渡しする内容を調整する必要がある	LPC の使用法に基づいた内容で I/F 仕様を決定することが可能で調整しやすい
詳細設計／コーディング	既存機能を利用して全て実装する必要がある。また GUI を利用した場合に、通信対象によりイベントループの見直しが必要となる可能性がある	LPC の機能を用いて、上位プログラムに依存した部分のみを実装することが可能である
テスト	比較的難易度の高い既存機能の利用によるバグ対応が必要で、実装により設計への後戻りの可能性がある	LPC で実装されている部分については、品質が確保されるため、上位プログラムに依存した部分のテストとなる

各開発工程で必要工数を以下で見積もった場合、1プログラム当たり約2人月の工数短縮が可能の見込である。

開発工程	既存機能を使用	LPC を使用
知識習得	0.30 人月	0.10 人月
I/F 設計	0.75 人月	0.25 人月
詳細設計	0.75 人月	0.25 人月
コーディング	0.75 人月	0.25 人月
テスト(問題対応)	0.75 人月	0.25 人月
合計	3.30 人月	1.10 人月

更に、効率化の規模はアプリ数に比例して大きくなる為、複数のアプリを開発する場合や、アプリの開発規模が大きい場合は、組込機器全体のSW開発工数削減に対する効果も大きい。



また、実際の開発ではプロセス間通信で不具合が発生した場合、問題調査や対応に難航する場合が多い。
LPCを使用した場合、通信の状況をLPCの通信ログにて確認するなどの方法で解析することが可能となり、問題の解決までに要する時間の短縮だけでなく、解析方法についても一つの手法で対応できることによる、組込機器のSW開発全体の品質向上にも繋がる。

2-4-3-2 処理順序の改善について

優先度を付与したイベント通知が、受信側のプロセスで優先度と受信順序により、正しく処理されることを確認するために、検証用のアプリを作成して実装と動作の検証を行った。

・検証1:室温変化と画面操作の競合

【検証内容】

連続した画面の操作(ボタン連打)中でも、重要な事象通知がリアルタイムで処理される。

【検証方法】

現在の室温を表示する画面を作成し、室温変化を検出により現在の温度を表示する機能とタッチパネルのタップ操作によるボタン押下でボタン形状を変更する機能を実装して動作確認を行った。

【検証結果】

タップ操作によるボタン連打中でも室温変化が発生したタイミングで、室温の表示が変更されることを確認した。

2-4-3-3 プロセス間通信の性能検証

既存の標準／拡張ライブラリに比べて処理性能が大きく劣化していないことを検証する必要がある。

・拡張インターフェースによる性能改善

【検証内容】

性能測定の結果からデータコピーのサイズに比例して、性能が劣化することが判明した。データコピーを減らす仕組みを検討した結果、上位プログラムと LPC のイベント送信関数にて、引数の送信データ、受信データの受け渡しで不用なデータコピーを行なわずにイベント通知を行う拡張インターフェースを提供することとした。

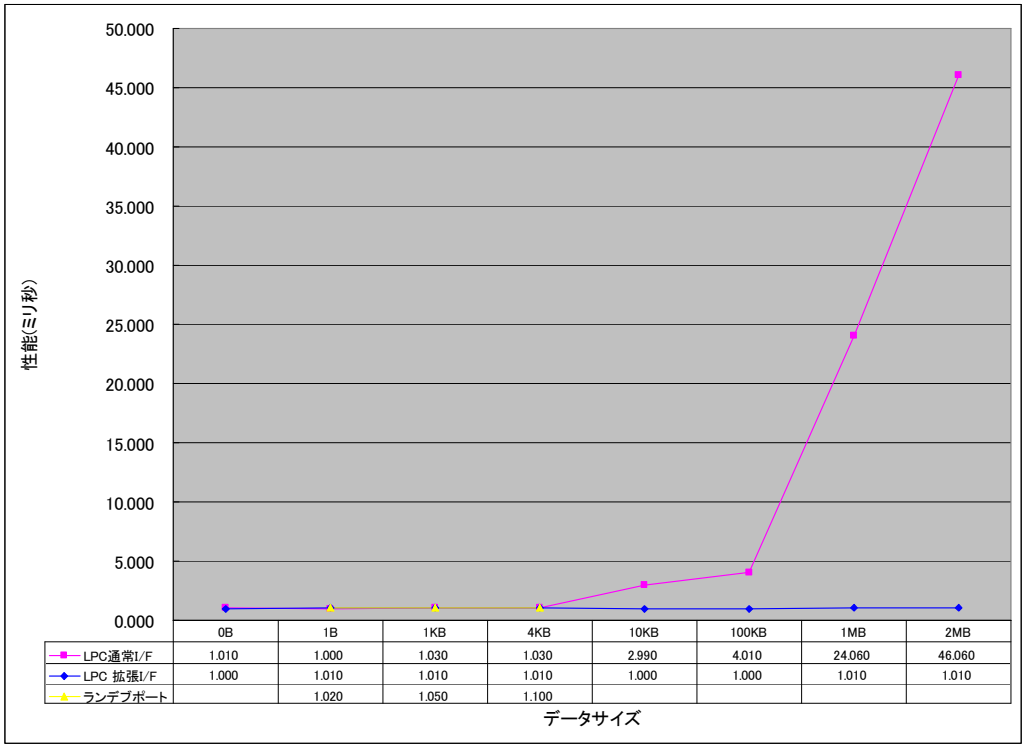
これにより、イベント通知を行う前に用意したメモリ空間に直接上位プログラムが送信データと応答データを書き込むことで、API 関数間での不要なデータコピーを回避することが可能である。

【検証結果】

拡張インターフェースを使用することにより大幅に性能が向上した。既存の標準／拡張ライブラリで同期イベント通知とほぼ同等の機能であるランデブポートと性能を比較しても大きな差は存在せず LPC を使用することによる性能面での影響はないと考える。

また、既存の標準／拡張ライブラリのランデブポートは、送受信可能なデータサイズが１～４０９６バイトとなっており、データサイズが増えた場合の比較が行えない。

ランデブポートと比較して、機能面での優位性があり更にデータサイズに影響を受けない拡張インターフェースの付加価値は大きいと考えている。以下に LPC の通常インターフェース、LPC 拡張インターフェース、既存のランデブポートの性能比較を行ったデータを示す。



	0B	1B	1KB	4KB	10KB	100KB	1MB	2MB
LPC通常I/F	1.010	1.000	1.030	1.030	2.990	4.010	24.060	46.060
LPC 拡張I/F	1.000	1.010	1.010	1.010	1.000	1.000	1.010	1.010
ランデブポート		1.020	1.050	1.100				

【性能測定における制約について】

アプリプロセスからのシステム時刻取得の精度が１０ミリ秒単位となっており、正確な時刻が取得できないため、システムパラメータを変更してタイマ精度を１ミリ秒にすることで測定を行った。しかし、プロセス間通信の処理の性質上から一回の通信は、数百マイクロ秒で処理可能であると考えられる。

測定にはマイクロ秒の精度での検証も行う必要があるため、上記の測定以外にも詳細な処理の性能検証にはマイクロ秒単位で測定可能なタスクトレーサを用いて行った。

性能測定の処理時間の中にはイベント通知からイベント受信の間にタスク切り替えの待ち時間が含まれている。参考として、タスクトレーサで測定した結果に対する分析結果を以下に示す。これにより、タスク切り替

えの待ちを除外しても、既存機能との大きな性能差はない※と考えている。
※既存機能の 436.290 ミリ秒に対して、LPC は 539.656 ミリ秒であり、目標の30%以内である

【LPC通信性能のタスクトレーサの分析結果】

経過時間(μ秒)	種別	タスク名	処理名
125,736	測定開始		
162,448	処理開始～終了	送信	wai_sem
207,480	処理開始～終了	送信	sig_sem
236,496	処理開始～終了	送信	snd_mbf
285,584	処理開始	送信	rcv_mbf
338,000	タスク切替	送信	
457,016	処理終了	受信	dly_tsk
477,400	処理開始～終了	受信	rcv_mbf
566,736	処理開始～終了	受信	wai_flg
597,208	処理開始～終了	受信	set_flg
632,048	処理開始～終了	受信	set_flg
658,048	処理開始～終了	受信	rcv_mbf
700,064	処理開始～終了	受信	wai_flg
721,072	処理開始～終了	受信	set_flg
744,368	処理開始～終了	受信	set_flg
771,824	処理開始～終了	受信	snd_mbf
817,896	処理開始	受信	dly_tsk
839,736	タスク切替	受信	
944,464	処理終了	送信	rcv_mbf
963,392	測定終了		

性能 837.656 ミリ秒
待ち時間 298.000 ミリ秒
待ちを除いた性能 539.656 ミリ秒

【既存機能(ランデブポート)のタスクトレーサの分析結果】

経過時間(μ秒)	種別	タスク名	処理名
115,128	測定開始		
123,344	処理開始	送信	cal_por
179,920	タスク切替	送信	
540,112	処理終了	受信	btk_dly_tsk
556,856	処理開始～終了	受信	get_nam
644,944	処理開始～終了	受信	acp_por
746,864	処理開始～終了	受信	rpl_rdv
783,264	処理開始～終了	受信	get_nam
850,760	処理開始～終了	受信	acp_por
897,144	処理開始	受信	btk_dly_tsk
918,464	タスク切替	受信	
1,018,408	処理終了	送信	cal_por
1,032,864	測定終了		

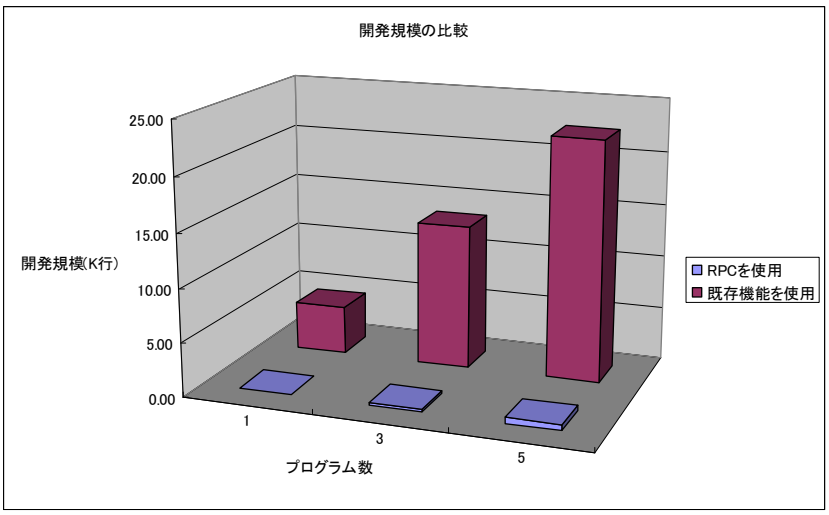
性能 917.736 ミリ秒
待ち時間 481.456 ミリ秒
待ちを除いた性能 436.280 ミリ秒

2-4-3-4 ネットワーク上の機器との通信に関連する開発効率を改善

上位プログラム側でRPCと同様の機能を持たせる場合、プログラム当たり子機の管理で約3.5K行、他プラットフォームを含む親機の管理で約1.0K行の実装が必要と考える。

RPCを使用した場合は、RPCの提供する関数の呼び出し部分の実装が必要となり、その規模は、プログラム当たり約100行と考えている。RPCを使用した場合と既存機能を使用する場合と比べ、約2%の規模で開発可能であり、更に上位アプリの数に比例して効率化される規模は大きくなる。

また、他プラットフォームとの連携するための、プラットフォーム向けの共通処理も提供可能である。今回の開発では、Windows の C#言語プログラムと Linux の C 言語向けの共通処理を作成した。



プラットフォーム	開発規模(有効 K 行)
Windows	0.5
Linux	0.8

この結果から、T-Kernel プラットフォームだけでなく、他のプラットフォームとの連携を含めたシステム開発における開発効率の改善は十分に行えていると考える。さらに様々なプラットフォームからも容易に親機、子機を管理する機能を実装することが可能であることを検証できた。

2－5　アプリケーション開発

前記①～④の開発項目はミドルウェアが主となる為、開発の効果を確認する為、アプリを開発し、視覚的な確認を行った。以下に実際に開発したアプリと効果の特徴を記載する。

■タッチパネル
◇長押し
一つの表示部品(パーツ)で複数の操作が可能となった。

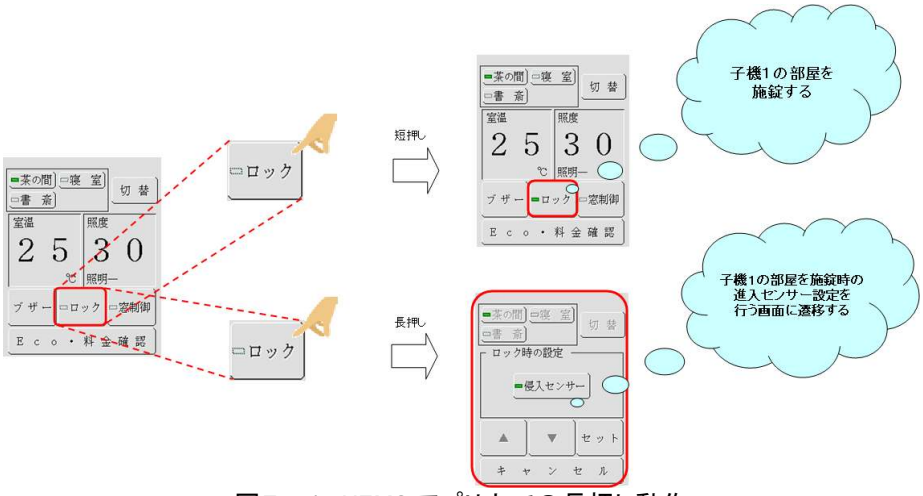


図5－1．HEMS アプリ上での長押し動作

◇リピート

シリアルボックス以外のパーツでもリピート操作が可能となり、他のパーツと連動できるようになった。
下図はボタンパーツ押下で、テキストの表示が変更されている図

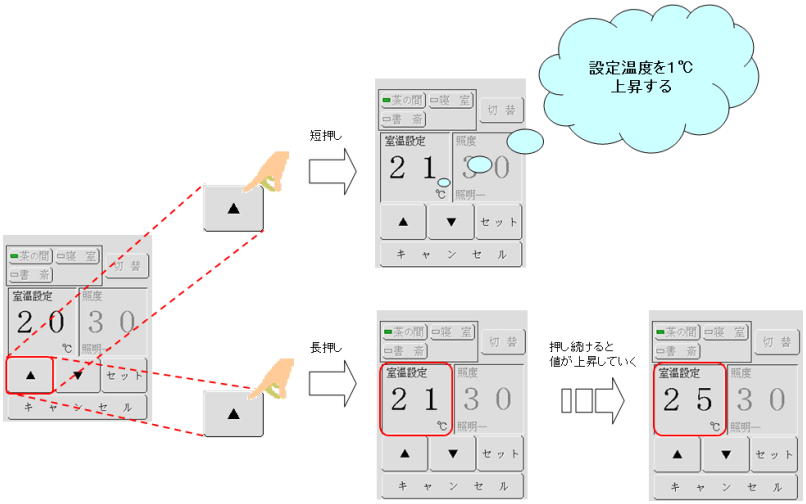
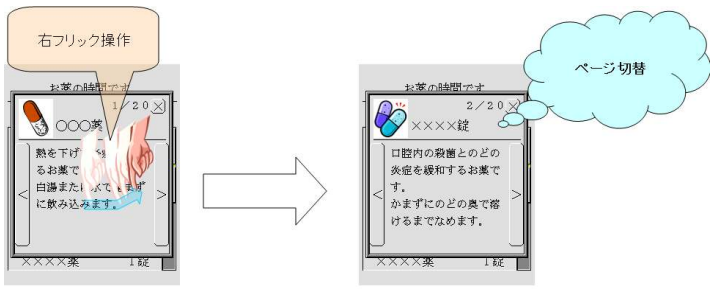


図5－2．HEMS アプリ上でのボタンパーツによるリピート操作

◇フリック
新規のイベントが追加されたことで、ユーザの操作性が向上した。



◇マルチタッチ
タッチパネルのみで簡易キーボード並みの入力が可能となった。

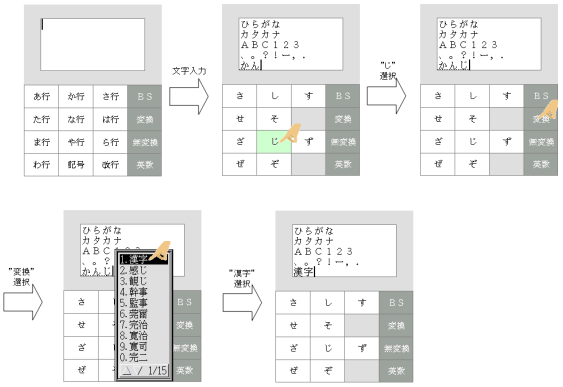


図5－5．メモアプリ上でのマルチタッチ操作

■キー操作
◇長押し
一つのボタンで複数の操作が可能となった。

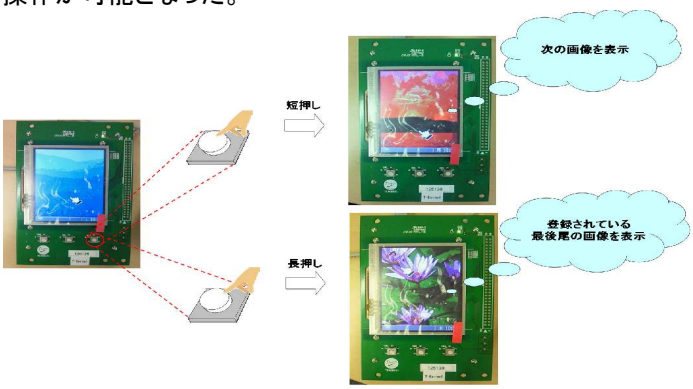
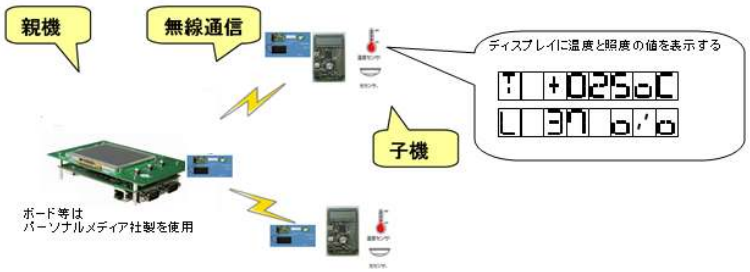


図5－6．フォトブック上におけるキー長押し操作

◇子機通信
無線通信で、離れた位置の子機の状態を監視、遠隔操作が可能となった。



無線通信で室温の設定/参照、
照明のON/OFF、ブザーの鳴動が
可能となっている。

2－6 用語説明

本文に記載した略語について正式名を以下に示す。

略語	読み
OS	オペレーティングシステム
RTOS	リアルタイム OS
SW	ソフトウェア
HW	ハードウェア
PF	プラットフォーム
MW	ミドルウェア or ミドル
lib	ライブラリ
T-Kernel/SE	T-Kernel/StandardExtension
HEMS	Home Energy Management System
UI	ユーザインターフェース
GUI	グラフィカルユーザインターフェース
I/F	インターフェース

略語（本文特有）	読み
APL	アプリケーションまたはアプリ
KEYM	キー制御ミドル
TPM	タッチパネル制御ミドル
APM	アプリケーション管理ミドル
LPC	Local Procedure Call （プロセス間通信）
RPC	Remote Procedure Call （遠隔手続呼出）

第3章　まとめ

約3ヶ月という短期間の開発であったが、下記5つのテーマ全ての開発を行う事ができた。
テーマ①③④⑤については、目標全てに対して、実際にT-Kernel上でプログラム開発と動作検証を行い、
テーマ②についても、コア部分について実際の動作検証を行う事ができた。

本研究開発では、タッチパネルやアプリケーション制御、プロセス間通信等、実際の商用機器の開発で要求された、厳しい品質、高い信頼性、高い商品性を実現してきた我々の組込SW開発スキルを、T-KernelベースのSW開発においても十分に効果が出せる事を確認できた。

今後は、本研究開発の成果物を用いて、T-Kernel搭載の組込機器の発展にも貢献していきたい。

◇研究目的

UI関連ミドルウェアやプロセス間通信ミドルウェアの充実化による
・アプリケーション開発の開発規模の削減。開発期間の短縮
・アプリケーション開発のSW開発品質の向上
・タッチパネルやキーボタン操作の操作性の向上
・タッチパネルやキーボタン操作の信頼性の向上

◇研究開発テーマ

①キーボタンの操作性の向上　　（目的：UI 系の機能強化）
②キーボタン操作の信頼性の向上（目的：UI 系の機能強化）
③タッチパネルの操作性の向上　（目的：UI 系の機能強化）
④ミドルウェア開発　　（目的：基盤ミドル充実＆通信ミドル強化）
⑤アプリケーション開発　　（目的：①②③④の動作検証）

◇成果物の効果

・LPC／RPC等、基盤ミドルウェア
⇒T-Kernelでの新規SW開発時の効率化/品質向上/信頼性向上

・キー制御ミドルとアプリケーションマネージャ
⇒キーボタン搭載組込機能のキーボタン操作性の向上/信頼性の向上/商品性の向上
キー操作対応の新規SW（アプリ）開発時の効率化

・タッチパネル制御ミドル
⇒タッチパネル搭載組込機器のタッチパネル操作性の向上/信頼性の向上/商品性の向上
タッチパネル対応の新規SW（アプリ）開発時の効率化

◇成果物が効果を発揮する分野

・リアルタイム性が要求される画面表示とUIを有する組込機器
・キーボタンやタッチパネル等、ユーザ操作の結果に対して、高い信頼性が必要とされる組込機器
・HEMS（Home Energy Management System）等の、異なる端末を複数制御する組込機器
・タッチパネル等をつかった柔軟または高い操作性が要求される組込機器
・画面有無に関係なく、複数機能を搭載し、プロセス間通信が多発する組込機器
・下記の様なHW環境でも充実したUI機能が求められる組込機器

◇補足

開発環境
・CPU　SH7727（100MHz）　ARM(200MHz)
・搭載メモリ最小 16MB
・タッチパネル付のT-Kernelが動作するボード
・画面周りの開発は市販のT-Kernel/PFを活用
（本研究で開発したミドルウェアは特定PFに依存しない様に開発を行った。）

以上