モデル駆動開発による 消費電力自己適応型ソフトウェアの開発方法論

田中 文也[1,a)]　久住 憲嗣[2,3,b)]　石田 繁巳[3,c)]　福田 晃[2,3,d)]

概要：組込みシステムは電力制限と多機能化による稼働時間短縮の背景から最大消費電力削減の必要がある．組込みソフトウェア開発には消費電力削減とサービス品質の両立の要求が存在し，ソフトウェアがハードウェアの消費電力に合わせて自身の消費電力を変更できれば要求を満たすことができる．本論文では，消費電力について自己適応を行うソフトウェアのモデルベースの開発方法論を提案する．提案手法では，ソフトウェアプロダクトライン開発におけるフィーチャモデルと ExecutableUML で記述したステートマシン図を関連付け実行時に振る舞いを変更するソフトウェアを開発する．この手法を用いることで端末の消費電力に応じたソフトウェアの消費電力の変更が可能でトレードオフの両立が可能になる．評価の結果，平均応答時間 0.22 秒，適応率 87.6%で消費電力について適応を行うことができた．

キーワード：組込みシステム，自己適応ソフトウェア，モデル駆動開発，消費電力解析

A Methodology to Develop Energy Adaptive Software Using Model-Driven Development

Tanaka Fumiya[1,a)]　Hisazumi Kenji[2,3,b)]　Ishida Shigemi[3,c)]　Fukuda Akira[2,3,d)]

Abstract: In an embedded system development, it is one of the crucial tasks to reduce maximum power consumptions in order to power source limitation. We have to solve a trade-off between power consumption and quality of service. If the software can change the power consumption in accordance with the power consumption of hardware, the software can achieve both of reduction of maximum power consumption and service quality. In this paper, we propose a model-based development methodology of software performing self-adaptive for power consumption. In the proposed method, we develop software which changes its behavior at runtime by linking state machine diagrams described by ExecutableUML to a feature-model used in Software Product Line Development. This method makes it possible to change power consumption caused by software behavior according to the power consumption of whole of the target device. The target software can maximize the quality of service in a certain power constraint. Therefore the target software can achieve the tradeoff between power consumption and quality of service. As a result of the evaluation, average response time was about 0.22 seconds, and the adaptive rate was about 87.6%.

Keywords: Embedded System, Self-adaptive Software, Model-Driven Development, Energy Analysis

[1] 九州大学大学院システム情報科学府
Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University, Motooka 774, Nishi-ku, Fukuoka 819-0395, Japan
[2] 九州大学システム LSI 研究センター
System LSI Research Center, Kyushu University, Motooka 774, Nishi-ku, Fukuoka 819-0395, Japan
[3] 九州大学大学院システム情報科学研究院
Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University, Motooka 774, Nishi-ku, Fukuoka 819-0395, Japan
[a)] tanaka@f.ait.kyushu-u.ac.jp
[b)] nel@slrc.kyushu-u.ac.jp
[c)] ishida@f.ait.kyushu-u.ac.jp
[d)] fukuda@f.ait.kyushu-u.ac.jp

1. はじめに

組込みシステムはハードウェアの大きさや製造コストに制限がある．バッテリー駆動の場合，バッテリーサイズが制限されると蓄電容量が小さくなるため稼働時間が短くなる．また，機能の多様化にともなうタスク実行シナリオの多様化により組込みシステムの使用頻度は上昇しており，稼働時間の短縮が問題である．これらの背景からユーザが要求する稼働時間を実現するためにソフトウェアの消費電力を制限する必要があり，最大消費電力削減の要求が存在

する．一方で，消費電力とサービスの品質にはトレードオフが存在する．組込みソフトウェア開発ではソフトウェアの消費電力削減とサービス品質向上の両立が要求される．ソフトウェアがハードウェアの消費電力に合わせて自身の消費電力を変更することができれば，端末の最大消費電力の削減とサービス品質の両立が可能である．周囲の状況に応じて振る舞いを変えることができる自己適応型ソフトウェア [1] が存在する．組込みソフトウェアは設計時に想定されていないシナリオで実行された場合に組込みシステムの動作を保証することができないため，実行時のシナリオに応じたタスク実行の機能が期待される．実行時のシナリオは常に変化する．そのため状況に応じた最適な振る舞いは設計時には記述することが難しく，変化する実行時の状況に応じてソフトウェアは最適な振る舞いをするように動的に判断を下す必要がある．本論文では他のソフトウェアの消費電力や電源の状況といった電力の状況を周囲の状況として取り扱い自己適応を行う．

ソフトウェア開発の手法にモデル駆動開発 [2] とソフトウェアプロダクトライン開発方法論 (Software Product Line; SPL)[3] という手法が存在する．モデル駆動開発はより上流の過程で検証を行うことで訂正コストを削減することを目的とした開発手法である．また，コードを自動生成するためモデルとコードの一貫性を保つことが容易である．SPL はソフトウェア単体ではなく，ソフトウェア群全体の開発を最適化する手法である．この手法では，ソフトウェア群中のソフトウェア間での共通性と可変性を分析したツリー構造をフィーチャモデルとしてモデル化する．フィーチャモデルのツリーの各可変性がソフトウェア間の相違点であり，ツリーから共通フィーチャと可変フィーチャを組み合わせて選択し設計することで製品群全体としての生産効率を向上させることが可能である．

本論文では SPL 手法で分析したフィーチャモデルと，ExecutableUML[4] を用いて記述したステートマシン図を紐付けて開発を行う消費電力自己適応型ソフトウェアの開発手法を提案する．フィーチャをソフトウェアの振る舞いに割り当て，バリエーションによって異なる振る舞いの違いをステートマシン図にステートの違いで記述する．消費電力状況に応じてフィーチャを動的に選択することで実行時に自身の振る舞いを変更し自己適応を可能とする．設計時にあらかじめ，モデルベースの消費エネルギー解析手法を用いてバリエーションごとの消費電力を推定しておき，推定値をもとに振る舞いを変更できるようにする．実行時にはほかのソフトウェアの消費電力や電源の状況に応じて振る舞いを変更することで消費電力とサービス品質の観点から最適に実行されるソフトウェアを記述することが可能である．

本論文の構成は以下の通りである．第 2 章では既存研究として自己適応ソフトウェアの開発における SPL 手法の応用について述べる．第 3 章では提案手法についての説明を行う．第 4 章では評価方法と評価結果を述べ，第 5 章ではまとめと今後の課題について述べる．

2. 既存研究

本節では SPL の手法を用いた自己適応についての既存研究について説明する．既存研究は，設計時に準備をし実行時に動的に判断を下す点で本研究との類似性がある．

2.1 可変モデルを用いたサービス構成の動的適応

文献 [5] では，Web サービスの操作をフィーチャとした SPL の手法を用いて，実行時にサービス構成の動的適応を行うフレームワークが提案されている．このフレームワークは Web サービスの設計時と実行時の両方で利用される．設計時には，実行時の動的適応をガイドするモデルの生成をサポートする．実行時にはこれらのモデルを用いて適応を行う．実行時の適応はフレームワーク中の MoRE-WS(Model-based Reconfiguration Engine for Web Service) が行う．MoRE-WS による実行時の適応の流れを以下に述べる．

(1) CONTEXT MONITOR によって検出されるコンテキストの変化に応じてコンテキストモデルを更新.
(2) コンテキストモデルの情報から定められた SLA(Service Level Agreement) を違反していないかを判断.
(3) SLA を違反していた場合，フィーチャの活性・不活性を行い適応規則に従って可変フィーチャモデルを適応.
(4) 適応されたフィーチャモデルからサービス構成の再調整案を生成しサービス構成を調整.
(5) 調整されたモデルは，WS-BPEL(Web Services Business Process Execution Language) コードにフラグメントを追加・削除してサービス構成に反映.

文献 [5] でコンテキストとして扱われ自己適応を引き起こしているのは Web サービスの操作に付随するプロパティである．プロパティの値が規定の値になった時に達成される context conditions によって SLA の違反が表現される．また，自己適応は SLA の違反によって引き起こされるため，適応のコンテキストとして扱われているものはサービスの操作のプロパティであると言える．

フレームワークの動的適応に関する評価を GQM(Goal Question Metric) モデル [6] を用いて評価している．評価に用いられた二つの GQM モデルを表 1，表 2 に示す．

表 1 の GQM モデルの評価では 30,000 個の要素を持つモデルに対して最大 300 ミリ秒で適応の操作を行うことができた．これはフレームワークで取り扱う範囲では十分な応答速度である．

表 2 の GQM モデルの評価では，1 ミリ秒から 5 ミリ秒間隔で 100 個の問題のあるコンテキストイベントを発行したが MoRE-WS は性能を低下させることなく context

表 1 GQM モデル 1
Table 1 GQM model 1

評価層	内容
Goal	MoRE-WS の視点から効果的なサービス構成の動的適応の実行
Question	MoRE-WS が効果的に動的適応を実行可能か
Metrics	モデル内の要素の数
	サービス構成の追加・削除の応答速度
	現在の設定の取得速度
	フィーチャを割り当てたサービス操作の取得速度
	構成モデルの要素を WS-BPEL に反映する速度

表 2 GQM モデル 2
Table 2 GQM model 2

評価層	内容
Goal	システム分析側の視点から高負荷下での飽和回避
Question	MoRE-WS が高負荷下で十分な性能を発揮可能か
Metrics	問題のあるイベントの数
	問題のあるイベント間のタイムフレーム
	コンテキストを監視する頻度
	影響されうる context conditions 数
	高負荷下での応答速度

conditions を調整することができた．よって MoRE-WS は高負荷下でも性能を低下させることなく動作できると評価できる．

2.2 既存研究の問題点

既存研究で提案されるフレームワークは高い抽象度で適応を実現している．そのため実装に近いモデルや情報を取り扱う適応に関して問題がある．例えば，既存研究では端末上の消費電力は実装に非常に近い情報であるためコンテキストとして取り扱うことができない．また，ソフトウェアレベルで消費電力を扱うことができる手法 [7] は存在するが消費電力を扱って自己適応を行うソフトウェアの開発方法論はまだ確立されていない．

3. 提案手法

既存研究では実装に近い情報を取り扱うことができなかったため，消費電力を対象とした自己適応が不可能であった．本手法ではモデルベースの消費エネルギー解析手法を取り入れ，端末上の消費電力をコンテキストとして取り扱い消費電力について自己適応を行う．この手法を用いることで消費電力状況に応じてソフトウェアの振る舞いを動的に変更することが可能である．

3.1 全体像

提案手法によるソフトウェア開発の全体像について述べる．

まず，設計時に電力状況によって異なるソフトウェアの

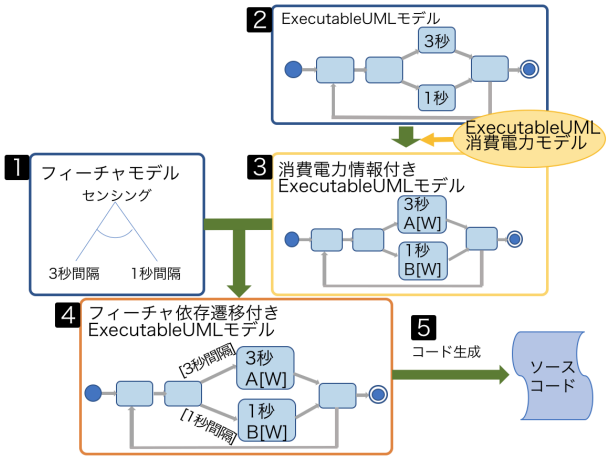

図 1 開発手法の全体像
Fig. 1 Overviw of proposed method

実行時の振る舞いをフィーチャとしてフィーチャモデルを作成する．つぎに，ExecutableUML で振る舞いのモデルを作成する．振る舞いのモデルでは，電力状況によって異なる振る舞いをステートマシン図のステートの違いで表現する．次に，モデルベースの消費エネルギー解析手法を用いて，作成したステートマシン図からバリエーションごとの消費電力を推定する．この推定値をステートマシン図に追加することで消費電力情報付きの ExecutableUML モデルを作成する．次に，消費電力情報付きモデルとフィーチャモデルを紐付けフィーチャ依存遷移付き ExecutableUML モデルを作成する．このモデルは選択フィーチャによって状態遷移に制限が存在する．最後に，モデル駆動開発に基づく自動コード生成によってソースコードに変換しソフトウェアとして実行する．開発ソフトウェアは，端末の実行時の電力状況に応じた動的バリエーション変更によって，実行時に要求される最大消費電力以下での実行が可能である．図 1 に提案手法の全体像を示す．

3.2 フィーチャモデルの作成

フィーチャモデルの作成について説明する．フィーチャモデルはフィーチャ間の関係を表現したモデルである．本論文ではソフトウェアの実行時の振る舞いをフィーチャとしてフィーチャモデルを作成する．

実行時の振る舞いのうち，バリエーションによらない振る舞いを共通フィーチャ，バリエーションによっては行われない振る舞いを可変フィーチャとする．可変フィーチャのうち，複数の振る舞いが排他的に選択される振る舞いを択一フィーチャ，選択されない場合がある振る舞いをオプションフィーチャとする．フィーチャモデルは共通フィーチャと可変フィーチャをツリー構造にしてモデル化することで作成される．

ソフトウェアは可変フィーチャのうちで選択されたフィーチャと共通フィーチャから成る．フィーチャモデルにおい

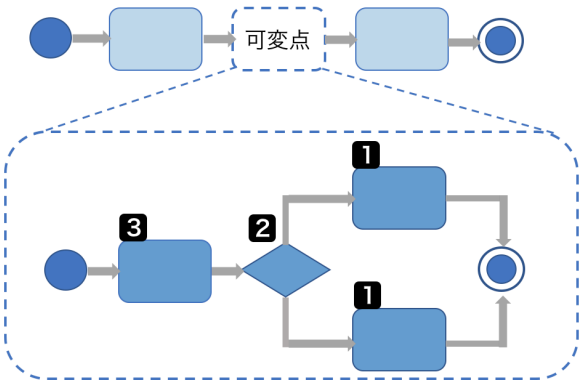

図 2 振る舞いのモデルのメタモデル
Fig. 2 The metamodel of the behavior model

て，各可変点で選択されたフィーチャによって生成される選択性のない一つのツリーがソフトウェアの持つ全てのフィーチャを持ち，全ての振る舞いを表現する．本論文で提案する手法で開発するソフトウェアではこの選択性のない一つのツリーが一つの振る舞いのバリエーションを表現することになる．

3.3 ExecutableUML モデルの設計

ExecutableUML モデルの設計について説明する．ソフトウェアの機能を実現する振る舞いのモデルと消費電力状況をモニタリングするモデルを ExecutableUML モデルのステートマシン図で記述する．モニタリングするモデルでは消費電力状況を取得する周期的な振る舞いを設計する．振る舞いのモデルでは，可変点毎に自己適応に必要なステートを設ける．必要なステートは，消費電力状況からバリエーションを判断するステートと選択されたステートへの遷移を行うステートである．バリエーションを判断するための情報は次項 3.4 で説明する方法でモデルに追記する．設計する二つのステートマシン図は実行時には非同期で並列に動作する．図 2 に振る舞いのモデルの設計のメタモデルを示す．また，以下に ExecutableUML モデル作成における操作の流れを述べる．操作 (1)〜(3) は振る舞いのモデルの設計操作について説明しており図 2 中の番号に対応している．

(1) フィーチャモデルに設計したソフトウェアの振る舞いをステートに割り当て，遷移先のステートの違いでバリエーションの違いを表現．
(2) バリエーションを表現するステート群の前に選択されたバリエーションのステートへの遷移を扱うステートを設置．
(3) (2) で追加したステートの前に，消費電力状況を参照しバリエーションの判断を行うステートを設置．
(4) モニタリングモデルとして周期的な振る舞いを行うステートマシン図を記述．
(5) モデルの振る舞いが想定した動作であるかを検証．

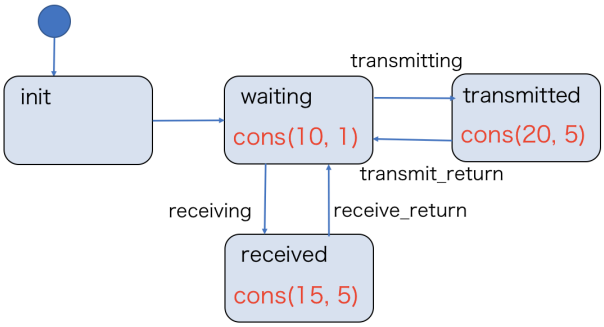

図 3 モデルベース消費エネルギー解析手法の全体像
Fig. 3 Overview of Model-based Energy Analysis Method

3.4 消費電力情報付き ExecutableUML モデルの生成

消費電力情報付き ExecutableUML モデルの生成について説明する．消費電力情報付き ExecutableUML モデルはモデルベースの消費エネルギー解析手法を用いて推定した推定消費電力を作成した ExecutableUML モデルに追記することで生成する．

モデルベースの消費エネルギー解析手法 [7] は，ExecutableUML を用いてモデルベースでステートごとの粒度で消費エネルギーを推定することができる手法である．図 3 にモデルベース消費エネルギー解析手法の全体像を示す．この手法では，ExecutableUML モデルのステートマシン図にリソース消費量を書き加え，これをもとに消費電力モデルからステートごとの消費エネルギーを推定する．文献内で消費エネルギー推定に用いられている電力モデルを以下に示す．

$$Power(A) = 0.3819 + 0.0003 \times CPU(\%) + 0.0071 \times Wi\text{-}Fi(MB/sec) \quad (1)$$

電力モデル 1 にはパラメータとして CPU 使用率と Wi-Fi 通信量の二つを用いている．図 3 に記述されている cons はそのステートでの (CPU 使用率,Wi-Fi 通信量) を表している．設計時にステートマシンのレベルでの消費エネルギー推定ができるため，どの振る舞いでどれだけのエネルギーが消費されているかのボトルネックを設計時に発見することができる．この手法による推定では，平均誤差 9.0%の精度で消費エネルギーを推定することが可能である．

本論文ではこの手法を用いてモデル設計の段階で振る舞いの各バリエーションの消費電力を推定する．推定値は振る舞いの変更の基準として使用するために ExecutableUML モデルに追記し消費電力情報付き ExecutableUML モデルとする．

3.5 フィーチャとステートの紐付け

フィーチャとステートの紐付けについて説明する．本手法では，ソフトウェアの機能をフィーチャとして機能の振る舞いをステートマシン図で設計する．作成したフィー

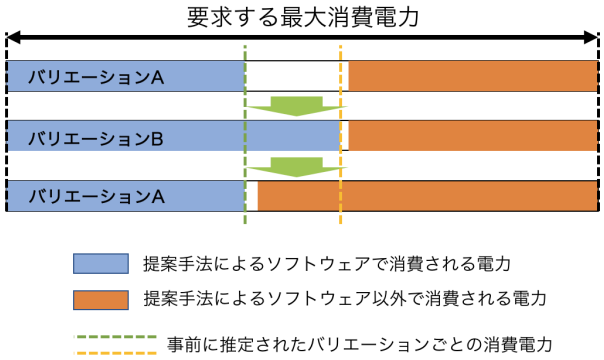

図 4 実行時の振る舞い
Fig. 4 Runtime behavior

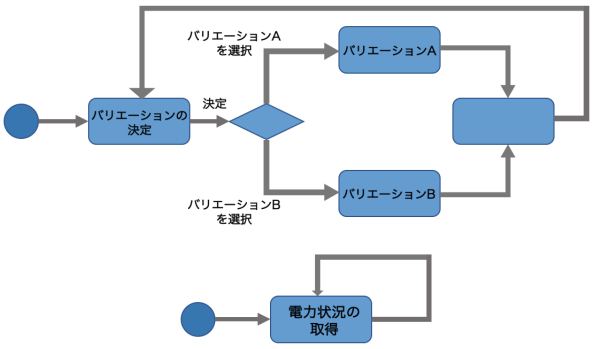

図 5 電力状況の取得とバリエーションの変更
Fig. 5 Power situation acquisition and variation change

チャモデルと消費電力情報付き ExecutableUML モデルをそれぞれの構成要素で紐付けて新たに ExecutableUML モデルを生成する．

フィーチャとステートの紐付けには可変点付き状態遷移図を用いる．可変点付き状態遷移図は可変性をガード条件で表現することでフィーチャの活性・不活性によって変わる状態遷移の有無を表現する．フィーチャモデルでは，ツリーの可変点から一つずつフィーチャを選択することで一つのバリエーションを表現する一つのツリーが得られる．また，可変点付き状態遷移図では，状態遷移に付随するガード条件によって選択されたフィーチャに依存する状態遷移のみが有効になる．これらのことから，バリエーションが存在する状態遷移図が一つのバリエーションにしか遷移し得ないようになる．結果として，フィーチャモデルにおいて選択されたフィーチャに依存する状態遷移のみが有効な状態遷移図が作られる．一つのフィーチャツリーと一つの状態遷移図が一対一で対応することになりフィーチャとバリエーションのステートは紐付けされたと言える．生成されたフィーチャ依存遷移付き ExecutableUML モデルはモデル駆動開発における自動コード生成を用いてソースコードへと変換される．

3.6 実行時の振る舞い

提案手法によって開発されたソフトウェアの実行時の振る舞いについて説明する．実行時には端末上の消費電力状況を取得しながら振る舞いを変更する．実行時の振る舞い変更は事前に見積もったバリエーション毎の消費電力の情報を基に行われ，実行時に要求される最大消費電力以下で最大のパフォーマンスで動作するバリエーションが選択される．実行時の振る舞い変更のイメージを図 4 に示す．

図 4 では，バリエーション B の方が消費電力が大きく高パフォーマンスなバリエーションである．バリエーション A と B は消費電力とパフォーマンスについてトレードオフの関係がある．参照した消費電力状況から，バリエーション A で実行中の端末がバリエーション B で実行しても問題ないと判断した場合にバリエーション B へと自身の振る舞いを変更する．また，バリエーション B で実行中に他のソフトウェアの消費電力の増大などの原因で端末の消費電力が増大し，バリエーション B での実行に問題があると判断した場合，バリエーション A へと変更するでことで最大消費電力を削減する．

3.7 電力状況の取得とバリエーションの変更

提案手法によって実行時に振る舞いを変更するためには，実行時に端末の消費電力を取得しバリエーションを変更する必要がある．また，それを実現する機構を設計時の ExecutableUML モデルに埋め込む必要がある．本項では実行時の消費電力状況の取得方法とバリエーションの変更方法について説明する．

電力状況の取得はモニタリングのステートマシン図に記述される．取得動作はソフトウェアの振る舞いと非同期で周期的に行われ，ソフトウェアの振る舞いのステートマシンと電力状況を取得するステートマシンは並列で実行される．実行時の消費電力状況は，周期的実行における前回の実行から消費されたリソース消費量からモデルベースの消費エネルギー解析手法を用いて取得される．

バリエーションの変更は可変点ごとに行われる．バリエーションの決定は，最新の消費電力状況を参照し，これからバリエーションを判断する関数によって行われる．バリエーション判断関数は消費電力状況から，要求される最大消費電力を超えずに最大のパフォーマンスを発揮するバリエーションを選択する．図 5 に電力状況の取得とバリエーションの変更を行うステートマシン図を示す．

4. 評価

本節では，提案手法を用いて開発したソフトウェアについての評価を述べる．評価環境，評価項目，評価手順について説明した後，本実験で評価するソフトウェアとテストプログラムの仕様を説明し，最後に評価結果について述べる．評価ソフトウェアの作成には以下の式 2 を消費電力モ

表 3 評価環境
Table 3 Evaluation environment

種類	機器名
端末	Raspberry Pi Model B+
電流計	Agilent 34411A 61/2 Digit Multimeter
電源装置	HEWLETT PACKARDE 3616A DC POWER SUPPLY
拡張ボード	Grove Pi+
センサモジュール	Grove - Temperature Sensor v1.2

デルとして使用した．消費電力モデルは文献 [7] を参考に線形モデルを選択した．消費電力モデルの各パラメータを新たに設定するため，それぞれに対して段階的に変化させるプログラムを実行してデータを取得した後，最小二乗法による線形回帰で決定した．

$$Power(A) = 0.279 + 0.0007 \times CPU(\%) + 0.0071 \times Wi\text{-}Fi(MB/sec) \quad (2)$$

本評価では評価ソフトウェアが端末の消費電力状況を取得することができ，取得した情報を用いて振る舞いを変更することができることと振る舞い変更の精度に着目して評価を行う．

4.1 評価環境

表 3 に示す環境で評価を行った．Raspberry Pi には電源装置と電流計と接続して測定を行った．センサモジュールは Raspberry Pi と接続した Grove Pi+のアナログ端子に接続する．評価実験 1 と評価実験 2 は同じ評価環境で行った．

本実験では評価するソフトウェアのモデリングを BridgePoint[8] で行った．BridgePoint は ExecutableUML の方法論を取り入れたツールで，記述したモデルをモデルのまま動作させて検証することが可能である．モデルからコードの自動生成を行うことができモデル駆動開発におけるツールとして活用されている．

BridgePoint では，ExecutableUML の方法論によって記述されたクラス図のうち状態を持つものにステートマシン図を記述する．また，ステートマシン図のステートのうちステート内で振る舞いがあるものは専用のアクション言語を用いて詳細な振る舞いを記述できる．アクション言語は抽象度の高い言語であり，変換先のソースコードの種類によらない．モデルは Bridge や Function を用いて外部のコードと接続することが可能である．本研究ではアクション言語や Bridge，Function を用いて完全なモデル駆動開発を行うことが可能であると考え BridgePonint を開発ツールとして採用した．

表 4 評価項目
Table 4 Evaluation item

項目	内容
応答時間	消費電力状況の変化からバリエーション変更までの時間
ミス頻度	消費電力状況に合わないバリエーション変更の 1 秒あたりの回数
修正時間	ミスの発生からあるべきバリエーションへの変更の時間
適応率	正しいバリエーションである時間的割合
推定誤差	消費電力推定の相対誤差
オーバーヘッド	自己適応性を持つことによる消費電力上昇率

4.2 評価項目

本評価実験における評価項目について説明する．提案手法によって開発されるソフトウェアには消費電力状況に対してより正確でより高速な適応が求められる．また，自己適応機能を付加したことでソフトウェアの消費電力は上昇すると考えられる．これらの要求から表 4 に示す 6 項目について評価を行った．応答時間，ミス頻度，修正時間，適応率，消費電力推定の 5 項目については評価実験 1 で評価を行い，オーバーヘッドについては評価実験 2 で評価を行う．評価項目のミスの発生は消費電力推定の誤推定によって消費電力状況を正しく推定できなかった場合に発生する．

4.3 評価手順

評価実験 1 と評価実験 2 の評価の手順を説明する．

4.3.1 評価実験 1

評価実験 1 の手順を示す．消費電力は Raspberry Pi に電流計を接続して電流値を測定し，バリエーションは変更が発生した時の実行時間を記録する．

(1) 提案手法に従ってモデルからコードを生成し実験環境へ転送．
(2) CPU 使用率を変化させる実験用ソフトウェアを実行した状態で評価ソフトウェアを実行．
(3) 実行時の電流値の測定値，推定値とソフトウェアのバリエーションを記録．
(4) 実行時の消費電力からバリエーション変更が発生すべき時刻を求め，記録した時刻と比較し評価項目 1 から 4 を評価．
(5) 測定値と推定値を比較し消費電力推定誤差を評価．

4.3.2 評価実験 2

評価実験 2 の手順を示す．評価実験 1 で生成された評価ソフトウェアを用いる．

(1) 評価実験 1 と同様に，実験環境へコードを転送．
(2) 自己適応機能を持たないソフトウェアを実行しその時の電流計の測定値を記録．
(3) 評価ソフトウェアをバリエーションが変更しないように単体で実行した時の電流計の測定値を記録．

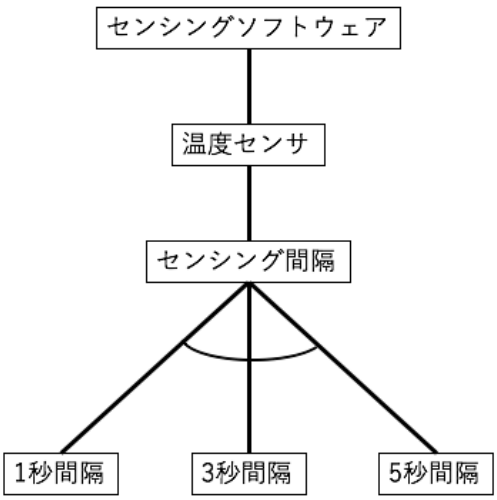

図 6 フィーチャ図
Fig. 6 Feature diagram

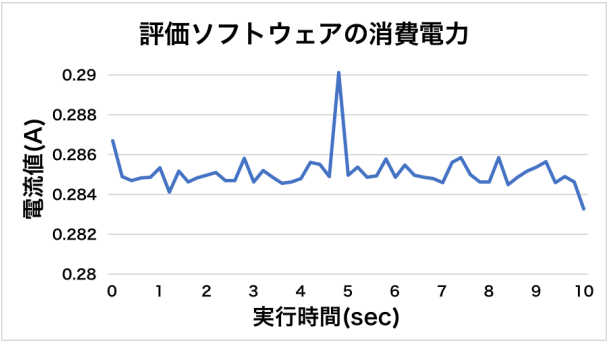

図 7 評価ソフトウェアの消費電力
Fig. 7 Power consumption of the evaluation software

(4) 二つの測定値の記録を比較し消費電力の上昇率を評価.

4.4 実験に用いるソフトウェア

評価実験 1 では提案手法を用いて開発された評価対象ソフトウェアと，端末上の消費電力状況を変化させるために CPU 使用率を変化させるテストプログラムを用いる．それぞれのソフトウェアの仕様を述べる．

4.4.1 評価対象ソフトウェア

評価ソフトウェアは温度センサによるセンシングを行う．Raspberry Pi の消費電力をコンテキストとして自己適応を行い，適応のバリエーションとしてセンシング間隔を扱う．評価ソフトウェアは 1 秒，3 秒，5 秒の三種類のセンシング間隔をバリエーションとして持つ．三種類のバリエーションの閾値には 0.29(A) と 0.31(A) を設定した．図 6 に評価ソフトウェアのフィーチャ図を示す．

評価ソフトウェアのバリエーションを変更させずに，センシング間隔を 3 秒間隔で実行した時の消費電力を図 7 に示す．

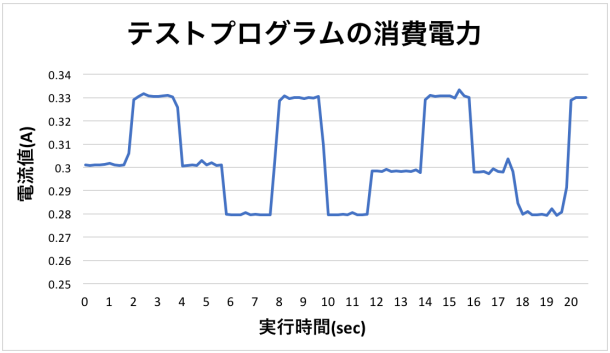

図 8 テストプログラムの消費電力
Fig. 8 Power consumption of the test program

4.4.2 テストプログラム

評価ソフトウェアがコンテキストとして取り扱う端末の消費電力を変化させるために CPU 使用率を変化させる．これは図 4 に示す「ソフトウェア以外で消費される電力」の部分を変化させるためのソフトウェアである．

評価実験に用いた電力モデルでは CPU 使用率と Wi-Fi 通信量をパラメータとしているが，テストプログラムは端末の消費電力を変化させることを目的としたプログラムであるため，簡単のために CPU 使用率のみを変化させている．

評価ソフトウェアが三つのバリエーションを持つため．テストプログラムは 0.2 秒間の平均電流値を三段階に変化させそれぞれ 2 秒ずつ実行する．図 8 にテストプログラムを単体で実行した時の消費電力を示す．

4.5 評価結果

本項では評価実験の結果を述べる．表 5 に表 4 の評価項目の評価結果を示す．実験の結果，応答時間は平均で 0.22 秒であった．テストプログラムを 80 秒実行する間のミスは 11 回であり，ミスに対する修正時間は平均 0.20 秒であった．消費電力状況に対して想定するバリエーションである時間的割合は全体として 87.6%であった．実験時の消費電力に対する消費電力推定の相対誤差は平均 1.52%であった．実験時の実測消費電力と推定消費電力の比較を図 9 に示す．提案手法を用いて自己適応性を持つことによるオーバーヘッドを評価した．評価ソフトウェアと自己適応性を持たないセンシングソフトウェアを実行した時の消費電力をそれぞれ測定し比較したところ，ソフトウェアの消費電力の上昇率は 2.12%であった．図 10 にオーバーヘッドの評価結果を示す．

評価結果から，適応率は 87.6%であり適応性能は十分であると考える．適応性能を追加したことによる消費電力のオーバーヘッドは 2.12%で，許容範囲内のオーバーヘッドである．

表 5　評価結果
Table 5　Evaluation result

評価項目	評価結果
応答時間 (sec)	0.22
ミス頻度 (回/sec)	0.14
修正時間 (sec)	0.20
適応率 (%)	87.6
推定誤差 (%)	1.52
オーバーヘッド (%)	2.12

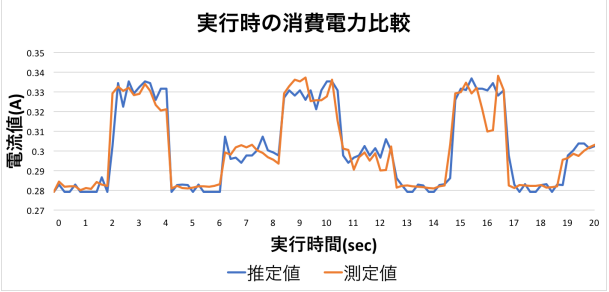

図 9　実行時の実測値と推定値の比較
Fig. 9　Comparison between actual value and estimate at runtime

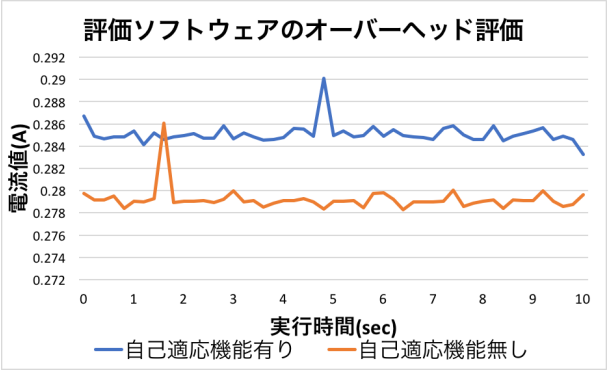

図 10　評価ソフトウェアのオーバーヘッド評価
Fig. 10　Evaluation of overhead

5. おわりに

はじめに，組込みシステムにはハードウェアに制限があることが多くこの制限にともなって稼働時間の要求を満たすため消費電力削減の必要があるという背景を述べた．しかし、消費電力とサービス品質にはトレードオフがありこれらの両立の要求がソフトウェア開発に存在する．消費電力とサービス品質の両立を達成するために，周囲の状況に応じて実行時に振る舞いを変更できる自己適応ソフトウェアを利用すること述べた．しかし，既存のモデルベースによる自己適応ソフトウェアは端末の消費電力のように実装に近い情報を対象に適応することができないという問題があった．これはモデルという上流段階で電力についての情報を取り扱うことができなかったからであった．そこで本論文ではモデルベースの消費電力推定手法を用いてモデルの段階で電力についての情報を取り扱うことができるようにし，モデルベースで消費電力自己適応型ソフトウェアを開発する手法を提案した．本手法では適応のバリエーションを選択性のあるフィーチャとしてソフトウェアプロダクトラインの手法を取り入れ，フィーチャモデルと UML モデルを紐付けた．択一のフィーチャの活性・不活性によってソフトウェアの振る舞いを決定し，実行時には端末の消費電力をコンテキストとして自己適応を行うことが可能である．この手法によって要求される最大消費電力で実行するとともにサービス品質を維持することができ消費電力とサービス品質の両立が可能である．

BridgePoint を用いてモデル図の作成を行い提案手法によって開発したソフトウェアを端末の消費電力を変化させるテストプログラムと同時に実行して評価を行った．評価の結果，平均応答時間 0.22 秒，適応率 87.6%であり消費電力推定の平均誤差は 1.52%という結果が得られた．また，自己適応機能を付加したことによるオーバーヘッドを評価したところ消費電力の上昇率は 2.12%であった．

本研究における今後の課題として，以下のようなものが考えられる．

- 応答速度，適応率の向上
- 可変点を複数持つソフトウェアでの評価
- 消費電力とパフォーマンスのトレードオフ関係の最適化
- オプションフィーチャと UML モデルの紐付け

参考文献

[1] F. D. Macías-Escrivá, R. Haber, R. del Toro, and V. Hernandez, ”Self-adaptive systems: A survey of current approaches, research challenges and applications,” Expert Systems with Applications 40. 18, pp.7267-7279, 2013.
[2] J. A. Estefan, ”Survey of model-based systems engineering (MBSE) methodologies,” Incose MBSE Focus Group 25. 8, 2007.
[3] K. Pohl, G. Böckle, and F. J. van der Linden, ” Software Product Line Engineering: Foundations, Principles and Techniques,” Springer-Verlag , 2005.
[4] S. J. Mellor, M. Balcer, and I. Jacoboson, ”Executable UML: A foundation for model-driven architectures, “ Addison-Wesley Longman Publishing Co., Inc., 2002.
[5] G. H. Alfred, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz, ”Dynamic adaption of service compositions with variability models,” The Journal of Systems and Software, pp.24-47, 2014.
[6] V. R. Basili, G. Caldiera, and H. D. Rombach, ”Goal question metric paradigm,” Encyclopedia of Software Engineering 1, pp.528-532, 1994.
[7] R. Yoshimoto, T. Kadono, K. Hisazumi, and A. Fukuda, ”A Software Energy Analysis Method Using ExecutableUML,” IEEE TENCON 2016, pp.218-221, 2016.
[8] BridgePoint ホームページ，https://xtuml.org/(最終アクセス日 2017/2/9)