修士論文　　2008年度(平成20年度)

マイクロユビキタスノードにおける
高可用性を実現するバッテリ管理機構

慶應義塾大学大学院 政策・メディア研究科

森 雅智

修士論文要旨　　2008年度(平成20年度)

マイクロユビキタスノードにおける高可用性を実現するバッテリ管理機構

論文要旨

近年，携帯電話やセンサノードといった，バッテリ駆動型で複数のアプリケーションが同時に動作するマイクロユビキタスノードに関する研究が活発である．マイクロユビキタスノードは急激な進化を遂げており，一台のノードに搭載されるアプリケーションの種類は年々増加している．例えば，初期の携帯電話は電話機能だけだったものが，今日ではメール送受信機能や音楽再生，TV 視聴機能等が搭載されている．

一方で，マイクロユビキタスノードの電源管理は複雑化している．第一に，複数アプリケーションが同時に動作する環境では，単機能のデジタルカメラなどと違いバッテリ切れの予測が困難である．それぞれのアプリケーションは異なるデバイスを異なる使用頻度で使用するため，消費電力はアプリケーションや利用者の使用頻度に応じて異なる．そのため単純に残り動作時間を予測することができず，利用者の意図しないタイミングでバッテリが切れてしまうという問題が発生する．第二にマイクロユビキタスノードでは複数のアプリケーションが一つのバッテリ資源を共有するため，利用者にとって優先度の低いアプリケーションが，より優先度の高いアプリケーションの動作を抑制してしまうという問題がある．こうした問題は，ゲームなどのさほど重要でないアプリケーションでは大きな問題にならないが，人命に関わるような安全性を確保するためのアプリケーションにとっては致命的な問題である．こうした問題により，現在のマイクロユビキタスノードは高可用性を要求される分野に適用するのは難しい．

本研究ではこうした複数のアプリケーションが混在するマイクロユビキタスノード環境において，特定アプリケーションの動作時間を保証するシステムを提案し，実装，評価を行った．そして動作保証を実現する上で，アプリケーション別の消費電力測定機構とその収集データを基にアプリケーションを制御する機構を作成した．本システムにより，重要なアプリケーションの動作時間を保証することができるようになり，従来マイクロユビキタスノードが提供することのできなかった品質保証型のサービスが実現可能となる．

キーワード：

1 資源管理　2 ユビキタスコンピューティング
3 消費電力　4 電源管理　5 高可用性　6 マイクロユビキタスノード

慶應義塾大学大学院　政策・メディア研究科
森 雅智

Abstract of Master's Thesis

Academic Year 2008

Battery Resource Management Method to Achieve High Availability for Micro Ubiquitous Nodes

Summary

Multifunctional mobile devices such as smart phones, digital audio players and sensor nodes have gained significant attention and popularity over the last years. They are often referred to as Micro Ubiquitous Nodes (MUNs). MUNs are battery-powered, and could execute more than one applications simultaneously. Additionally, the number of applications executed on a MUN has been increasing rapidly. For example, a traditional cellular phone was merely a portable phone, while the modern equivalent could be used for reading and writing emails, browsing web sites, listening to music, watching TV shows and playing games.

Though this multi-functionality has improved our life to become more convenient, the battery resource management has also become more complex and difficult. It is difficult to estimate the lifetime of a MUN because energy consumption of each applications are diverse. It depends on the resource consumption behavior of the application; such resource including CPUs, networks, sensors, LCDs, and any other devices. For example, some applications may require an Internet connection, while others may not. This problem causes unexpected battery shutoffs. Also, because all applications consume the same battery resource, low-priority applications could excessively consume battery resource, while high-priority applications are blocked. The problem is unignorable in mission critical application. For these reasons, it is difficult to adopt applications which requires high availability to current MUN platforms.

This thesis describe an energy reservation system pSurvive, which manages energy consumption for each individual application running on a MUN. pSurvive guarantees application service running time by two modules. One module monitors energy consumption of all applications, and estimates the future energy consumption for each application. The other controls non-reserved applications to save the battery.

Keywords:

1 Resource Management 2 Ubiquitous Computing 3 Energy Consumption
4 Battery Resource Management 5 High Availability 6 Micro Ubiquitous Nodes

Keio University Graduate School of Media and Governance
Masato MORI

目 次

第 1 章　序論 **1**
1.1　研究背景 2
1.2　研究目的 4
1.3　本論文の構成 5

第 2 章　マイクロユビキタスノードと電源管理 **6**
2.1　マイクロユビキタスノードの普及 7
2.1.1　マイクロユビキタスノードの種類と構成 7
2.1.2　マイクロユビキタスノードを用いたアプリケーション・サービス 11
2.2　マイクロユビキタスノードにおける電力消費 13
2.2.1　電力管理の現状 13
2.2.2　アプリケーション別の消費電力の違い 14
2.3　マイクロユビキタスノードにおける電力管理問題 14
2.3.1　予測不可能な残り動作時間 14
2.3.2　利用者側からの制御自由度の低さ 14
2.3.3　電力資源の共有に起因する優先度逆転問題 15
2.4　本章のまとめ 15

第 3 章　システム設計 **16**
3.1　想定環境 17
3.2　機能要件 18
3.3　基本設計 19
3.3.1　アプリケーション別消費電力計測・予測機構 19
3.3.2　アプリケーション動作時間保証機構 22
3.4　本章のまとめ 25

第 4 章　実装 **26**
4.1　実装環境及び動作環境 27
4.1.1　ハードウェアの選定 27
4.1.2　ソフトウェアの選定 29
4.2　ソフトウェア構成 29
4.2.1　アプリケーション別消費電力計測・予測機構 30
4.2.2　アプリケーション別動作時間保証機構 33

4.3 本章のまとめ 35

第 5 章 実験と評価 **36**
5.1 実験環境の設定 37
5.2 デバイスの単位時間あたり消費電力の計測 37
5.3 アプリケーション別消費電力計測機構の評価 39
5.4 本章のまとめ 41

第 6 章 関連研究 **42**
6.1 資源管理に関する研究 43
6.1.1 リアルタイム処理 43
6.1.2 資源予約 45
6.2 省電力に関する研究 46
6.2.1 プロセス毎の資源管理 46
6.2.2 デバイス毎の資源管理 47
6.2.3 消費電力計測・予測 47
6.3 まとめ 48

第 7 章 結論 **49**
7.1 今後の課題 50
7.1.1 アプリケーション別消費電力計測・予測機構の課題 50
7.1.2 アプリケーション別動作保証機構の課題 51
7.1.3 システムの可搬性に関する課題 51
7.2 まとめ 52

図 目 次

1.1 多機能ポータブルオーディオプレーヤ iPod Touch 3
1.2 Panasonic LUMIX FX37 の仕様（バッテリに関する部分のみ抜粋） . . 3
1.3 docomo PRO series HT-01A の仕様（バッテリに関する部分のみ抜粋） 4

2.1 各デバイス毎の多機能化の過程 10
2.2 携帯電話におけるバッテリ残量表示 13

3.1 pSurvive システム構成図 20
3.2 デバイスの使用状態と消費電力量の関係 21

4.1 Gumstix Vertex XM4-BT 27
4.2 NUTS センサボード 28
4.3 クロスコンパイル環境 30
4.4 アプリケーション別消費電力計測・予測機構の構成 31
4.5 アプリケーション別動作時間保証機構の構成 34

5.1 無線 LAN の ON/OFF による電力消費量の変化 37
5.2 通信の有無による電力消費量の変化 38
5.3 CPU 負荷の有無による電力消費量の変化 39
5.4 予測値と実測値の変化 41

6.1 ソフトリアルタイムの評価関数 43
6.2 ハードリアルタイムの評価関数 44
6.3 破滅的なハードリアルタイムの評価関数 44

表 目 次

2.1 マイクロユビキタスノードの進化と同機能化 11

4.1 Gumstix Vertex XM4-BT の仕様 28

5.1 各デバイスの単位使用量辺りの消費電力 39
5.2 CPU とネットワークを使用するアプリケーションの資源使用量 40

第1章

序論

本章では，本論文の背景となるマイクロユビキタスノードとその応用について述べ，その要素技術である電源管理技術について述べる．その後，現在のマイクロユビキタスノード環境における問題を明らかにし，本研究の目的を示す．最後に本論文の構成を記す．

1.1 研究背景

近年，情報科学や電子工学の活発な研究開発により，小型端末向け組み込み CPU の小型化，低消費電力化，様々な低消費電力センサの登場，小型無線通信用 LSI の登場といった進化が起こっている．これらの技術は携帯電話やポータブルオーディオプレーヤ，センサノードといった実際の製品への利用が進んでおり，今日我々の生活に急激に浸透している．

本論文では，こうした急激に普及が進んでいるデバイスをマイクロユビキタスノードと呼称する．本論文で扱うマイクロユビキタスノードとは，以下の条件を満たすデバイスである．

1. 人が容易に持ち歩けるか，部屋や公共空間などに埋め込める程度に小型なこと
2. バッテリ駆動であること．ただし充電可能なものも含む
3. 一台のノード上で複数のアプリケーションが動作すること

この定義に従うと，昨今の多機能携帯電話や無線 LAN への接続が可能なポータブルオーディオプレーヤ，PDA （Portable Digital Assistant），複数アプリケーションが動作可能なセンサノードなどのデバイスはマイクロユビキタスノードである．一方，デジタルカメラは上記 1，2 の定義には従うが，写真を撮るという単機能のデバイスであるためこの定義に含まれない，ただし，昨今の上位機種に見られるようなネットワーク連携機能などを有していればその限りではない．

こうしてみると，PDA といった例外を除けば，現在マイクロユビキタスノードとなるデバイスの多くは元は単機能のデバイスであったことが分かる．初期の携帯電話は電話機能のみが搭載されていたし，ポータブルオーディオプレーヤもオーディオ再生機能のみに特化されていた．センサノードは現在も消費電力や導入コストの面で単機能のものが用いられているが，複数のアプリケーションを動作可能とするための研究も盛んである．

こうした従来の単機能のデバイスを基に，より便利にするために機能を追加したものがマイクロユビキタスノードの多くを占める．携帯電話はネットワークに接続することにより，ソフトウェアの追加や音楽データといったマルチメディアコンテンツの配信が可能となったし，メール機能は今では電話機能以上に重要度が増している．昨今では電子マネーとの統合も進み，ネットワーク越しに入金することも可能となっている [10]．ポータブルオーディオプレーヤは音楽データだけでなく，画像の閲覧やビデオの再生に対応することで，より手軽にマルチメディアコンテンツを持ち歩けるようになっただけでなく，その画面の広さを活かしてカレンダーや Web ブラウザ，メールといった機能までも搭載するようになった（図 1.1）．こうした事例から見るに，今後もマイクロユビキタスノードが普及していくことは確かだろう．その他のマイクロユビキタスノードの具体的な機能や応用例については第 2 章で詳しく述べる．

こうしてマイクロユビキタスノードが急速に普及していく中で，問題となっているのがバッテリ管理である．ここではマイクロユビキタスノード固有の問題を明らかに

図 1.1: 多機能ポータブルオーディオプレーヤ iPod Touch

プリンター連携	PictBridge（プリントサイズ・レイアウト・日付印刷指定対応）
電源	リチウムイオンバッテリーパック（付属、3.6V）、ACアダプター（別売、100～240V対応）
撮影可能枚数*	CIPA規格※8：約310枚 （付属バッテリーパック）
充電時間 （付属バッテリーパック）	最大約120分
外形寸法	幅約94.7 × 高さ約51.9 × 奥行き約22.0 mm（突起部を除く）

図 1.2: Panasonic LUMIX FX37 の仕様（バッテリに関する部分のみ抜粋）

するために，同じバッテリ駆動型でも単機能のデバイスとの比較を挙げながら解説していく．

バッテリ駆動型デバイス共通の問題としてまず考えられるのは，バッテリ寿命の問題である．バッテリ寿命はデバイスの性能を測る大きな指標である．バッテリ寿命が長ければそのデバイスの可用性は向上する．例えばバッテリが一日保たない携帯電話は使い物にならないように，充電型のデバイスであれば充電周期分だけの動作時間を確保することが一つの大きな指標になる．また，使い切り型のデバイスであればバッテリ寿命の長さがシステムの寿命となるので，バッテリ寿命はコストに直結するより重要な要素となる．このように，バッテリ寿命の問題については単機能，多機能に関わらず同じ問題を持っていることが分かる．

一方，バッテリ寿命をデバイス単位ではなくアプリケーション単位で見た場合，マイクロユビキタスノードはより複雑な問題を抱えている．単機能デバイスの場合，バッテリの利用用途は一種類のため，容易にアプリケーションの駆動時間を予測することができる．例えば，デジタルカメラメーカーでは満充電から何枚の写真が撮影できるかという形で仕様を公開している（図 1.2）．この値は概ね正確であり，信頼できる．一方，多機能を有するマイクロユビキタスノードにおいてはこうしたアプリケーションを基にした駆動時間の予測は困難である．なぜなら，アプリケーションから見た場合，複数のアプリケーションが一つのバッテリを共有して使用するため，一つのアプリケーションだけに着目したバッテリ駆動時間を知るだけでは意味がないからである．

質量	約 172g
連続待受時間 （静止時）	3G：約 400時間 GSM：約 240時間
連続通話時間 （音声通話時）	3G：約 220分 GSM：約 240分
メインディスプレイ	約 2.8インチ

図 1.3: docomo PRO series HT-01A の仕様（バッテリに関する部分のみ抜粋）

携帯電話メーカーでは連続待受時間や連続通話時間といった形でバッテリ寿命の仕様を公開している（図 1.3）が，これではメールや Web ブラウザを使う場合にはどの程度バッテリが保つのかはわからない．昨今の利用者は，通話機能よりもむしろメールや Web といった機能を重視しているので，こうした通話機能のみを考えたバッテリ寿命情報の有用性は低い．

こうした背景から，マイクロユビキタスノードにおいては，利用者がバッテリ残量を見ながら自分でバッテリ管理をしているのが現状である．しかし，利用者自身が管理するのではちょっとしたミスが頻発し，携帯電話で TV を見ていたらバッテリが切れてしまい，重要なメールが見られなくなったり，電子マネーでの決済ができなくなったりしてしまうというリスクが常に付きまとう．

こうしたアプリケーションの動作時間を保証できないという問題がマイクロユビキタスノードの可用性を下げ，信頼性を下げる要因となっている．

1.2 研究目的

前節で紹介した通り，現状のマイクロユビキタスノードにはアプリケーションの動作時間を保証できないという問題がある．そこで本研究では，アプリケーション毎の消費電力を細かく計測，予測し，そのデータを基にした指定アプリケーションの動作時間保証を実現するシステム pSurvive を提案する．

具体的な例を挙げて説明する．アプリケーション A，B，C（以下 A，B，C とする）が動作する環境において，A が動作時間保証を行いたいアプリケーションとし，B，C は利用者によって任意に起動されるものとする．こうした環境において，本研究では以下の形式で動作時間保証を実現する．

1. 将来起動される A の動作時間を N 分保証する
2. 現在起動中の A の残り動作時間を N 分保証する

二つの違いは動作保証を実行するタイミングの違いだけで，保証内容は同様である．利用者が A の動作時間を予約すると，pSurvive は A が N 分動作するための電力量を確保し，A 専用とする．A 専用に確保した電力量は，A 以外によって使われることはない．その後，B，C が起動してバッテリが A 専用に確保した分しか残らなくなると，pSurvive は B，C を停止する．それにより，A の動作時間を保証することができる．

また，このシステムを実現するためには，A，B，C のアプリケーション別の使用電力量を知る必要がある．そのため，pSurvive は常時ノード上の各デバイスを監視し，どのアプリケーションがどのデバイスをどれだけ使用しているかを記録する．消費電力は CPU やネットワークなどのデバイス毎に異なるので，デバイス毎の使用量と単位あたりの消費電力から実際のアプリケーションの電力使用量を計算し，それを計測結果とする．さらに，将来の電力使用量もこの計測結果を基に計算する．pSurvive の具体的な動作については 3 章で詳しく解説する．

本研究が達成されることによって，二つの成果が得られる．一つは現状のマイクロユビキタスノード環境に pSurvive を導入することにより，重要度の高いアプリケーションの可用性を高めることである．これにより利用者の利便性が向上するだろう．もう一つはアプリケーションの動作時間保証によって，従来は実現できなかった新しいサービスが出現する可能性が生まれることである．特に，高可用性，高信頼性を要求されるサービスが実現可能となるだろう．

1.3 本論文の構成

本論文は，全 7 章で構成される．次章では，本研究の対象とするマイクロユビキタスノードについて述べ，電源管理の問題について言及する．3 章では，2 章で取り上げた問題を基に，マイクロユビキタスノードにおけるアプリケーション動作保証機構の設計について解説する．4 章ではその実装について解説し，5 章で評価結果を述べる．6 章では本研究の関連研究について紹介し，7 章で本論文のまとめを行い，今後の課題について言及する．

第2章

マイクロユビキタスノードと電源管理

本章では，本研究の対象となるマイクロユビキタスノードのこれまでの進化と現状について述べる．その後マイクロユビキタスノードにおける電源管理の現状と問題について論じる．

2.1　マイクロユビキタスノードの普及

本節ではマイクロユビキタスノードの進化と現状の具体例について触れる．まずマイクロユビキタスノード製品の種類と構成を紹介し，次にマイクロユビキタスノードを用いた具体的なアプリケーションやサービスについて紹介する．これらについて触れることで，本研究の対象となるマイクロユビキタスノードの現状を示す．

2.1.1　マイクロユビキタスノードの種類と構成

第 1 章で述べた通り，マイクロユビキタスノードの多くは単機能，または限られた機能のために特化したデバイスが多機能化したものであると考えられる．まずは各デバイスの種類別にどのようにして多機能化が進んでいったかを示すことで，次節で述べる問題の背景を解説する．本論文では具体例として，昨今の代表的なマイクロユビキタスノードである携帯電話，ポータブルオーディオプレーヤ，センサノード，PDA を挙げる．なお，特に国を限定しない場合は日本での出来事とする．

- 携帯電話

  携帯電話は今日のマイクロユビキタスノードの中でも最も普及しているものである．その進化は著しく，インフラ，ハードウェア，ソフトウェアが相互に進化してきた．その中でもインフラは一般に第一世代，第二世代，第三世代という区分けがなされているので，本章では各世代別に各要素の進化の過程を解説する，

  第一世代携帯電話は 1993 年までの最も初期のアナログ式携帯電話である．当時の携帯電話は音声による通話のみを目的として作られており，データ通信を行うことはできなかった．この当時の携帯電話が持っていたデバイスは，通話機能に最低限必要なものだけである．ここで挙げる最低限の機能とは，マイク，スピーカー，液晶ディスプレイなどである．端末の大きさも車載電話より少しは小さい程度のものであり，気軽に持ち歩けるものでは無かった．また，ソフトウェアも書き換えが可能なものではなかった．

  その後，1993 年からデジタル方式の第二世代携帯電話が登場し始める．第二世代携帯電話の大きな特徴として，電子メールや SMS，NTT ドコモの i モードといった専用ネットワークへの接続が可能となったことと，ネットワークを通して様々なコンテンツやアプリケーションをダウンロードし，携帯電話に新たな機能を追加することができるようになったことが挙げられる．1997 年からスタートしたデータ通信サービスは，携帯電話にメール端末を接続して電子メールを送信するというものであり，これが携帯電話がインターネットに繋がるきっかけとなった．1998 年には SMS がスタートし，携帯電話単体でメールのやりとりが可能となった．これにより，当時テキスト通信の主流であったポケットベルを携帯電話が駆逐していくこととなった．1999 年には NTT ドコモの i モードサービスがスタートし，専用ネットワークに接続してコンテンツを視聴する事が可能と

なった．2000 年にはカラー液晶を搭載した端末が登場し，コンテンツも画像を含むものが登場した．2001 年には NTT ドコモの端末で動作する i アプリが登場し，ゲームなどのコンテンツが広く開発されるようになった．2002 年には着うたといった比較的大きめのデータもやりとりされるようになり，カメラが搭載されるようになった．その後，第二世代携帯電話は同年に登場した第三世代携帯電話に徐々に移行していったが，2009 年 1 月現在でもまだサービスは提供されている．

2002 年にサービスを開始した第三世代携帯電話の大きな特徴は，第二世代携帯電話に比べて高速なデータ通信をサポートし，国際ローミングをサポートしたことである．通信サービスとしても，2004 年には従来の従量課金制の料金体型に加えて定額制の料金体系が現れ，よりネットワークサービスの重要性が高まっていることが伺える．また，データ通信速度は従来の第二世代携帯電話の数十 Kbps 程度から数 Mbps へと高速化したことで，ネットワークコンテンツも従来のテキストや静止画像，静的コンテンツ主体のものから動画やオンライン対応といったコンテンツが現れるようになった．また，ハードウェアに非常に多くの機能が追加されていったのもこの時期である．センサとしては加速度センサやジャイロセンサ，GPS が内蔵され，入力デバイスとしてはタッチパネルやキーボードが搭載されるようになり，従来の数字キーによる操作以外の選択肢が加わることとなった．ネットワークについては携帯電話網への接続以外に，Bluetooth を用いて PC やカーナビと接続することでモデムとして利用したり,，無線 LAN によって高速ネットワークへの接続を可能とするなど多くの選択肢を選べるようになった．その他，ワンセグチューナーによる TV 視聴機能，Felica に代表される IC カード機能が搭載される．また，カメラや動画といったマルチメディア処理のために専用 LSI を搭載するものが現れ，デジタルカメラと見劣りしない機能を実現した端末も現れていった．ソフトウェアについては，新モデルが登場する度に CPU の処理性能が上がっていったことと，搭載メモリ量が増えたことからアプリケーションもゲームを初めとしてより大容量で高性能な処理を必要とするものが増えていった．搭載 OS は従来は組み込み向け OS を専用開発することが多かったが，この頃になると TRON や Linux，Windows Mobile といった汎用プラットフォームを用いるものが主になった．

- ポータブルオーディオプレーヤ

  最初期のポータブルオーディオプレーヤは 1998 年に登場し，MP3 などの音声データの再生に特化したものであった．音声データの転送方法は PC と有線接続するしかなく，音声データのエンコードなどは利用者が手動で行う必要があった．またメモリーカードを交換可能な製品もあったが，記憶容量は 64MB〜128MB 程度であり，CD 約 2 枚分のデータが収まる程度であったため，当時主流であった MD と同等の容量でしかなかった．ハードウェアも曲名を表示するための液晶ディスプレイと再生関連のボタン程度しか無く，こちらも当時主流であった MD プレーヤと大きく変わらないものであり，多機能 MD プレーヤと同様にラジオ

や録音機能を備えたものも登場した．

しかし，1999〜2000 年頃から容量単価の安い HDD を搭載した製品が登場し始め，2001 年には iPod が登場し，利用者が容易に持ち歩くことのできる大きさのポータブルオーディオプレーヤとして広く普及した．初期の HDD 搭載製品では 4〜6GB のものが使われていたが，これは従来のメモリーカードを使ったものに比べて圧倒的に大きなものであったため，従来のメモリーカードを用いたものに比べて利用者は所有する音楽データの全てを入れて持ち歩く事も可能となった．また，膨大な音楽データを管理するため，iTunes や SonicStage を初めとする音楽データ管理ソフトウェアが製品に付属するようになった．

2004 年頃にはカラー液晶が搭載された製品が登場し始め，画像が閲覧可能となる．さらに 2005 年には動画再生に対応した製品が登場し，音声，画像，動画という主なマルチメディアコンテンツをサポートするようになった．また，ハードウェア面では入力装置にタッチパネルを採用する製品も現れた．

その後，2007 年には iPod touch に代表されるさらに機能を追加した製品が現れた．こうした製品では PC のスケジュールや電話帳データを同期したり，無線 LAN を通じて単独でネットワークに接続したりする機能を持つ．また，開発環境を配布して独自のアプリケーションを動作させることもできるようになった．他にも iPod Touch のようなジャイロセンサを持った製品や，カメラを搭載した製品も現れるようになった．

- センサノード

  センサノードの定義は広く，uPart[2] のような小型のものからフィールドサーバのような大型で多機能なものまであり，多彩である．その中で本論文が対象とするのは 1 章で述べた通り，複数アプリケーションが動作し，十分に小型なものである．具体的には Crossbow 社の MOTE[3] や Gumstix 社の Gumstix[7] といった製品がある．これらのセンサノードは複数アプリケーションを動作させるための OS を備え，複数のセンサと無線通信機能を備え，バッテリ駆動である．

  現状は他のマイクロユビキタスノードとは違い，一般消費者向けに販売しているものではない．しかし，必要なアプリケーションはハードウェアがサポートしていれば，利用者が自分で導入することができる．

- PDA

  PDA は他のマイクロユビキタスノードと比較すると元から OS を備え，複数アプリケーション向けに設計されている点が異なっているように見える．しかし，初期の Palm や Zaurus などが音楽再生機能などを持たず，メモ帳やスケジュール管理機能といった電子手帳機能に特化していたと考えれば，他のマイクロユビキタスノードと同等に単機能から進化したと考えることができる．

  初期の PDA はディスプレイは白黒であり，1993 年にシャープ社の Zaurus PI-3000 や Apple の Newton が発売された．これらの機能はあくまで電子手帳的な役割に

とどまっていた．しかし，1996 年に Palm Pilot 1000 が発売されると，利用者が PDA を PC に接続して追加のソフトウェアをインストールすることで機能を追加することができるようになった．同年 Zaurus も MI シリーズを発表し，同様の機能をサポートするようになった．その後，カラー端末も現れるようになるが，当時の PDA はスピーカーや音声出力は持たず，液晶とタッチパネル，いくつかのボタンを備えていただけであった．

2000 年になると音声出力に対応するようになり，動画，音声再生機能が搭載された．また，この頃には携帯電話を通してネットワークに接続できる製品が登場し，Web ブラウザやメーラーといったソフトウェアが提供されるようになった．

その後はメモリーカードによる記憶容量の拡張，CF カードによる機能追加や無線 LAN に対応すると共に，処理速度やメモリ容量の増加により PC と同等のアプリケーションが動作するようになっていった．

以上，それぞれのマイクロユビキタスノードにおける機能の進化をまとめたものが，図 2.1 である．この図を見ると，センサノード以外の全てのデバイスにおいてネット

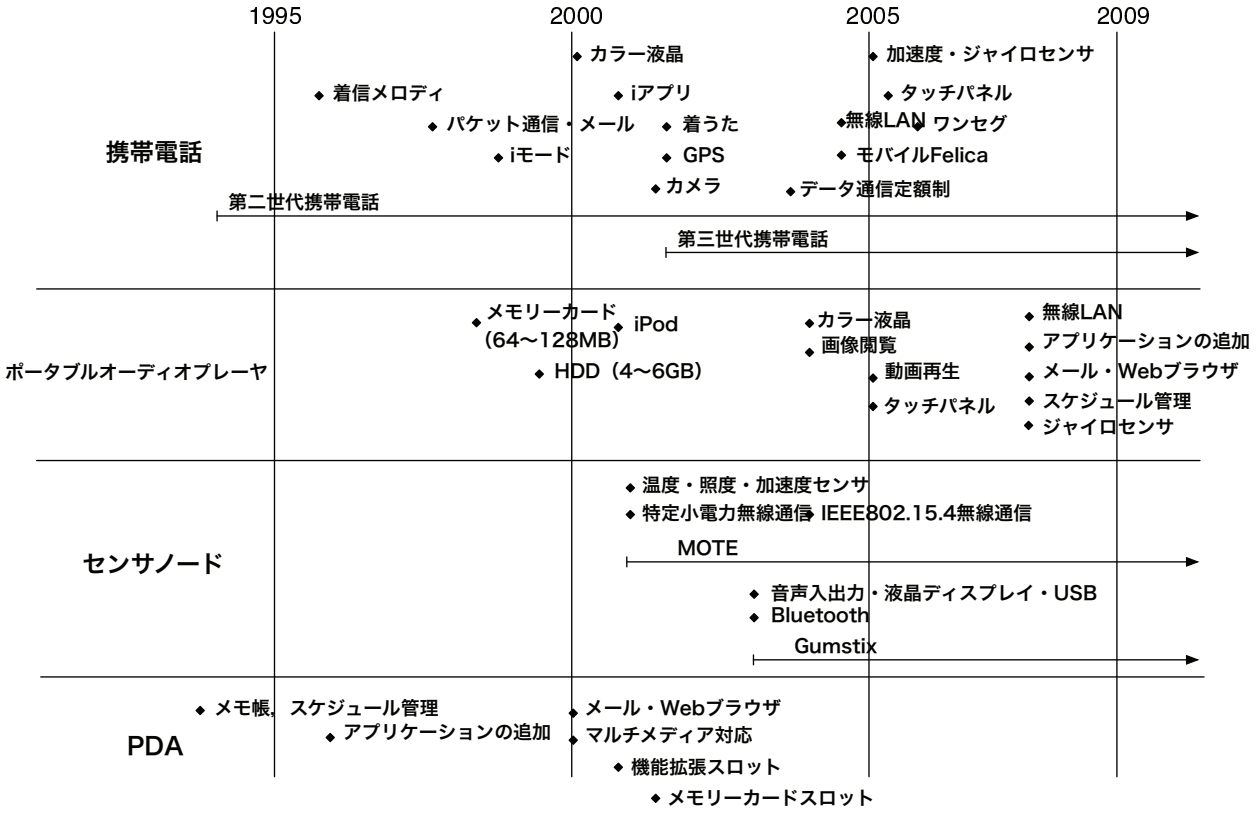

図 2.1: 各デバイス毎の多機能化の過程

ワークへの接続は後から追加された機能であることが分かる．またアプリケーションの追加機能については，時期は違えど全てのデバイスで実現されている．さらに，各デバイスが初期持っていた機能と最新の機能を表にしたものが表 2.1 である．下線が引か

表 2.1: マイクロユビキタスノードの進化と同機能化

	初期の機能	2009 年時点での機能
携帯電話	音声通話	音声通話，メール，Web ブラウザ，音楽・動画再生，画像閲覧，アプリケーションの追加，無線 LAN，カメラ，タッチパネル，センサ，GPS，ワンセグ
ポータブルオーディオプレーヤ	音楽再生	メール，Web ブラウザ，音楽・動画再生，画像閲覧，アプリケーションの追加，無線 LAN，タッチパネル，センサ
センサノード	センサ，無線通信	センサ，無線通信
PDA	スケジュール管理	メール，Web ブラウザ，音楽・動画再生，画像閲覧，アプリケーションの追加，無線 LAN，タッチパネル，拡張スロット

れた項目はセンサノード以外の全てのデバイスに共通する項目である．センサノードはアプリケーションやセンサを任意に実装可能なので，これらの共通項目を含んでいると考えてよい．このことから，単機能デバイスの多機能化は結果として同じ機能を持たせる方向で進化しているといえる．こうした傾向の典型例といえるのが，2004 年頃からアメリカで爆発的に普及した BlackBerry や 2008 年に日本でも発売された iPhone 3G のようなスマートフォンと呼ばれる製品である．これらの製品は，従来の携帯電話と PDA を合わせたコンセプトの製品であり，携帯電話の通信，通話機能と PDA のどのようなアプリケーションでも搭載できる汎用性を備えた製品として登場し，注目を集めている．

しかし，これらのデバイスは機能としては同じ機能を有していても，実際の利用用途が全く同じになるわけではない．次小節ではこれらのマイクロユビキタスノードを用いたアプリケーション・サービスについて解説する．

2.1.2 マイクロユビキタスノードを用いたアプリケーション・サービス

これまで，マイクロユビキタスノードそれぞれについてハードウェアとソフトウェアの面から機能を比較してきた．本小節では具体的なアプリケーションやサービスを取り上げ，それぞれのデバイスがどのような分野に利用されているのかを解説する．

- 携帯電話

  携帯電話は多機能化に伴って非常に多くのアプリケーションが提供されているが，本節では利用者が明示的に起動する能動的アプリケーションと，バックグラ

ウンドで自動的に動作する受動的アプリケーションに分けて解説することで，後述する電力管理の問題を明確にしていく．

ほとんどの携帯電話向けアプリケーションは，利用者が明示的に起動して利用する能動的なものである．例えば，乗り換え案内サービスやゲームアプリケーション，ワンセグ，メール送信といったものは利用者がサービスを受けたいときに起動し，サービスを受け終われば終了する．

一方，受動的アプリケーションは普段から動作していたり，ある一定周期で起動したりするものである．例えば，親が子供に GPS 搭載の携帯電話を持たせ，携帯電話から現在位置を管理サーバへと通知することにより，子供の現在位置を追跡するシステムなどが挙げられる．このサービスは，子供が危険な場所に出入りしていないかを知りたいという昨今の親のニーズに合致したサービスである．他にも，NTT ドコモの i チャネル [16] は自動的に最新のニュース情報を取得して更新するサービスである．このサービスは利用者が興味のある特定のニュース情報だけを自動的に取得して，表示する．

- ポータブルオーディオプレーヤ

  ポータブルオーディオプレーヤでは，Podcast と呼ばれる RSS フィードを利用したコンテンツの自動更新，ダウンロードの仕組みが広く使われている．Podcast では，ネットラジオやニュースといった日々更新されるコンテンツを，iTunes などのコンテンツ管理システムが自動的にチェックし，更新があれば自動的にダウンロードする．

  また，センサを使ったものでは Nike+iPod[1] が挙げられる．これは，靴にセンサを取り付け，センサとポータブルオーディオプレーヤ間で無線通信する事で，運動量を記録できるシステムである．

- センサノード

  センサノードは様々な応用が研究されているが，ここでは自然環境センシングの例として Airy Notes[8] を取り上げる

  Airy Notes はある特定の範囲の温度や照度といったデータを地理的に非常に細かいレベルでセンシングするシステムである．従来行われてきた都道府県や市町村レベルの情報ではなく，よりミクロに情報を集めることで，都市圏での地区別の気候の違いや公園などの影響を調べることができる．

- PDA

  PDA はその汎用性の高さと持ち運び可能な点，ソフトウェアの多くに PC と同じ資産が使えることから企業が社内システムの端末として採用する例がある．サービスとしてはメールやスケジュール管理，物品管理や工程管理などのシステムが多く，市販のソフトウェアを利用するケースもあるが，各社が独自に開発することも多い．

このように，各デバイスに非常に多様なアプリケーション，サービスが提供されている．しかしこうした多様なサービスが提供されることで，電力消費に関する問題がより顕著となってきた．次節ではこの点について解説する．

2.2　マイクロユビキタスノードにおける電力消費

本節ではマイクロユビキタスノードにおける電力消費の現状と，電源管理がどのように行われているかについて記す．

2.2.1　電力管理の現状

現在，多くのマイクロユビキタスノードにおいて，バッテリ残量の管理はノード単位で行われている．例えば，携帯電話のバッテリ残量は電池型のアイコンで示される（図 2.2）．

図 2.2: 携帯電話におけるバッテリ残量表示

利用者はこうした電池残量を基に，経験から残りどれくらいでバッテリが切れるのかを予測する．

電源管理に関する機能として，省電力機能を有した製品がある．こうした製品は液晶ディスプレイのバックライトの明るさを暗くしたり，常時使わない機能を停止することによって動作時間の延命を図る．また，モバイル Felica では携帯電話本体のバッテリが切れても，少しでもバッテリが残っていれば Felica 機能のみ利用することが可能である．しかし，完全に放電してしまうと，再充電を行うまで Felica 機能を利用することはできない．

2.2.2　アプリケーション別の消費電力の違い

アプリケーションによって大きく消費電力が異なるのがマイクロユビキタスノードの特徴である．例えば音楽再生であれば 10 時間程度動作するノードも，動画再生時には数時間でバッテリが切れてしまう．また，同じアプリケーションでも利用者の使い方によってその消費電力は大きく異なる．Web ブラウジングなどは，テキスト主体のサイトを閲覧するのであれば消費電力は少ないが，画像や動画主体のサイトを閲覧するとより多くのネットワーク資源や計算資源を使うため，消費電力は大きくなる．

2.3　マイクロユビキタスノードにおける電力管理問題

本節では前節で挙げた現状を基に，現在発生している問題についてまとめる．

2.3.1　予測不可能な残り動作時間

まず第一に，マイクロユビキタスノードでは残り動作時間が予測し辛いという問題が挙げられる．その原因は 2.2.2 で示したとおり，アプリケーションによって消費電力が異なることにある．単機能デバイスの場合，アプリケーションは一種類なので，動作時間を高い精度で予測することができる．バッテリは一つのアプリケーションにしか使われないので，動作時間が予測したものと大きくずれることはない．例えば残バッテリ量を 100 として，アプリケーションの消費電力が 1 時間に 20 とした場合，残り稼働時間は 5 時間であると容易に予測できる．

一方，複数機能を持つマイクロユビキタスノードにおいては，一つのバッテリを複数の消費電力の異なるアプリケーションが共有するため，単純に予測することができない．例えばアプリケーション A が 1 時間に 20，B が 40 の電力を消費する場合，残バッテリ量が 100 と分かっても A と B がどの程度使われるのかが分からなければ残り稼働時間は分からない．

このように，マイクロユビキタスノードでは残バッテリ量が分かっても残り動作時間を予測することは難しく，これが利用者の予期しないバッテリ切れという問題を引き起こす原因となっている..

2.3.2　利用者側からの制御自由度の低さ

次に，電源管理の自由度の低さが挙げられる．現状の電源管理では，デバイス全体の液晶ディスプレイの明るさを落としたり，一部のセンサの電源を落とすといったデバイス単位での制御となっている．しかし，この手法では同じデバイスを使うアプリケーションが複数あった場合に，特定のアプリケーションだけを抑制することはできない．例えば，定期的にメールチェックをするアプリケーションと Podcast から新着音声ファイルをダウンロードするアプリケーションの場合を考えてみる．音声ファイ

ルのダウンロードは通信料が多く，消費電力が大きいのでこちらのアプリケーションを抑制したいと考えたとしても，この場合はどちらのアプリケーションもネットワークへの接続を必要とするため，ネットワークデバイスを止めてしまうとメールチェックもできなくなってしまう．

このように，利用者が特定のアプリケーションだけを抑制しようとしても，デバイス毎の電源管理機能ではそうした要求を満たすことができない．

2.3.3 電力資源の共有に起因する優先度逆転問題

最後に，アプリケーション優先度の逆転問題を紹介する．複数のアプリケーションが提供されている場合，それらのアプリケーションに優先度が存在することがある．例えば，携帯電話において通話，メール，ワンセグといった機能が提供されているとして，ある利用者にとってはメール，通話，ワンセグの順に優先度が高いとする．この利用者がワンセグを視聴したり通話を行った後，メールアプリケーションを利用しようとした際に，バッテリが切れてしまって利用することができないというケースが発生する．これは結果的には優先度の低い通話やワンセグアプリケーションが，より優先度の高いメールアプリケーションの動作を抑制してしまっており，優先度の逆転が起こっていることを示している．

こうした優先度逆転問題は高信頼性を必要とするアプリケーションの場合に重大な問題となる．例えば，モバイル Suica のような電子マネーがバッテリ切れにより使えなくなったり，建物などで本人認証をするためのアプリケーションなどが使えなくなってしまうという事例は，ワンセグや音楽再生ができなくなるという事例に比べて被害の大きさが異なる．また，今後災害対策アプリケーションといった人の生死に関わるサービスが提供されていった場合，こうした優先度逆転問題は無視できないものになる．

2.4 本章のまとめ

本章では，本研究の対象となるマイクロユビキタスノードのこれまでの進化と現状について述べた後，その後マイクロユビキタスノードにおける電源管理の現状と問題について論じた．マイクロユビキタスノードは元は単機能のデバイスであったものが多機能化する事により，それぞれが同じような機能を持つこととなった．また，その電源管理はアプリケーションに着目したものではなく，物理的なデバイスに着目したものであり，アプリケーションの多機能化の現状に即したものでは無くなってきている事を示した．

第3章

システム設計

本章では前章で取り上げた問題を基に、マイクロユビキタスノードにおけるアプリケーション動作保証機構の設計について示す。まずは想定環境についてまとめ，pSurviveの必要条件を示す．その後，前章で挙げた問題を解決するためのpSurviveの機能要件をまとめ，具体的なシステム設計について述べる．最後にまとめを行う．

3.1 想定環境

本システムはマイクロユビキタスノードでの動作を前提としている．マイクロユビキタスノードとは，これまでにも触れた通り以下の条件を満たしたノードのことである．

- 人が容易に持ち歩けるか，部屋や公共空間などに埋め込める程度に小型なこと
- バッテリ駆動であること．但し充電可能なものも含む
- 一台のノード上で複数のアプリケーションが動作すること

次に，pSurvive 要求として，マイクロユビキタスノードの中でも次の機能を有している必要がある．

1. ノード全体のバッテリ残量が取得可能であること
2. 各デバイスがデバイスドライバのような共通 API を通して行われていること
3. 各デバイスの使用量を測定するために，最低でも各デバイスの ON/OFF 状態が分かること

一番目について，pSurvive はノードの残バッテリ量を元に消費電力量の測定を行う．そのため，ソフトウェア側が全体のバッテリ残量を知る方法が不可欠である．また，ここで取得できるバッテリ残量は 0，25，50，75，100 % しか状態が無いような大雑把ものではなく，値がある程度細かく取れることを前提とする．この値の細かさは，消費電力量測定の精度に関わるので重要な項目である．しかし，この機能は既存のほとんどのマイクロユビキタスノードが既に持っている機能である．例えば，NTT ドコモの端末で動作する Java ライブラリ Doja では，バッテリ残量を取得するための PhoneSystem クラスが定義されているし，Windows Mobile においても GetSystemPowerStatusEx API を呼び出すことでバッテリ残量が取得可能である．よって，この前提条件は実現上さほど問題にならないと考えられる．

次に，二番目と三番目は実装上の問題である．消費電力量測定の際，各デバイス別の使用時間を計測するため，デバイスの使用状況をモニタリングする必要がある．そのため，デバイスドライバのような共通 API にモニタリングのための機構を実装するのが望ましい．また，無線やセンサなどの電源を切ることの可能なデバイスについては，そのデバイスの使用量を計測するためにデバイスの動作状態を取得できる必要がある．この機能についても，既に多くのマイクロユビキタスノードが実装済みである．なぜなら，マイクロユビキタスノードの省電力化のための試みは既に長く続けられており，消費電力削減のために使っていないデバイスの電源を落としたり，短い時間であってもアイドル時にスリープさせるといった機能が搭載されている．そのため，システム側にも必ずデバイスを制御するためのインターフェースが存在するので，その部分にモニタリングのための機能を搭載すればよい．よって，これらの前提条件もシステムのハードウェアに大きな変更を強いるものではなく，現状のマイクロユビキタスノードにも十分に対応可能である．

以上が本システムが動作するための必須環境となる．

3.2 機能要件

次に，2 章で取り上げた問題を解決するための機能要件を洗い出す．pSurvive は，**特定のアプリケーションの動作時間を予約することで，アプリケーションの動作時間保証を実現する**．これにより，予想外のバッテリ切れを回避し，利用者はアプリケーション単位での電力管理が可能となる．また，優先度逆転問題についても優先度の高いアプリケーションの動作時間を予約することで，優先度の低いアプリケーションの影響を回避することができる．

具体的な機能として，以下の機能が必要である．

- アプリケーションの動作時間を予約するための API
- 予約内容に基づいて保証対象のアプリケーションのための電力をキープする機構

一番目は例えば「将来起動するアプリケーション A を N 分間動作させるための電力を予約」や「現在起動しているアプリケーション B を残り M 分間動作させるための電力を予約」といった形式になる．具体的な予約方法については，利用者かアプリケーション自身が pSurvive に対して予約 API を呼び出す形式となる．

二番目は，pSurvive が予約要求された内容に合わせて電力をコントロールする部分となる．しかし，この機能を実現するためには 2.3.1 で取り上げた問題が残る．それは，アプリケーションの動作保証のためにどれだけの電力量を残す必要があるのかという情報の必要性である，現状のシステムでは各アプリケーション別の消費電力量が正確に分からないため，その前提として各アプリケーションの消費電力量を前もって知る必要がある．

これらの問題を解決するため，pSurvive はアプリケーション別消費電力計測・予測機構とアプリケーション動作時間保証機構の二つのコンポーネントで構成される．アプリケーション別消費電力計測・予測機構では各アプリケーションの消費電力量をモニタリングし，今までの電力消費状態を記録すると共に，将来単位時間あたりどのくらいの電力を消費するのかを予測する．アプリケーション動作時間保証機構では，消費電力予測の結果を基に，実際に保証対象アプリケーションのための電力をキープし，必要であれば他の保証対象外アプリケーションを停止させる．

また，機能以外にシステムを普及させるための要件として，次の点も考慮する．

- ノードに追加のハードウェアを実装する必要が無く，ソフトウェアのみで実現されること
- 既存のアプリケーションに修正の必要がないこと

一番目はハードウェアコストや互換性に関する配慮である．計測のための専用のハードウェアを導入する手法を考えてみると，いくつかの点で問題が生じる．まず，既存のマイクロユビキタスノードは本システムの対象から除外されてしまうので，本システムが全く新しいマイクロユビキタスノード上でしか動作しなくなってしまう．次に

ノード設計上の問題として，マイクロユビキタスノードは小型，軽量であることが求められるので，新しくハードウェアを追加するということはそれ自体がハードウェア設計上の制約となる．また部品点数が増えることでノード単価も上昇し，製品の価格にも響いてくることになる．これらの制約を背負った上でもアプリケーションの動作保証を実現したマイクロユビキタスノードを開発するかどうかは製品メーカーに依存するが，これまで行われてきた多機能化に対するコスト増とは違い，見た目上新しい機能が増えるわけではないので費用対効果を考慮すると動機は薄い．一方，ソフトウェアのみで実装することでこうしたハードウェア上の問題は解決される．また，既に広まった製品に対しても付加機能の一つとして追加搭載することができるため，この要件は重要である．

二番目はソフトウェアの互換性に関する配慮である．OS やデバイスドライバ側にシステムを実装することで，従来のアプリケーションを変更することなく pSurvive の恩恵を受けられるようになる．このことは既に多くのソフトウェア資産を有している開発者側にとって，大きな利点となる．OS 側に修正の必要があるということがコスト増に繋がるという考えもあるが，近年のマイクロユビキタスノードでは多くの場合 TRON や RT Linux，Windows Mobile といった汎用組み込み OS を利用しているため，それぞれの OS のために開発したソフトウェア資産は以降の製品にもそのまま受け継いでいくことができる．これにより，一製品あたりの追加コストは全てのアプリケーションを修正していくことに比べると小さくなる．

3.3 基本設計

前節で示した通り，pSurvive はアプリケーション別消費電力計測・予測機構とアプリケーション動作時間保証機構の二つで構成される（図 3.1）．

3.3.1 アプリケーション別消費電力計測・予測機構

本節では pSurvive の根幹であるアプリケーション別消費電力計測・予測機構の設計について解説する．まず pSurvive が用いる消費電力量モデルを示し，その後各変数をどのようにして計測し定義するかを示す．

消費電力量モデル

まず始めに，マイクロユビキタスノードにおいてデバイスの消費電力がどの程度大きいのかを調べるために事前実験を行った．実験環境は 4 章で後述するものと同じマイクロユビキタスノード環境で，各デバイスの使用状態別に消費電力を測定したものが図 3.2 である．縦軸であるバッテリ残量の単位はバッテリ残量を取得した際のレジスタの値を正規化せずに表示している．ここでは電力量の単位ではなくバッテリ残量の低下度合いにのみ着目するので，特に正規化の必要は無いと判断した．

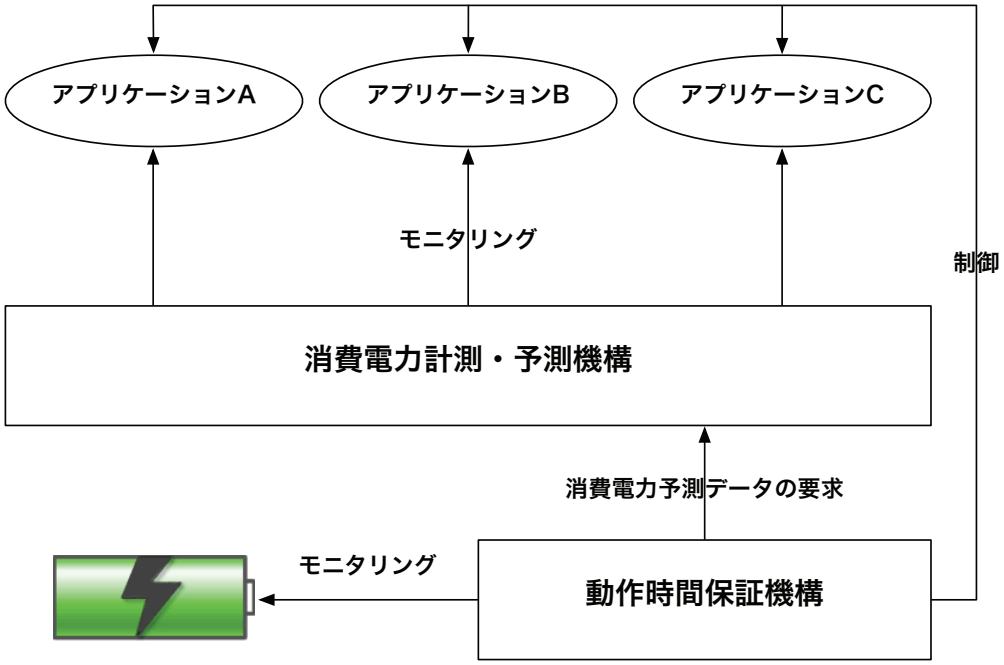

図 3.1: pSurvive システム構成図

この実験ではデバイスの使用電力に着目するために，システム上では特にアプリケーションは起動させず，ノードを起動させる最低限の環境で実験を行っている．それぞれ，Bluetooth とセンサデバイスの二種類のデバイスについて，電源 ON と電源 OFF にした場合の動作時間を計測した．結果，各デバイスの電源を入れた状態と入れていない状態では明らかに動作時間が異なることが分かった．特に，両方のデバイスの電源を ON にした場合では動作時間に倍以上開きがある．なお，バッテリ残量の曲線が 800 を下回った辺りで急激に低下していることがわかるが，これは使用しているリチウムイオンバッテリの特性であると考えられる．この実験により，マイクロユビキタスノードにおいては各デバイスの消費電力がノード全体の消費電力の大きな部分を占めていることが分かる．

実験結果を踏まえて pSurvive では，アプリケーションの使用電力量を各アプリケーション，各デバイス別に計測し，それらの電力の和を全体の消費電力として扱う．まず，ノードが n 個のデバイスを持っていると考える．このデバイスとは，CPU やメモリ，ネットワークやセンサといった電力を消費するハードウェアを指し，各デバイスを $d_1, d_2, d_3, ..., d_n$ とする．そして，各デバイスの単位使用量あたりの消費電力を c [mW/units] とし，$c_1, c_2, c_3, ..., c_n$ とする．これは例えば液晶ディスプレイを 1 分間動作させるのに必要な電力といったものに対応する．この単位使用量の単位 unit とは，デバイスによって定義が異なる．例えば液晶ディスプレイといったデバイスでは使用時間が電力消費量の指標になるが，センサデバイスではセンシングする回数で使用電力が決まるため，時間ではなくセンシング回数となる．

この時，デバイスの消費電力の合計 E はデバイスの使用量を t [units] とすると，$E = ct$ [mW] で表される．そのため，各デバイスの使用時間を $t_1, t_2, t_3, ..., t_n$ とすると，ノー

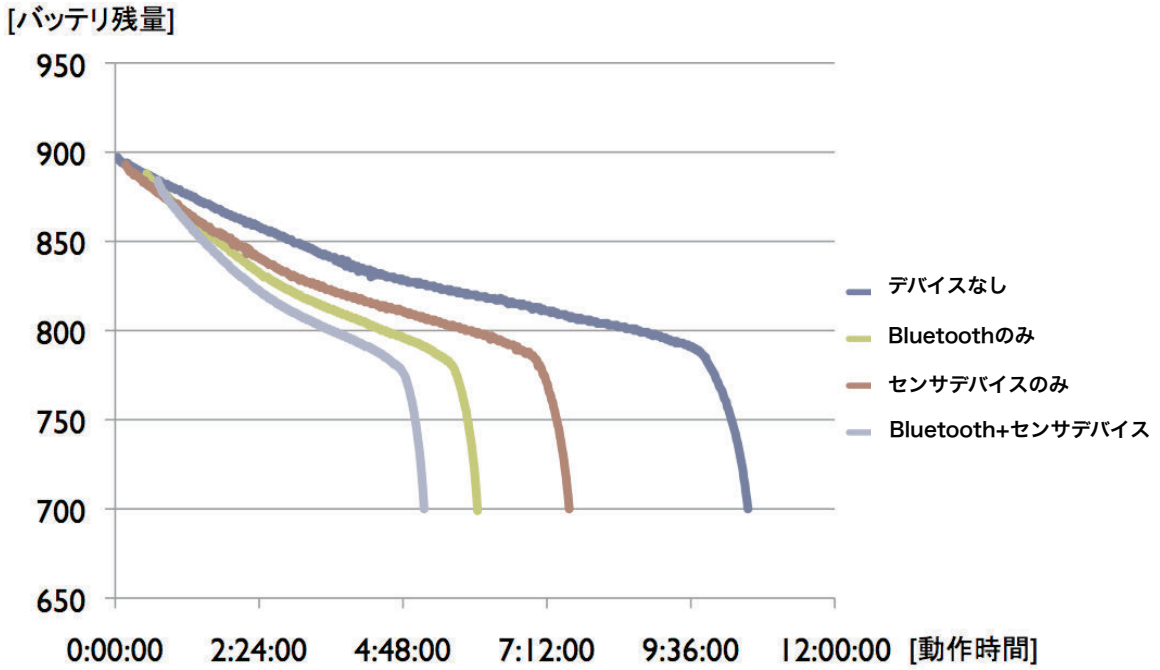

図 3.2: デバイスの使用状態と消費電力量の関係

ド全体の消費電力量は式 3.1 で表される

$$E = \sum_{i=1}^{n} c_i t_i \tag{3.1}$$

次に，ノード上では m 個のアプリケーションが動作しているとし，$p_1, p_2, p_3, ..., p_m$ で表す．このアプリケーションはアプリケーションに対応している．アプリケーションは各デバイスを使用するため，アプリケーションの使った各デバイスの消費電力量の総和がそのアプリケーションの消費電力量であると考えられる．ここで，アプリケーション p がデバイス d を使用した使用量を求める関数 $f(p, d)$ を定義する．すると，アプリケーション p の消費電力 E_p は式 3.2 で表される．

$$E_p = \sum_{i=1}^{n} f(p, d_i) c_i \tag{3.2}$$

ここで，OS や待機電力を特殊なアプリケーションとして抽象化すると，デバイス全体の消費電力は各アプリケーションの消費電力の和となる．そこで，式 3.1 と式 3.2 から，式 3.3 が導き出される．

$$E = \sum_{i=1}^{m} \sum_{j=1}^{n} f(p_i, d_j) c_j \tag{3.3}$$

pSurvive では，この式に従って消費電力を計測する．そのために必要な情報は $f(p, d)$ 及び c，つまりアプリケーションがどのデバイスをどれだけ使っているのかという使用

量と各デバイスの単位使用量あたりの消費電力量である．よって，アプリケーション別の消費電力計測ではこの二つを計測する．

アプリケーション別消費電力量の計測

まず，アプリケーションのデバイス使用量を計測する方法については，アプリケーションはデバイスを使用する際に必ずデバイスドライバを通してアクセスを行うため，デバイスドライバに機能を追加するという手法を用いる．アプリケーションがデバイスを使用する度に使用量を通知するという手法も考えられるが，この場合デバイス上の全てのアプリケーションに修正の必要があるため，現実的ではない．また，CPU やメモリについてはデバイスドライバに相当するものが無いため，OS の統計情報などからデータを取得する．

次に，デバイスの単位使用量あたりの消費電力量 c については前もってキャリブレーションを行い，実測値から求めることで対応する．しかし，キャリブレーションで得た値を静的に利用するのでは動作環境に大きく影響を受けるデバイスについて，実際の値と大きく異なった値になってしまう恐れがある．例えば，無線 LAN を考えた場合，送信についてはアプリケーションが明示的にデバイスドライバを通して送信を行うため計測可能であるが，受信の際の OS オーバヘッドを考えた場合，無線 LAN チップレベルでは自分宛以外のパケットも受信してしまうため，多くの無線 LAN 機器が同じチャネルで通信している環境の場合，消費電力量がキャリブレーション時と異なる値になってしまう．こうした問題に対応するために，システムの動作時には静的な値だけでなく動作環境に応じて c を微調整していく必要がある．微調節の際に用いるアルゴリズムについては，第 4 章で触れる．

将来の消費電力量予測

次に，計測結果から将来の消費電力量を予測する部分について述べる．pSurvive では，消費電力の計測時に消費電力量の合計値だけでなく，どのように電力を消費してきたかの履歴を記録しておくことで，時系列に過去の消費電力履歴を参照する機能を持つ．これにより，消費電力が常に一定とならないアプリケーションについても動的に予測を立てることが可能となる．

3.3.2 アプリケーション動作時間保証機構

本節では，前節の消費電力予測を元に，どのようにしてアプリケーションの動作時間保証を実現するかについて述べる．

保証対象となるアプリケーション

まず，どのようにして保証対象のアプリケーションを決定するかについて解説する．保証対象アプリケーションには，利用者やアプリケーション側から任意に設定可能なものの他に，システムを最低限維持するために必要なものがある．

任意に設定可能なものは，基本的に全てのアプリケーションが対象で，具体的には音楽再生やメール，通話や位置情報通知などのアプリケーションである．これらは利用者が明示的に指定するか，アプリケーション側から保証要求を行うことで保証対象となる．アプリケーション側からの要求とは，サービス提供者側がアプリケーションの一定の動作時間を保証したい場合に行われる．例えば，子供の見守りアプリケーションなどはそのサービスを実現するために最低限 1 日程度の動作を保証する必要があるとすると位置情報を知るための GPS センサ，位置情報を送信するための無線通信，さらに後述するシステムを最低限動作するための電力を 1 日分予約することになる．

システムに最低限必要なものとしては，OS そのものや待機電力といったものが挙げられる．これらはノードの電源を入れている以上必ず必要になるものなので，常に考慮しておく必要がある．そのため，これらはシステム維持のための一つのアプリケーションとして抽象化する．これにより，他の一般アプリケーションと同じ枠組みで消費電力管理が可能となり，システムが単純化される．

予約電力量モデル

まず始めに，保証対象となるアプリケーションを $R_1, R_2, R_3, ..., R_k$ し，それぞれに要求された保証時間を $T_1, T_2, T_3, ..., T_k$ とする．すると，アプリケーション R を T だけ動作するために必要な電力量 E_R は式 3.2 から式 3.4 で表される．

$$E_R = T \sum_{i=1}^{n} f(R, d_i) c_i \tag{3.4}$$

よって，全ての保証対象アプリケーションが必要な電力量 $E_{required}$ は，式 3.3 を用いて式 3.5 で表される．

$$E_{required} = \sum_{i=1}^{k} (T_k \sum_{j=1}^{n} f(R_i, d_j) c_j) \tag{3.5}$$

この式から，pSurvive では $E_{required}$ だけの電力量を保証対象アプリケーションのために残しておくことで，動作時間保証を実現できることが分かる．

動作時間保証の実現方法

pSurvive では，動作保証を実現するために動作保証対象以外のアプリケーションの動作を抑制する．具体的にはバッテリ残量が $E_{required}$ になった時点を動作保証対象以

外のアプリケーションにとっての電池切れと考え，制御を行う．制御の方法にはいくつかの種類が考えられる．

まず最も簡単なものは単純にアプリケーションを終了させるというものである．この手法はアプリケーションにとっては突然終了させられてしまうので柔軟性は無いが，実装が容易である．また，終了するという動作は通常の電池切れの場合と同様であるので，この手法は既存アプリケーションに手を加えずに採用する事が可能である．

次に，終了するのではなくアプリケーションから見た見かけ上の電池残量を少なく見せることにより，アプリケーションに省電力モードなどの機能があればそれを用いるように促すという手法がある．アプリケーションが持つ省電力機能とは，例えば 3.1 で紹介したようなバッテリ残量 API を用いて独自に処理を変更するといったもののことである．この手法は，既にアプリケーション側に省電力のための機構が実装されていれば有効だが，そうでないアプリケーションに対しては新たに実装する必要があるため，実装のための開発コストが必要になる．

最後に，アプリケーションではなく OS 側から各アプリケーションの資源利用を抑制することで，単位時間あたりの消費電力を減らして延命を図るという手法がある．この手法は，アプリケーションに変更を加えることなく適用可能であり，効果が高いように思われるが，マイクロユビキタスノード環境にそのまま適用するには問題がある．それは，マイクロユビキタスノードのような組み込みデバイスで動作するアプリケーションには，リアルタイム処理が含まれていることが多いという問題である．リアルタイム処理の具体例として，動画再生アプリケーションが挙げられる．動画再生はマイクロユビキタスノードが行う処理の中でも負荷が大きく，多くの資源を使用する．そして，コマ落ち無く動画を再生させるためには次のフレームまでの間に次のフレームのための処理を完了しなければならない．こうした期限時間までに仕事を終わらせなければならない処理はリアルタイム処理と呼ばれる．例えば，常時 80 の資源をリアルタイム処理のために必要とするアプリケーションがあったとして，通常時は 100 の資源が割り当てられているとする．ここで，バッテリ残量低下のために OS が資源の割り当てを減らしていった場合，80 までは問題ないが，それ以下になってしまうとアプリケーションはリアルタイム処理を完遂できなくなってしまう．通常マイクロユビキタスノードのような組み込みシステムで採用されるリアルタイム OS では，80 の資源を保証するための枠組みを持っているが，ここに本システムの機構をつけ加える場合，資源管理という仕事について競合が発生してしまう．このように，OS がアプリケーションの資源割り当てを抑制する機構を実装するには，既存のリアルタイム OS の枠組みにまで踏み込んだ議論が必要となり，システムの複雑性が増してしまう．

以上の手法の中から，pSurvive では問題をアプリケーションの動作時間保証に限定するために，単純にアプリケーションを終了させる手法を選んだ．資源の効率的な利用という観点からはその他の手法を選ぶ方が望ましいが，そうした議論は本研究の課題として最終章で取り上げる．

3.4　本章のまとめ

本章ではマイクロユビキタスノードにおけるアプリケーション動作保証機構の設計について示し，具体的な想定環境と機能要件についてまとめた．その中で，pSurviveでは既存のマイクロユビキタスノードに大きな変更を加えることが無く実装できることを示した．その後，アプリケーション別消費電力計測・予測機構とアプリケーション動作時間保証機構の実現方法について解説した．

第4章

実装

本章では，pSurviveの具体的な実装について述べる．まず最初に動作環境と実装環境について詳しく述べる．その後，ソフトウェア構成についてアプリケーション別消費電力計測・予測機構とアプリケーション別動作時間保証機構の二つについて詳しく解説していく．最後に本章のまとめを行う．

4.1 実装環境及び動作環境

まず，pSurvive の実装にあたって選定した実装環境及び動作環境を，ハードウェア，ソフトウェア両方の面から述べる．pSurvive の実装にあたり，3 章で述べた前提条件に加えて必要な要件が四つある．

1. OS に手を加えることが可能で，開発環境が利用できること
2. 実際にマイクロユビキタスノードでの採用例があるか，製品としてすでに販売されていること
3. 本システムの有効性を示すのに十分な評価が取れるだけの機能を有すること
4. 特殊なハードウェアを極力用いず，本システムの汎用性を示せること

1 番目は，アプリケーションの各デバイス使用量モニタリングを OS レベルで実装する必要があるので必要である．2 番目の要件については，実際の組み込みシステムでの採用例があることで，実験結果の信憑性が高まる．3 番目について，細かな評価を行えるよう，既存のマイクロユビキタスノードと同等かそれ以上の機能を持っていることが望ましい．最後の要件は，ハードウェア依存の実装になりがちな組み込みシステムの中で，なるべく特定のハードウェアに依存するところを無くし，実験環境以外のマイクロユビキタスノードでも同じことが言えるということを保証するために必要である．

4.1.1 ハードウェアの選定

本論文では，前述した要件を満たすマイクロユビキタスノードとして，Gumstix Vertex XM4-BT（図 4.1）及び NUTS センサボード（図 4.2）を用いる．

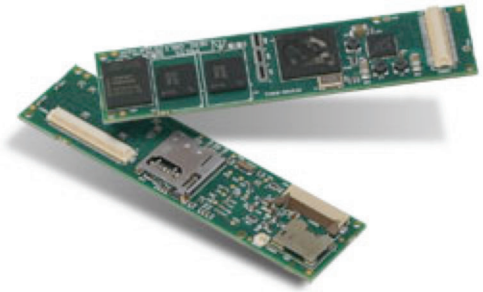

図 4.1: Gumstix Vertex XM4-BT

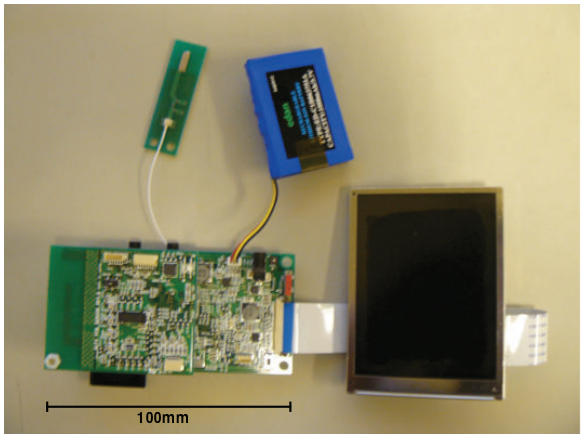

図 4.2: NUTS センサボード

Gumstix Vertex XM4-BT は，Gumstix 社が販売している ARM CPU を搭載した組み込みハードウェア製品である．大きさは板ガムサイズと小型であり，Marvell 社の Xscale PXA270 を CPU として搭載している．Xscale は PDA などで広く使用されている組み込み用 CPU であり，マイクロユビキタスノードとしての利用実績が豊富である．また，広く普及している ARM プロセッサであるため，既に多くのアプリケーションが公開されていることも選定の理由である．メモリは 64MB の RAM と 16MB のデータ保存用 Flash メモリを搭載しており，拡張コネクタを用いることで SD/MMC カードをストレージとして用いることもできる．さらに，本体に Bluetooth 通信機能が搭載されており，単体で無線通信が可能である．2 章で述べた通りネットワーク機能はマイクロユビキタスノードの多くで実装されているため欠かせないデバイスである．その他，詳しいハードウェア仕様を表 4.1 に示す．

表 4.1: Gumstix Vertex XM4-BT の仕様

CPU	Marvell PXA 270 with XScale, 400MHz
RAM	64MB
Flash	16MB
拡張コネクタ	60Pin ヒロセコネクタ, 80Pin コネクタ，24Pin Flex コネクタ
大きさ	80mm x 20mm x 6.3mm
重さ	8g
動作電圧	DC 3.6V - 5.0V
バッテリ	edan ED-C1B063751A, 1000mAh/3.7V

NUTS センサボードは，株式会社イーアールアイによって開発された，Gumstix Vertex の拡張コネクタに接続して使用できる独自開発されたセンサボードである．Gumstix の拡張デバイスとして働き，音声入出力，照度，加速度センサ，GPS，ロータリーエンコーダといったセンサや液晶ディスプレイを搭載できる．また，USB ポートを備え

ているため，無線 LAN などのデバイスを拡張することもできる．NUTS センサボードの大きな特徴として，各デバイスの電力供給状態を細かに制御できる PMU（Power Management Unit）を備えていることが挙げられる．PMU は Gumstix 側の CPU とは独立して動作する電源管理専用のチップであり，前述した NUTS 上のデバイスへの電源供給を任意に ON/OFF することができると共に，デバイスの電源供給状態を知ることができる．また，NUTS センサボードはリチウムイオンバッテリを電源として用い，NUTS 上のデバイスと Gumstix への電源共有を行うが，PMU を通してバッテリ残量を 32bit 単位で細かく取得することができる．これにより，評価に必要な高精度のバッテリ残量を得ることができる．

本論文では Gumstix と NUTS センサボードを併用することで，Gumstix の持つ汎用性の高さと NUTS センサボードの持つ細かな電源制御を備えた実験環境として採用した．

4.1.2　ソフトウェアの選定

組み込みシステムにおけるソフトウェアの選定はハードウェアに依存するところが大きい．Gumstix は組み込みシステムとして一般的な ARM プロセッサを搭載しているため，既存の組み込み環境の多くが動作するが，本論文では前述の要件を満たす実装環境及び動作環境として，OpenEmbedded を選択した [17]．

OpenEmbedded は組み込み向け Linux ディストリビューションの一つで，多くの組み込みプラットフォームに対応している．特徴として，開発のためのクロスコンパイル環境を全て含んでおり，一貫した開発環境を手軽に揃えられることと，Bitbake というパッケージ管理ツールによりソフトウェアの新規追加や削除，また依存性のあるソフトウェアのインストールなどを自動管理できるという点がある．実験時の Linux Kernel バージョンは 2.6.21 であり，基本ソフトウェア群は OpenEmbedded 標準のソフトウェアパッケージ gumstix-custom-vertex をベースに開発を行った．

また，クロスコンパイル環境としては Fedora Core 8 を選択した．クロスコンパイル環境は OpenEmbedded 環境がインストールできる環境であれば特に環境を選ばない．図 4.3 はクロスコンパイル環境におけるハードウェア構成図である．クロスコンパイル環境では資源上の制約の多い Gumstix 上でのプログラム開発は行わず，開発用 PC でソフトウェア開発を行い，Gumstix 用のプログラムのコンパイルを行う．その後，コンパイル済みのイメージを Gumstix へ転送することで，Gumstix 側でのプログラム実行が可能となる．

4.2　ソフトウェア構成

本節では具体的な実装の詳細について述べる．

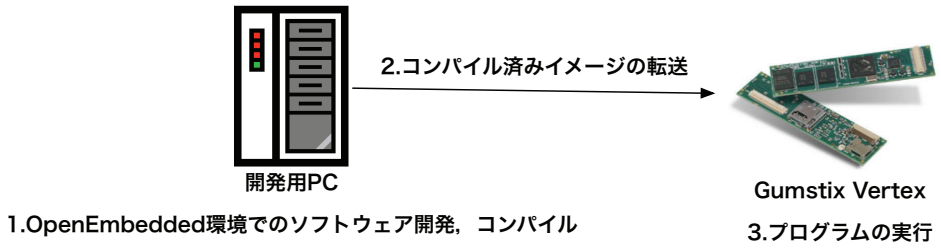

図 4.3: クロスコンパイル環境

4.2.1 アプリケーション別消費電力計測・予測機構

pSurvive の実装では，消費電力計測・予測機構をデバイス毎に実装するモニタリングモジュールとデータ集約モジュール，消費電力予測モジュールの三種類に分割した．その構成を表したものが図 4.4 である．

それぞれのモジュールは動作レベルでは互いに独立しており，それぞれ別途開発が可能である．このようにモジュールを分割することで，各部の機能追加や交換を可能とし，拡張性を高めた．

モニタリングモジュール

モニタリングモジュールは各デバイスに依存するモジュールで，基本的にデバイスと一対一で対応する．モニタリングモジュールが最低限持つ機能はデバイスの使用量をカウントする機能で，後述するデータ集約モジュール内のアグリゲータからの要求に応じてアプリケーション動作中のデバイス使用量をカウントする．デバイス使用量は 3 章で解説した通り，CPU なら使用時間，ネットワークであればパケット数といったものが単位となる．モニタリングモジュールはデバイスに対応するものであり，特にマイクロユビキタスノード自体に依存するものではないので，同じデバイスを使っているノードであればそのまま再利用できる．

具体的な実装方法はデバイスに大きく依存するところがあるが，基本的には Linux Kernel Module（LKM）を用いて実装することを想定している．LKM を用いて実装する理由はデバイスとアプリケーションの対応を取るためである．アプリケーションレベルで実装してしまうと，デバイスの使用量を取得することはできるが，その消費量がどのアプリケーションに関連付いているかを取得することができない．なぜならば，モニタリングする時点で既にスケジューラはモニタリングアプリケーションに処理を渡してしまっているので，実際にデバイスを使用したアプリケーションが限定できないからである．

LKM によって実装することでこの問題を解決する．LKM によってデバイスのモニタリングを実装することで，コンテキストスイッチ無しにデバイスの使用量を監視することができる．これにより，デバイス使用時にスケジューリングを割り当てられているプロセスが実際にデバイスを使用したアプリケーションとなるため，デバイスと

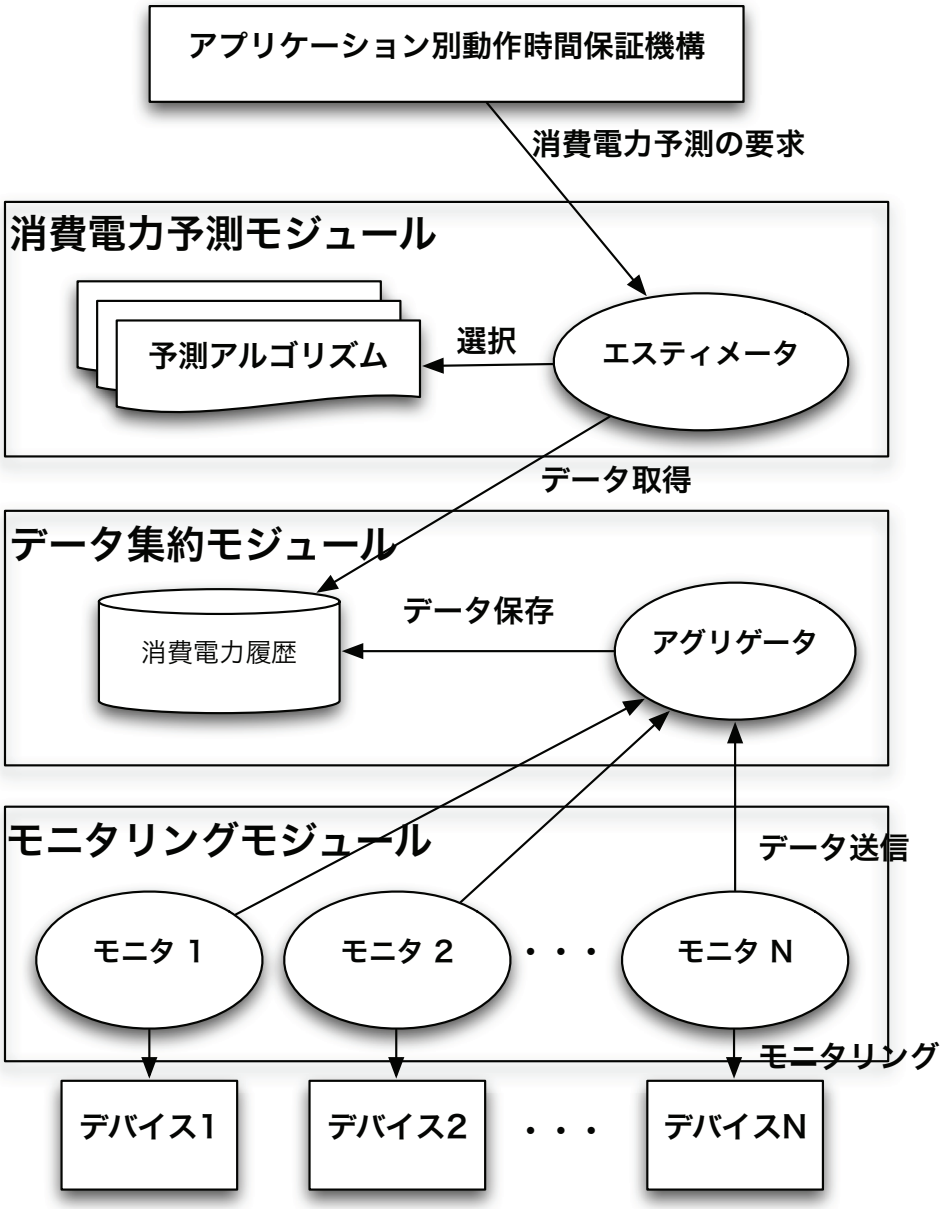

図 4.4: アプリケーション別消費電力計測・予測機構の構成

アプリケーションの対応を取ることができる．

データ集約モジュール

データ集約モジュールは，定期的にモニタリングモジュールに対してデバイスの消費量を取得するアグリゲータと，取得したデータを時系列で保存する消費電力履歴データで構成される．

アグリゲータは特定の周期でモニタリングモジュールに各アプリケーションのデバイス消費量を問い合わせ，消費電力履歴として保存する．モニタリングの周期を短くすることで，よりアプリケーションの詳細なデータが取れるが，その一方でデータ集約モジュールのオーバヘッドは増加する．しかし，このモニタリング周期の決定には，バッテリの特性が大きく関わってくる．バッテリ残量の解像度が細かい場合は頻繁なサンプリングを行うことで詳細なデータを取ることができるが，解像度が低い場合に頻繁なセンシングを行った場合，連続して同じ値が帰ってきてしまうことが考えられる．今回の実装環境であるリチウムイオンバッテリと NUTS センサボードの組み合わせにおいては，5 秒〜30 秒程度のセンシング周期では値がほとんど変化しなかったため，1 分という周期を採用した．

データ集約モジュールは LKM によるカーネルレベルでの実装ではなく通常のアプリケーションとして実装する．その理由はデータ集約モジュール自体も電力を消費するため，オーバヘッドを考慮する必要があるからである．LKM による実装を行った場合，pSurvive の消費電力は OS のオーバヘッドに含まれることになるが，pSurvive の評価を行う段階で pSurvive を搭載することによるオーバヘッドを測定することができない．また，LKM による実装では利用できる OS の機能が著しく限定されてしまうため，消費電力履歴の保存先の変更といった柔軟な対応が難しくなるということも理由の一つである．

消費電力予測モジュール

消費電力予測モジュールは，アプリケーション別動作時間保証機構からの要求に応じて，データ集約モジュールが保存した消費電力履歴を元に消費電力予測を行うエスティメータと，その予測アルゴリズムで構成される．

消費電力予測は時系列のデータから未来の消費電力を予測するものであるが，必ずしも全てのアプリケーションに対して一つのアルゴリズムが精度の高い予測結果を返すとは限らない．例えば，ほぼ一定周期で動作する GPS 位置情報通知のようなアプリケーションであれば，過去のデータからの移動平均といった値が合致すると考えられるが，Web ブラウザのような利用者の使用や環境によって電力消費の傾向が大きく変わるアプリケーションの場合，移動平均だけでは精度の高い予測は難しい．そのため，消費電力予測モジュールでは複数のアルゴリズムを用意し，アプリケーションに応じた予測アルゴリズムを選択可能としておく．これにより，アプリケーション毎に柔軟な予測を行うことができ，予測精度が高まる．本論文における実装では，2 種類の予測

アルゴリズムを用意する．一つは，消費電力モジュールが直近に計測したのと同じ量の消費電力を使うという単純なモデルである．このモデルは常に電力消費が一定になるようなアプリケーションに最も適合すると考えられる．もう一つは移動平均アルゴリズムである．但し，パラメータとして参照する履歴の幅を指定できる様にする．履歴の幅を短く設定すると，急激な値変動に対して敏感に予測値が変動するため，より早くアプリケーションの消費電力変動に対応できる一方で，一時的な値変動の影響を大きく受けてしまう．これは電力を消費するときにある程度長い期間，急激に消費するようなアプリケーションが適合するだろう．また，履歴の幅を長く設定することで，一時的な値変動の影響は小さくなるが，値変動後に変動後の値が続くような場合には予測値が理想値に近づくのが遅くなる．このアルゴリズムは短い時間に一時的に消費電力が大きくなる様なアプリケーションが適合するだろう．

消費電力予測モジュールもデータ集約モジュールと同じく通常のアプリケーションとして実装する．その理由もデータ集約モジュールの場合と同じく，拡張性のためである．特に予測アルゴリズムを容易に追加，交換可能とすることは pSurvive の汎用性を高めるために重要な点である．また，予測アルゴリズムを誰がどのようにして選択するのかといった課題が残るが，この問題は 7 章で今後の課題として議論する．

4.2.2 アプリケーション別動作時間保証機構

アプリケーション別動作時間保証機構では，アプリケーション別消費電力計測・予測機構からの予測結果を基に動作保証外アプリケーションの停止処理を行う．本機構の内部構成は図 4.5 の通りである．

本機構はバッテリ監視モジュールと動作保証モジュールの二つで構成され，前述のアプリケーション別消費電力計測・予測機構の結果を利用する．

バッテリ監視モジュール

バッテリ監視モジュールは，バッテリ残量を定期的に監視し，動作保証モジュールへ通知する．本モジュールの役割は，動作保証に必要な電力が残っているかを後述の動作保証モジュールへ伝えることである．バッテリ残量低下時の動作については動作保証モジュールが行うので，本モジュールはバッテリの監視のみを行う．

本モジュールを分離する意図は，マイクロユビキタスノード毎の違いを吸収することにある．バッテリ残量を取得する方法は各ノードの種類によって大きく異なるため，このモジュールを分離しておくことで，pSurvive のノード依存度を下げることができる．結果としてバッテリ監視モジュールは各ノード専用に実装する必要があるが，本機能は単純な機能のみを有するので開発のためのコストは小さい．

本実験環境である NUTS センサノードの場合，バッテリ残量は PMU デバイスドライバから sys ファイルシステムを通して取得することができる．具体的には/sys/devices/platform/pxa2xx-i2c.0/i2c-0/ 0-0048/battery/battery_voltage に対して通常通りファ

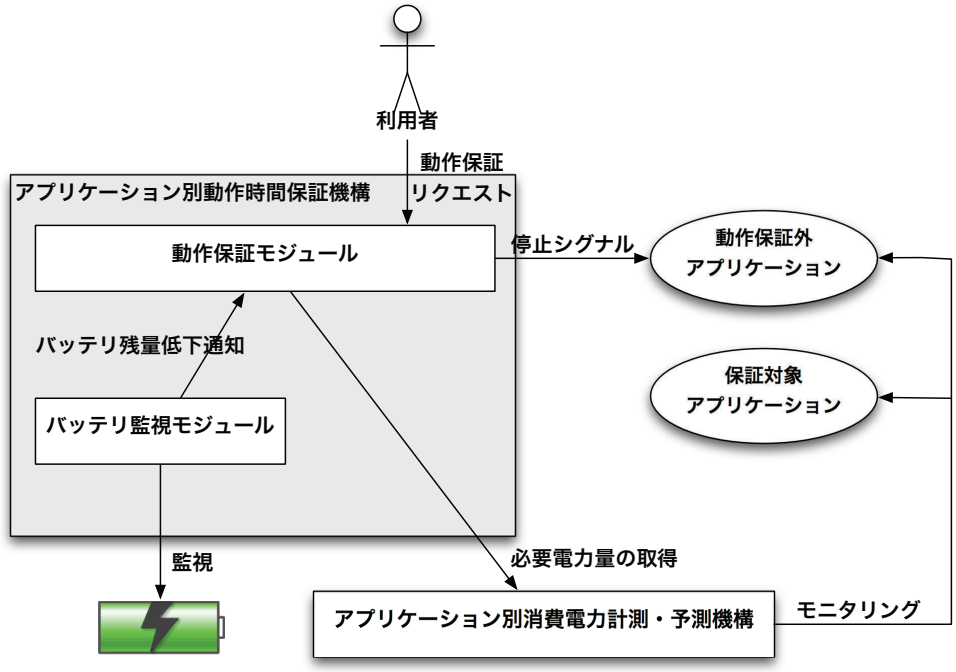

図 4.5: アプリケーション別動作時間保証機構の構成

イルとしてアクセスすることで，PMU のレジスタからバッテリ電圧を取得する．この様に，標準のファイルシステムインタフェースからバッテリ残量が取得できるため，本モジュールはアプリケーションとして実装する．

動作保証モジュール

動作保証モジュールは利用者からの動作保証リクエストを受け取り，どのアプリケーションをどれだけの時間，動作保証するのかを保持する．また，前述のアプリケーション別消費電力計測・予測機構の消費電力予測モジュールに対し，保証対象アプリケーションの必要電力量 $E_{required}$ を問い合わせることで，どれだけの電力量を確保すべきなのかを取得する．そして，バッテリ残量の低下がバッテリ監視モジュールから通知された際に残りバッテリ量が $E_{required}$ に近ければ，動作保証外アプリケーションを終了させる．また，残りバッテリ量が $E_{required}$ を下回り，動作保証外アプリケーションを動作させる余裕が無いときには，起動された動作保証外アプリケーションを終了させる役割も担う．

本実験環境では OS として Linux を利用しているため，アプリケーションの停止は KILL シグナルを送ることで実現する．

4.3 本章のまとめ

本章では，pSurvive の具体的な動作環境と実装について述べた．実験環境として本論文では Gumstix と NUTS センサボードを用い，Linux 環境を採用した．また，アプリケーション別消費電力計測・予測機構とアプリケーション別動作時間保証機構はそれぞれモジュールに機能分割されており，システムの可搬性を高めていることを示した．

第5章

実験と評価

本章では，pSurviveの実験と評価について述べる．評価はアプリケーション別消費電力計測・予測機構とアプリケーション別動作時間保証機構の各機能別のものと，システム全体の評価について行い，pSurviveの有効性について考察する．最後にまとめを行う．

5.1　実験環境の設定

まず最初に，実験環境について解説する．図 3.2 のバッテリ減衰傾向を見て分かるように，リチウムイオンバッテリは常に線形な電圧低下を行うわけではない．そのため，本実験ではバッテリ特性による測定結果への影響を避けるため，減衰傾向が線形となる電圧区間のみを有効な残バッテリ量として扱う．また，いくつかのバッテリを用いて実験したところ，同じ製品でも固体によって満充電時の電圧値に開きがあることがわかった．こうした個体差が測定結果に影響を及ぼさないようにするため，一連の実験は全て一つの同じ端末において，同じバッテリを用いて行った．

5.2　デバイスの単位時間あたり消費電力の計測

無線 LAN 機能の消費電力は「デバイスの待機電力＋通信にかかった電力」で表される．デバイスの待機電力は，デバイスの電源を入れているだけで消費するものであり，ネットワーク通信のない最低動作環境のシステムで，無線 LAN を ON にした場合と OFF にした場合の消費電力を見ることで測定できる．実際の測定した結果が図 5.1 である．縦軸は NUTS センサボードの PMU から取得された正規化前のバッテリ残量レジスタの値である．

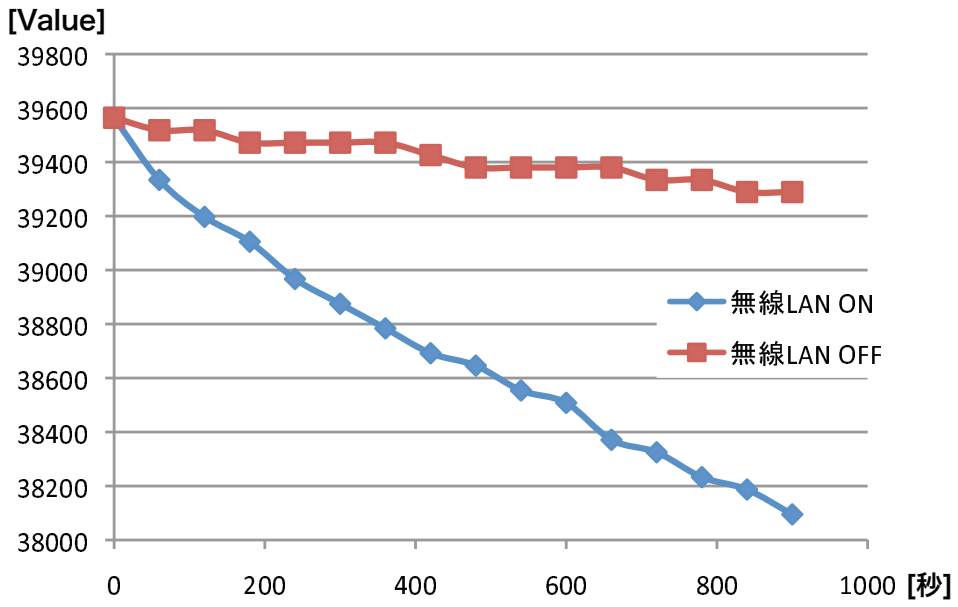

図 5.1: 無線 LAN の ON/OFF による電力消費量の変化

この結果から，無線 LAN デバイス起動時の待機電力は 15 分辺り 1,194，即ち 1 分辺り 79.6 であることが分かった．次に，無線 LAN での通信にかかる電力を測定するために，最低動作環境で無線 LAN を ON にし，通信プログラムを起動して通信を発生させる．通信プログラムは，255 バイトの UDP パケットを送信し続けるというもの

で，このプログラムによる送信パケット数をカウントする．データ転送プログラムには Flash への書き込みによる電力消費などを押さえるため，適当なデータを生成してそのまま転送する自作の通信プログラムを利用した．転送したデータはそのまま破棄し，Flash などへの書き込みは行わない．こうして測定した結果が図 5.2 である．

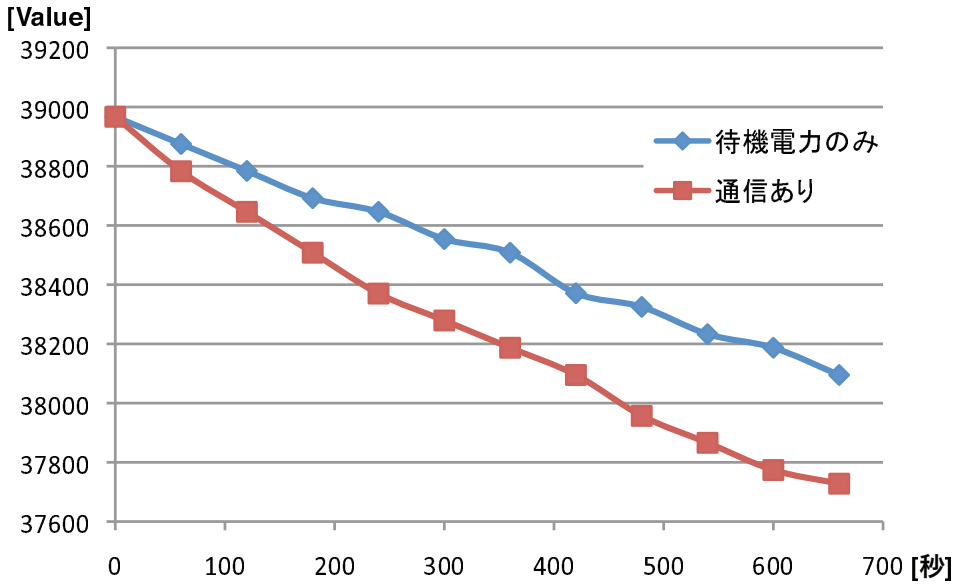

図 5.2: 通信の有無による電力消費量の変化

この実験において，11 分間に送受信したパケット数は 612,951 で，通信有りの場合と無しの場合の消費電力の差は 367 である．この結果から，255 バイトの UDP パケット 1,000 パケットの送信辺り 0.599 の電力を消費することが分かった．

次に，CPU の消費電力を計測する．Linux においてプロセスに対する CPU の割り当て時間は標準で取得されており，スケジューラなどから利用されているので，本実験においてもこの値を取得する．CPU 使用量はプロセスを表す task_struct 構造体に記録されており，proc ファイルシステムを通して参照することができる．具体的には/proc/[プロセス番号]/stat の中にある utime と stime の値を参照する．utime はプロセスがユーザモードで動作している実行時間であり，単位は jiffies というカーネルの内部単位時間を用いる．stime はプロセスがカーネルモードで動作している実行時間であり，こちらも単位は jiffies である．

ここで，CPU 資源だけを消費するアプリケーションとして，円周率計算のプログラムを用意し，バッテリ残量の変化を記録した結果が図 5.3 である．

この実験の中でこのプロセスが消費した CPU 時間は 71,925 であり，CPU 負荷による消費電力の差分は 826 であった．ここから，CPU 時間を 1,000 消費する毎に 11.484 の電力を消費することが分かった．

以上の測定結果から，本実験環境における各デバイスの単位使用量辺りの消費電力を表 5.1 にまとめた．

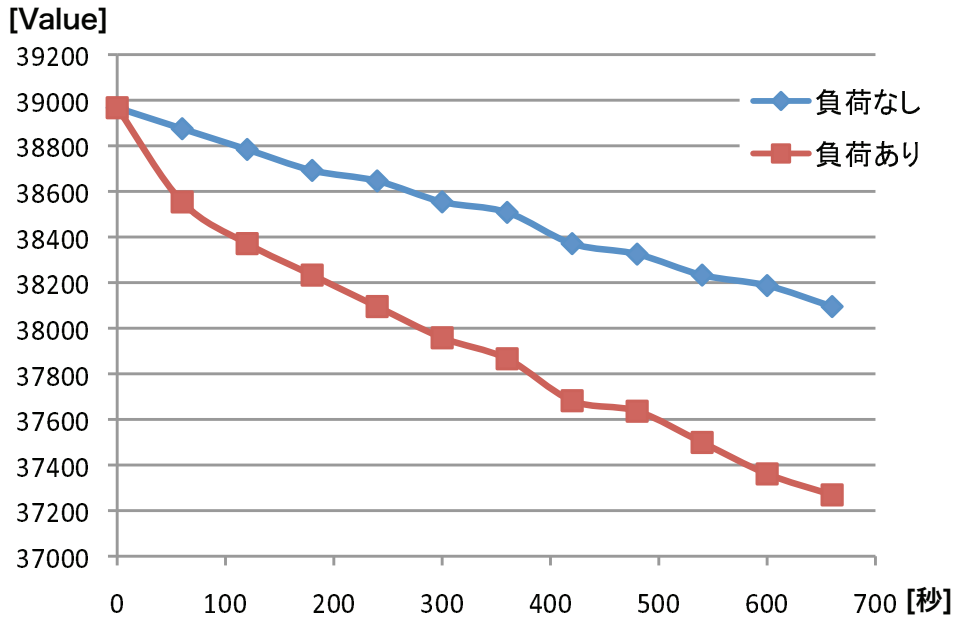

図 5.3: CPU 負荷の有無による電力消費量の変化

表 5.1: 各デバイスの単位使用量辺りの消費電力

	消費電力量	1 単位使用量
システムの最低限動作	18.3	60 秒
無線 LAN 待機電力	79.6	60 秒
無線 LAN 送信電力	0.599	1,000 パケット
CPU 負荷	11.484	1,000 jiffies

5.3 アプリケーション別消費電力計測機構の評価

次に，計測した単位使用量を元にして，計算で求めた消費電力と実際の消費電力を比較することで，消費電力計測の精度を求める．前節のネットワーク計測のプログラムと円周率計算の両方の処理を行うプログラムを作成し，二つの処理を同時に行った場合の CPU とネットワークの使用量を計測する．この時得た 12 分間分の計測結果が表 5.2 である．

この時のノード全体の予想電力使用量 $E_{estimated}$ は各デバイスの消費電力の和であるから，表 5.1 を用いて次のように計算できる．

$$E_{estimated} = \frac{\sum \text{送信パケット数}}{1000} \times 0.599 + \frac{\sum \text{CPU 使用 jiffies}}{1000} \times 11.484 + 12 \times 79.6 + 12 \times 18.3$$

表 5.2: CPU とネットワークを使用するアプリケーションの資源使用量

秒	残バッテリ量	送信パケット数	CPU 使用 jiffies
0〜60	39197	55180	2335
60〜120	38921	55619	2414
120〜180	38600	55110	2396
180〜240	38508	55368	2444
240〜300	38325	55732	2476
300〜360	38233	55165	2518
360〜420	38187	55463	2471
420〜480	37866	55596	2468
480〜540	37728	54889	2523
540〜600	37636	55586	2420
600〜660	37407	55128	2509

$$
\begin{aligned}
&= \frac{663823}{1000} \times 0.599 + \frac{29470}{1000} \times 11.484 + 12 \times 79.6 + 12 \times 18.3 \\
&= 1910.863
\end{aligned}
$$

一方，バッテリから取得できる消費電力量の実測値 E_{real} は以下のように計算できる．

$$
\begin{aligned}
E_{real} &= 39197 - 37407 \\
&= 1790
\end{aligned}
$$

ここで，$E_{estimated}$ と E_{real} の差が 120.683 あり，これが誤差である．誤差率は次のように計算され，

$$
\frac{|E_{real} - E_{estimated}|}{E_{real}} \times 100 = 6.74
$$

結果，6.74%となる．この誤差について考察するため，予測値と実測値がどのようにずれていったのかを図にしたものが 5.4 である．縦軸は時刻 t までの消費電力量の合計を表す．

グラフを見ると，実測値が大きく上下にぶれていることが分かる．特に，400 秒近辺では 1 分間値が変わらないこともあり，ばらつきを見せている．この実測値のぶれはなぜ起こるのかを考察すると，リチウムイオンバッテリの電圧値自体が安定していない可能性が考えられる．システムの消費電力は長いスパンで見れば平均的に消費されていても，ある瞬間だけを見ると高い負荷がかかっていたり，逆に低い負荷になったりするため，バッテリ残量のセンシングのタイミングによってはいくらかのばらつきが発生する可能性がある．こうした細かなぶれを回避するには，もっと長いスパンでのセンシングを行い，十分な量のデータを集めるという対策が考えられる．

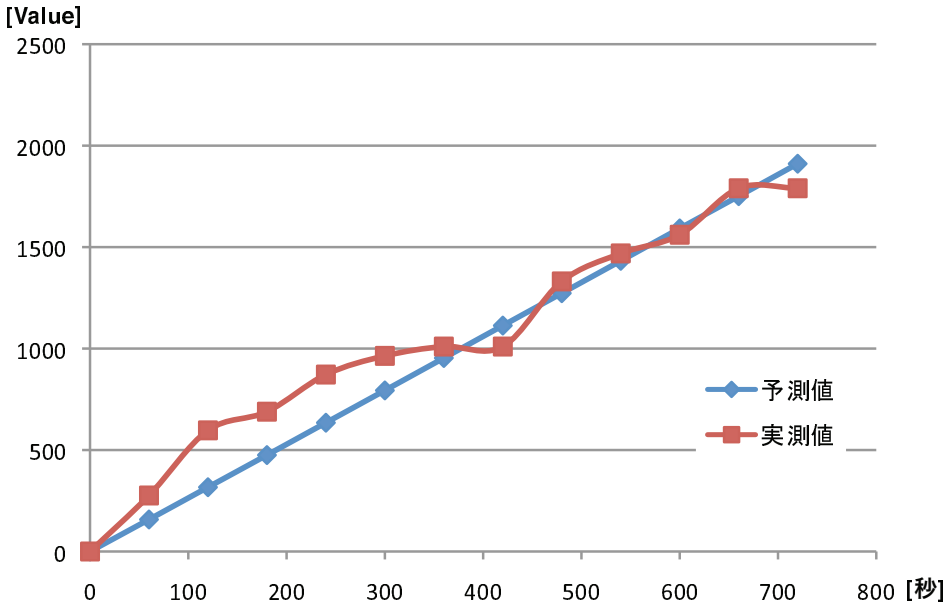

図 5.4: 予測値と実測値の変化

5.4 本章のまとめ

本章では pSurvive の実装を行い，実験と評価を行った．その結果，消費電力計測機構による消費電力量の理論値は，実測値に比べて 6.74% の誤差率の値であることが分かった．

第6章

関連研究

本章では，本研究の関連研究について述べる．関連研究は，資源管理という面での研究，及び省電力に関する研究に分けて紹介する．最後に，それらの関連研究と本研究との違いと優位性についてまとめる．

6.1 資源管理に関する研究

pSurvive は各デバイスという資源を監視することで消費電力を計測し，そこから予測した情報を元にアプリケーションを制御するが，こうした優先度に応じてアプリケーションのための資源を確保するといった手法はリアルタイム OS の分野などで多く研究されているため，まずはリアルタイム処理に関して本研究との関連を述べる．次に，既存の資源予約機構について紹介し，本研究との違いについて解説する．

6.1.1 リアルタイム処理

Clifford W. Mercer は，リアルタイム OS に関する代表的な事項をまとめている [12]．その中で紹介されている評価関数を用いたハードリアルタイムやソフトリアルタイムの評価手法は，本研究の評価でも参考にしているので，ここで紹介する．

リアルタイム OS の評価関数は，あるプロセスが時刻 R に起動し，時刻 D までに処理を完了することが要求されるとき，実際に処理が完了する時刻とその仕事に対する評価を関数にしたものである．リアルタイム OS における評価関数の種類は大きく分けて 3 種類あり，ソフトリアルタイム，ハードリアルタイム，破滅的なハードリアルタイムの場合がある．昨今の組み込みシステムでは，これらのうちハードリアルタイムをソフトリアルタイム，破滅的なハードリアルタイムを単にハードリアルタイムと呼称することが多いが，本論文では Clifford の紹介している呼称を用いる．

まず一つめはソフトリアルタイムで，評価関数は図 6.1 の形となる．横軸は時間の経過を示し，縦軸は時刻 t で処理を完了した際の評価関数の値であり，評価関数の値が大きいほど仕事を達成した割合が大きい．

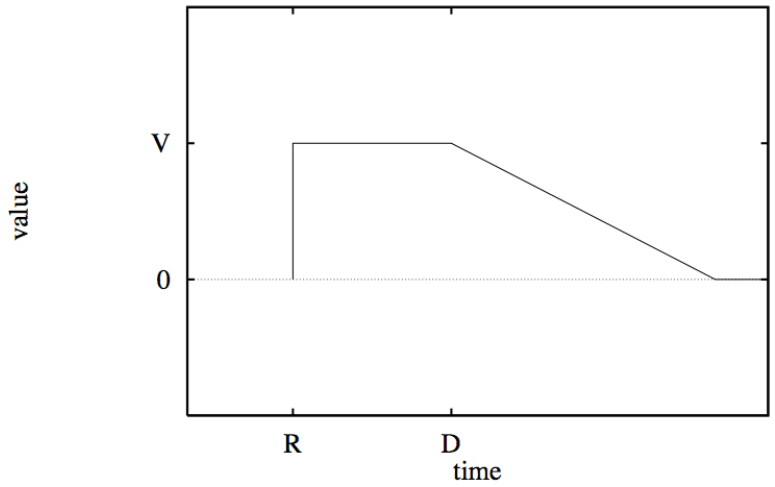

図 6.1: ソフトリアルタイムの評価関数

ソフトリアルタイムは，プロセスが要求された時刻 D までに仕事を完了しなかった場合でも，達成度合いは少なくなるが致命的な問題にはならない例である．これは，な

るべくデッドラインまでに処理を完了して欲しいという緩やかなリアルタイム処理である．例えば通話や音楽再生などが対応し，仮に音声の伝送や音楽の再生が一時的に詰まっても，システムや人命には致命的な問題を起こさない．

二つめはハードリアルタイムで，評価関数は図 6.2 の形となる．

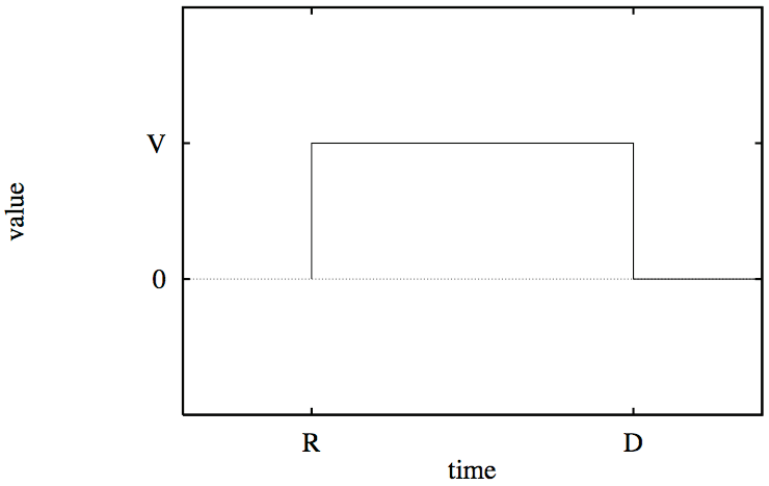

図 6.2: ハードリアルタイムの評価関数

ハードリアルタイムでは，プロセスが時刻 D までに仕事を完了できなかった場合には，それ以上仕事を続けることに意味が無く，それまでに行った仕事は無駄になってしまう例である．ソフトリアルタイムに比べて時間要求が厳しく，組み込み機器ではこうした要求が多く存在する．しかし，この評価関数の場合では，仮に仕事を完了できなくてもその処理は無視されるだけである．

最後は破滅的なハードリアルタイム（catastrophic hard realtime）で，評価関数は図 6.3 の形となる．

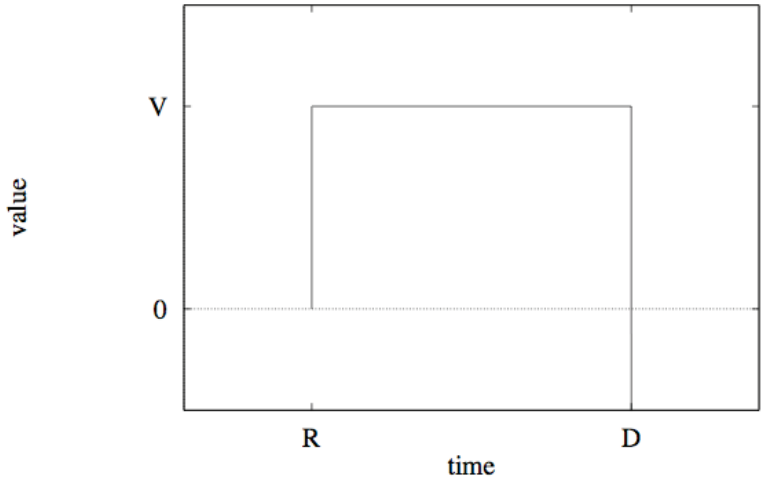

図 6.3: 破滅的なハードリアルタイムの評価関数

破滅的なハードリアルタイムは，プロセスが時刻 D までに仕事を完了できなかった場合にシステム自体がクラッシュしてしまったり，人命を危険に脅かしたりする致命的な処理を扱う例である．その為，評価関数上では時刻 D を満たせなかった時点で値が $-\infty$ となる．具体的な例としては車のブレーキングシステムなどが該当し，破滅的なハードリアルタイムシステムは，リアルタイムシステムに対する要求の中でも最も厳しいものである．

本研究におけるアプリケーション動作保証の評価もこのような評価関数による評価ができるが，リアルタイム処理が時刻 D までに処理を完了することが目的であることに対して，時刻 D よりも長く処理を行うことを目的とする，対称となる目的を持っていることが異なる．また，リアルタイム処理では CPU やネットワークといった資源を割り当てるが，本研究ではバッテリという単調減少する資源を扱うという点が異なる．それに関連してリアルタイム処理では数 ms〜数十 ms という短い制限時間のデッドラインを扱うが，本研究で扱うバッテリ管理ではそれに比べて比較的長い数分〜数十分単位での電力消費を扱う．この様に前提となる環境が大きく異なるため，リアルタイム処理に関する技術をそのままマイクロユビキタスノードにおける電源管理に適用するのには無理がある．

6.1.2 資源予約

西尾らの提唱する iReserve アーキテクチャは，メディアコンテンツの配信などにおける QoS（Quality of Service）を実現するためのミドルウェアである [14][15]．QoS とはサービス品質を保証する技術であり，メディアコンテンツの配信においては音飛びやコマ落ちの無い高品質なサービスを提供するために用いられる．

iReserve では既存の QoS システムが動作対象のハードウェアプラットフォームやソフトウェア環境に大きく依存しているために，アプリケーションプログラマがアプリケーションを実装する際に，その環境に応じたコーディングが要求されてしまうことを問題として，共通の資源予約機構を実現した．具体的には，要求される品質を抽象化されたユーザレベル，具体的なシステムレベル，そしてその間のミドルウェアレベルに階層を分けることで実現している．これにより，プログラム側からは良い，悪いといった抽象的なレベルで品質を指定することができる．ミドルウェアレベルでは 24fps で 22KHz といったメディアコンテンツの品質が定義され，システムレベルではそのハードウェア上で CPU やネットワークを何パーセント使用するのかといった具体的な要求が定義される．このように階層を分けることで，ハードウェア依存の部分が明確に分かれ，アプリケーションプログラマの負担やソフトウェア開発コストの減少を図ることができる．本研究が提唱する pSurvive でも，資源予約のためのモジュールの設計時にハードウェア依存部を明確に分離しており，アプリケーションプログラマへの負担を軽減することを考慮している．

しかし，資源管理について iReserve は CPU とネットワーク資源のみを予約するのに対し，pSurvive ではバッテリ容量を予約する．CPU やネットワークといった資源は，

システムが起動中に常に一定量供給されるものをどこに割り当てるかというものであり，時系列に処理できる量が減少するといったことはない．一方バッテリは有限の資源であり，時系列で減少していくという特殊な資源であるため，既存の QoS システムをそのままバッテリ管理に適用するのは難しい．

6.2 省電力に関する研究

本節では，バッテリ管理に関連して省電力に関する研究について触れる．

6.2.1 プロセス毎の資源管理

宮川らの提案する POPM は，昨今の CPU が備える DVFS（Dynamic Voltage and Frequency Scaling）機能を使い，プロセス毎に必要十分な CPU 資源の割り当てを行う [13][5] ことで消費電力を削減する．DVFS はその名の通り，動的に CPU の電圧と動作周波数を変化させることで消費電力を押さえる手法であり，昨今のほとんどの CPU が標準搭載している．

POPM はスケジューラがプロセスのコンテキストスイッチを行う際に CPU をプロセスに合わせた動作周波数に変更する．CPU 資源をどの程度割り当てるかについては静的な手法と動的な手法を提供しており，静的な手法ではあらかじめどのプロセスをどの周波数で動作するかの対応表を定義しておき，その対応表に従って動作周波数を変更する．また，動的な手法ではプロセスがインタラクティブなプロセスかそうでないかを調べ，インタラクティブなプロセスに対しては CPU 資源を多く割り当てる．これは，利用者から見て GUI などのインタラクティブなアプリケーションの応答性が落ちるとシステムの応答性が低下するためである．プロセスの判定方法にはプロセスのスリープ時間を参照する．入力待ちなどが多く発生するインタラクティブプロセスではプロセスは頻繁にスリープするはずである．逆にまとまった仕事を処理する入力待ちが少ないノンインタラクティブプロセスでは，スケジューラに割り当てられた CPU 時間を使い切ることが多くなる．POPM ではこの点に着目してインタラクティブプロセスとノンインタラクティブプロセスを判別することで，動的なプロセスの優先度決定を実現する．

本研究との関連としては，プロセス毎の資源管理を目指していることが共通している．しかし，POPM では CPU の資源管理にのみ着目しているため，CPU 以外の資源については特に対応していない．POPM は低消費電力を目指したものなので特に問題はないが，本研究の目的はアプリケーションの動作時間保証のため，CPU 資源のみを見ているだけではノード全体の電力消費量を知ることができず不十分である．特に，マイクロユビキタスノードでは図 3.2 で示した通り，CPU 以外のデバイスの消費電力が無視できないほどに大きいため，ノード全体の消費電力を考慮するためには複数のデバイスの消費電力を監視することが不可欠である．

6.2.2 デバイス毎の資源管理

P. Levis，S. Madden らの提案する TinyOS は，センサノードのための OS である [11]．TinyOS は資源の限られたセンサノードに対して，nesC と呼ばれる C の拡張言語を用いてプログラミングを行う [6]．nesC の特徴はプログラムをコンポーネントと呼ばれる小さな機能単位に分割し，それらのコンポーネントを接続するだけで基本的なアプリケーションを容易に作成できるところにある．nesC は C のプリプロセッサとして働き，コンパイル時に必要なコンポーネントを接続した上で C のソースコードを出力する．コンポーネント化により，容易にアプリケーションを構築できるだけでなく，各コンポーネントは必要な時にのみ有効にされるため，各デバイスは必要なときにだけ起動される．例えば，他のノードからの通信を待ち受ける場合，無線受信チップのみを有効にしてメイン CPU はスリープする．受信チップがパケットを受け付けると，無線チップからメイン CPU へと割り込みを発生させ，スリープ状態を解除する．この様な手法により，必要のない電力を極力使わないような設計がなされている．

こうした研究は，デバイス毎に電源を制御することでバッテリ管理を実現するという点において本研究との関連がある．しかし，P. Levis らの研究目的がノードの省電力化にあるのに対し，本研究の目的はアプリケーションの動作保証である点が異なる．省電力化によりバッテリ消費を抑えることはノードレベルで重要だが，アプリケーションの動作保証によってサービスの高可用性を確保することは利用者やサービス品質のレベルで重要である．

6.2.3 消費電力計測・予測

Xiaofan, J. らはセンサノードに取り付けることで，デバイスの消費電力量を詳細に計測できるハードウェアを開発した [9]．この手法では，既存のセンサノードに提案するマイクロパワーメータを実装する必要があり，研究・開発における実験環境であればまだしも，全ての製品に搭載するには 3.2 で挙げたような搭載コストを始めとする問題が残る．

この問題を受けて，Dunkels, A. らはセンサノードにおいてソフトウェアでの使用電力予測手法を提案している [4]．この研究で採用しているモデルは本研究とほぼ同じもので，各デバイスの使用時間を測定し，単位時間あたりの使用電力をかけあわせたものをノード全体の消費電力とする．Dunkels, A. らの研究では各デバイスの単位時間あたりの使用電力を計測するのにデジタルオシロスコープを用いて直接回路上での電力を測定している．この手法は，実際のハードウェアの消費電力を物理的に測定するので現実の値との誤差がほとんど無いデータを取得することができる．一方で，回路に直接プローブを接続するという計測手法は，近年のマイクロユビキタスノードの小型化に伴い実現が難しくなっている．例えば，1 チップ内に CPU，無線通信，センサなどの機能を全て搭載した LSI などの場合，そもそも機能別の電力を計るためのプローブを接続することができないため実現不可能である．

本研究では，単位時間あたりの消費電力もソフトウェアによって計測することでこ

の問題を解決している．ただし，実際のハードウェアから取得したデータではないので，計測したデータが正しいのかどうかを常に疑う必要がある．この問題については 7 章の今後の課題として取り上げる．

6.3 まとめ

本章では本研究における関連研究を紹介した．資源管理や省電力に関する研究の中には，本研究と共通する目的や構造を持つものもあるが，バッテリ管理に着目してアプリケーション別動作時間保証を実現することのできる既存研究は存在しないため，本研究には新規性がある．

第7章

結論

本章では今後の課題について述べ，最後に本論文をまとめる．

7.1　今後の課題

本節では，本研究を進める上で出てきた課題，及び現状の pSurvive では対応していないが，今後対応していく必要がある課題について議論する．

7.1.1　アプリケーション別消費電力計測・予測機構の課題

計測誤差への対処方法についての検討

本研究は消費電力計測をソフトウェアによって行うことでシステムの可搬性を高めているが，一方でどの程度の精度が保証できるのかという問題は常に考えなければならない．その際，計測の精度を上げていくこともちろん大事だが，ソフトウェアのみで実現する以上どうしてもある程度のばらつきが出てしまうことが予想される．こうした計測誤差についてどのように立ち向かうのかというのは重要な問題である．

一つの考えとして，ソフトウェアで計測した計測値とは別に，その計測自体の信頼度というものを定義するという手法が考えられる．この関係は，天気予報での降水確率とその予報自体の精度の関係に近い．天気予報では 100 パーセントの予報がされていても実際には雨が降らないこともあるが，天気予報自体が外れる可能性があるという前提があれば，何かしら前持って対処をすることができる．このように，仮に計測値が外れても，計測自体が当たる確率を知ることができ，仮に外れるとした場合にどの程度のばらつきが生じうるかということが分かれば，そのばらつきを含めた上で電力量を予約することも可能となる．当然，100 パーセントの精度が要求されるミッションクリティカルアプリケーションにおいて，こうした不確定要素を無視することはできないが，そうでないアプリケーションの場合には十分に検討の余地があるだろう．

電力予測アルゴリズム決定手法についての検討

電力予測アルゴリズムの選択について，誰がどのようにしてアルゴリズムを決定するのかという課題がある．本研究ではアルゴリズム自体の評価のために静的な実装を行ったが，実際にこの問題を考えるとシステム側が決定するか，アプリケーション側が決定するかということになる．システム側がアルゴリズムを決定する場合，アプリケーションは特に電力予測について気にする必要が無いという利点があるが，システム側はどのアルゴリズムを適用するかをアプリケーションの振る舞いに応じて自分で選択しなければならない．そのため，アプリケーションによっては必ずしも最適な予測アルゴリズムが決定されるとは限らない．一方，アプリケーション側から予測アルゴリズムを決定する場合，アプリケーション側に予測アルゴリズムを選択するという機能追加の必要がある．しかし，アプリケーション自身が自分がどのような仕事をするのかを知っている場合，アプリケーション自身が最適な予測アルゴリズムをシステム側に通知することができる．これにより，いくつかの状態を持ち，状態遷移によって大きく消費電力傾向が変わるようなアプリケーションにも柔軟に対応できるだろう．

7.1.2　アプリケーション別動作保証機構の課題

動作保証外アプリケーションの停止手順についての検討

アプリケーション動作保証の手法として，pSurvive では単純に保証対象外のアプリケーションを終了させるという手法を選択したが，終了のさせ方について問題が残る．具体的な問題の例を挙げる．利用者が自分で定義したいくつかのアプリケーションの動作保証が行われている環境下で，動作保証外である Web ブラウザを起動し，Web ブラウジングをしているとする．この状態でバッテリ残量が低下した場合，pSurvive は利用者が利用している Web ブラウザを問答無用で終了してしまう．しかし，利用者が今まさに使っている Web ブラウザを強制終了してしまうという手法はあまりにも乱暴である．なぜなら，利用者にとってバッテリ残量の予約はバックグラウンドで行われるので目に見えないが，Web ブラウザは目に見えるアプリケーションであるため，勝手に終了されてしまった時の衝撃は大きい．そのため，終了するだけでなく，終了する前にバッテリ予約のためにそのアプリケーションを終了する旨を通知するなどの機構が必要だと考えられる．

停止対象アプリケーションの選択についての検討

アプリケーションを終了させる際に複数の動作保証外アプリケーションが起動していた場合に，全てのアプリケーションを同時に終了してしまうのではなく，その中でも優先度の低いものから終了させることにより，動作保証外アプリケーションであっても優先度の高いものの動作時間を長くするといった機能が考えられる．例えば，優先度は低いが電力を 100 使うアプリケーション A と優先度は高いが電力は 10 しか使わないアプリケーション B が動作していた場合，A を少し早めに終了させることで B の動作時間を長くすることができるだろう．こうした機構を考える場合，どのようにして優先度を決定するのか議論が必要である．単純に思いつくのは，アプリケーションを起動した順や，利用者が頻繁に利用する順などである．このような柔軟な対応を行うことで，さらに利用者にとって便利なシステムが実現できるだろう．

7.1.3　システムの可搬性に関する課題

他のマイクロユビキタスノード環境での検証

本論文では Gumstix と NUTS センサボード環境によって実験を行ったが，pSurvive が他のマイクロユビキタスノード環境でも有効に動作するのかという検証を行う必要がある．より多くのマイクロユビキタスノード環境での有効性がとシステムの可搬性が示されることで，本研究の価値はさらに高まるだろう．また，異なる環境で動作させることにより，これまでには気づかなかった新たな問題や知見が得られるだろう．

7.2 まとめ

本論文では複数のアプリケーションが動作するバッテリ駆動のマイクロユビキタスノードにおいて，残バッテリ量からアプリケーション動作時間を予測したり，前もって特定のアプリケーションが指定時間動作するためのバッテリ量を予約する機構が無いことに着目した．このことは，これからもマイクロユビキタスノードの普及が進み，社会の中でより重要度の高いサービスを提供していく上で大きな問題となる．

そのため，本論文では，**特定のアプリケーションの動作時間保証を実現する pSurvive** を設計，実装し，評価を行った．pSurvive では，アプリケーションがマイクロユビキタスノード上の各資源を利用するのを監視し，どのアプリケーションがどれだけの電力を消費しているのかを計測した上で，将来の消費電力を予測する**アプリケーション別消費電力計測・予測機構**，及び，その予測結果を基に実際にアプリケーションの動作保証を行う**アプリケーション別動作保証機構**の二つのモジュールから構成され，相互に作用して動作時間保証を実現する．pSurvive はソフトウェアベースで実装されているため，特殊なハードウェアを必要とせず，既に利用されている製品や，今後登場するであろう製品に対しても低コストで導入することができる．

本研究の成果により，これまでバッテリ切れの問題によって信頼性や可用性が低かったマイクロユビキタスノード上で動作時間保証を行うことで，アプリケーションの品質保証を実現することができるようになる．このことにより，現在の利用者は意図しないバッテリ切れのリスクから解放され利便性が高まるとともに，サービスプロバイダは新たに高信頼性を要求されるアプリケーションの開発が可能となる．これはマイクロユビキタスノードがさらに普及していくであろう今後の社会において，大きな利益をもたらすものである．

謝辞

本研究を進めるにあたって貴重な御指導を賜りました，慶應義塾大学環境情報学部 教授 徳田 英幸教授に深く感謝致します．また，重要な御助言を頂きました， 慶應義塾大学環境情報学部 村井 純教授，並びに，東京電機大学未来科学部 戸辺 義人教授に深いお礼を申し上げます．

慶應義塾大学徳田・村井・楠本・中村・高汐・重近・バンミーター・植原・三次・中澤合同研究室の皆様に多くの御助言を頂きました．特に，活動の中心となった，ECN研究グループの間 博人氏及び斉藤 匡人氏には論文執筆，研究方針について大変多くの助言を頂きました．大変感謝しております．

お互いに厳しい論文執筆環境の中励まし合った本多 倫夫氏と金澤 貴俊氏の心遣いに感謝致します．また，お忙しい中にもかかわらず重要かつ的確な御助言をして頂きました中澤 仁氏に深く感謝します．

平成 21 年 1 月 13 日
森 雅智

参考文献

[1] Apple： Nike + iPod. http://www.apple.com/ipod/nike/.

[2] Michael Beigl, Christian Decker, Albert Krohn, Till Riedel, and Tobias Zimmer. μParts: Low cost sensor networks at scale. In *Ubicomp Demo Session*, 2005.

[3] P. Buonadonna, J. Hill, and D. Culler. Active message communication for tiny networked sensors. Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM ' 01), 2001.

[4] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th workshop on Embedded networked sensors*, pp. 28–32. ACM Press New York, NY, USA, 2007.

[5] Enhanced Intel SpeedStep Technology How To Document. http://www.intel.com/cd/channel/reseller/asmo-na/eng/203838.htm.

[6] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pp. 1–11, 2003.

[7] Gumstix website. http://www.gumstix.com/.

[8] M. Ito, Y. Katagiri, M. Ishikawa, and H. Tokuda. Airy Notes: An Experiment of Microclimate Monitoring in Shinjuku Gyoen Garden. In *Networked Sensing Systems, 2007. INSS'07. Fourth International Conference on*, pp. 260–266, 2007.

[9] X. Jiang, P. Dutta, D. Culler, and I. Stoica. Micro power meter for energy monitoring of wireless sensor networks at scale. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pp. 186–195. ACM Press New York, NY, USA, 2007.

[10] JR 東日本：モバイル Suica 公式サイト. http://www.jreast.co.jp/mobileSuica/.

[11] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. TinyOS: An Operating System for Sensor Networks. *Ambient Intelligence*, pp. 115–148, 2005.

[12] C.W. Mercer. An Introduction to Real-Time Operating Systems: Scheduling Theory. *Unpublished manuscript*, 1992.

[13] D. Miyakawa and Y. Ishikawa. Process oriented power management. *SIES '07.*, pp. 1–8, 2007.

[14] N. Nishio and H. Tokuda. iReserve Architecture: An Integrated Resource Reservation Mechanism. *Transactions of Information Processing Society of Japan*, pp. 2645–2658, 1999.

[15] N. NISHIO and H. TOKUDA. New Directions in System Software. Design and an Implementation of Resource Reservation with QOS Profiling Handler. *Transactions of Information Processing Society of Japan*, Vol. 42, No. 6, pp. 1570–1579, 2001.

[16] NTT ドコモ:iチャネル. http://www.nttdocomo.co.jp/service/news_message/ichannel/.

[17] OpenEmbedded project web site. http://wiki.openembedded.net/.