DFEAM: Dynamic Feature-oriented Energy-aware Adaptive Modeling

Fumiya Tanaka, Kenji Hisazumi and Akira Fukuda
Kyushu University, Fukuoka, Japan

Keywords: Embedded System, Self-adaptive Software, Model-driven Development, xtUML, Feature-oriented.

Abstract: There is an increasing demand for reducing the power consumption in the field of embedded-system development. A development methodology, which can change software's power consumption according to the power consumption of the hardware, can help fulfill this requirement. However, there will be a trade-off between the power consumption and service quality, which must be balanced for efficient operation. In this paper, we propose dynamic feature-oriented energy-aware adaptive modeling (DFEAM), which develops self-adaptive software through model-driven development for achieving a proper balance between the power consumption and quality of service (QoS). In this method, the application itself decides its behavior, according to the power-consumption situation, by linking the feature model describing the variability of the application with the description of its behavior, using the executable and translatable unified modeling language (xtUML). For achieving a satisfactory QoS for variations that are complex and dependent on variable points, a model is created to quantify the QoS values, which is then used as an index of comparison for finding the optimum variation. We conducted case studies on applications with multiple variable points, and evaluated them using the GQM model. The results of the evaluation showed that the adaptation incorporated provided the maximum software quality under the given power limitations, thus verifying the usefulness of the proposed DFEAM method.

1 INTRODUCTION

Size limitations and high manufacturing costs are common issues in embedded-system development. In the case of battery-driven devices, the battery size is restricted by the hardware size. This scales down the battery storage capacity and shortens the operating time. The operating time is also shortened when the embedded system provides multiple functions and is used in many types of scenarios. In this context, the power consumption needs to be reduced, as much as possible, to meet the uptime requirements. However, this will lead to a tradeoff between the power consumption and the quality of service (QoS). In embedded software development, there is a need to reduce the power consumption while, simultaneously, improving the QoS.

Self-adaptive software (Macías-Escrivá et al., 2013; Mens et al., 2017; Weyns et al., 2012) development is one of the solutions to this challenge. Self-adaptive software can change its behavior according to the runtime circumstances, and is widely recognized to be effective in dealing with complex and dynamic software requirements. As the runtime circumstances change constantly, a developer cannot describe the best behavior, during the development phase. The application software in an embedded system should automatically behave in a manner appropriate for the current environment, because not all possible patterns can be anticipated in the design phase.

In this research, we propose a modeling method named dynamic feature-oriented energy-aware adaptive modeling (DFEAM), which realizes self-adaptive software that achieves both reduction of power consumption and maintenance of QoS. The proposed method has two main elements: the behavior model and the adaptation concept.

The remainder of this paper is organized as follows. The related studies on self-adaptation using models are described in Section 2. In Section 3, the proposed method is explained using a meta model. The content of a case study and its evaluation are explained in Section 4, and finally, the outline of this research is summarized in Section 5.

The contribution in this paper is the proposal of the self-adaptive software development methodology to solve the trade-off between the power limitation of application and QoS, which is a problem in embedded system development.

Tanaka, F., Hisazumi, K. and Fukuda, A.
DFEAM: Dynamic Feature-oriented Energy-aware Adaptive Modeling.
DOI: 10.5220/0007370802900297
In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019)*, pages 290-297
ISBN: 978-989-758-358-2

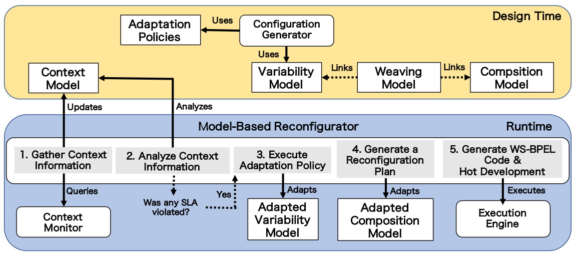

Design Time
Adaptation Policies
Uses
Configuration Generator
Uses
Context Model
Variability Model
Links
Weaving Model
Links
Compsition Model
Updates
Analyzes
Model-Based Reconfigurator
Runtime
1. Gather Context Information
2. Analyze Context Information
3. Execute Adaptation Policy
4. Generate a Reconfiguration Plan
5. Generate WS-BPEL Code & Hot Development
Queries
Was any SLA violated?
Yes
Adapts
Adapts
Executes
Context Monitor
Adapted Variability Model
Adapted Composition Model
Execution Engine

Figure 1: Overview of MoRE-WS (Alfred et al., 2014).

2 RELATED WORKS

In this section, we describe research on self-adaptive research using models and research on power-aware modeling.

In (Alfred et al., 2014), MoRE-WS, which is a framework for dynamic adaptation of web services, was proposed. This framework adapts web services by combining multiple models including a variable model. The framework supports the modeling of adaptive models at design time and performs adaptive execution at runtime. Figure 1 shows an overview of the MoRE-WS framework. The upper part of Figure 1 shows the support provided by the framework during the design phase. MoRE-WS supports the modeling of feature models and web-service configuration models, including those with variability, at design time and the generation of adaptation rules. The variable feature model is mapped to the web-service configuration model, and the service configuration is changed according to the configuration of the feature model. At runtime, the framework adapts the models modeled during design time, according to the adaptation rules and context changes detected by the context monitor. Properties of the web-service operation are handled in the context. Figure 1 shows the adaptation flow. The adaptation is realized by generating a reconfigured plan of the variable feature model, according to the changes in the context, and reflecting it on the code of WS-BPEL.

In (Abdallah et al., 2017), a model-driven power consumption reduction approach in SoC (System-on-Chip) design has been proposed. In SoC design, reducing power consumption is a major concern. However, a method known as an effective method is required to decide architecture configuration and power management technology at an early stage of design. They model power estimation parameters and dynamic power management to obtain power results at an early stage of design. By generating and simulating a power-aware simulation code from the model, it is possible to obtain the power result at the design time. In the proposed method, they model an application model describing the dynamic behavior of an application in an activity diagram and a power management model that summarizes architecture parameters and algorithms. After that, each model is converted into C++ simulation code and then linked. power results at the early stage of design can be obtained by simulation, power consumption estimation and analysis. Power consumption is estimated from the power management model using a known power characteristic model of processor.

In the related research that we have mentioned so far, it is realizing the application at the time of execution or the application for the electric power by using the model. However, QoS-aware and energy-aware self adaptation method using xtUML has not yet been proposed.

3 DFEAM

We propose a method that can realize self-adaptation, based on the power consumption and software quality, using xtUML. Based on model-driven architecture. It is capable of testing at the design stage and performance measurement and strongly supports model-driven development. The system specifications described by xtUML can be converted into source code, irrespective of the platform.

A methodology for self-adaptive software development, based on the power consumption, using a model-driven development and xtUML has already been proposed (Tanaka et al., 2017). In this method, the application itself can decide the behavior according to the power-consumption situation by linking the feature model describing the variability of the application with the description of behavior, using xtUML. However, it is not possible to compare the qualities of the complicated variations and find the variation that can solve the tradeoff between power consumption and quality. Therefore, in DFEAM method, we create a quantitative QoS model to compare the quality of variations and use it as an indicator of optimal variation determination.

The system composition of the proposed method is shown in Figure 2. The proposed method consists of two elements: behavior models based on xtUML and self-adaptation concept based on the concept of MAPE-K (Kephart and Chess, 2003). MAPE-K is a control loop model and includes a monitor, an analyzer, a planning component, and an execution component. In the DFEAM concept, *Monitor* mainly performs the role of the monitor and analyzer, and *Manager* performs the role of the planning and execution components. The *Monitor* and *Manager* are described

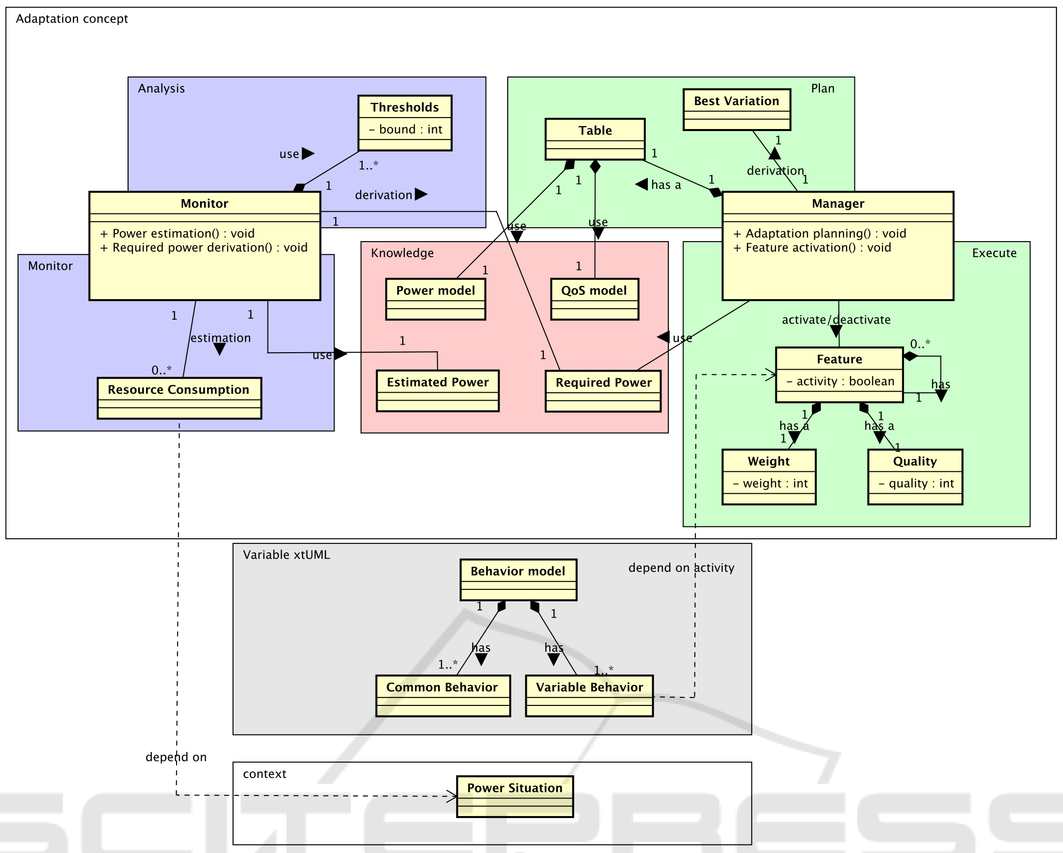


Figure 2: System composition of DFEAM.

by xtUML, and translated into source code by using model-driven approach. In this section, we will explain the development procedure at design time by DFEAM and the adaptation flow at runtime with the system configuration of Figure 2.

3.1 Procedure

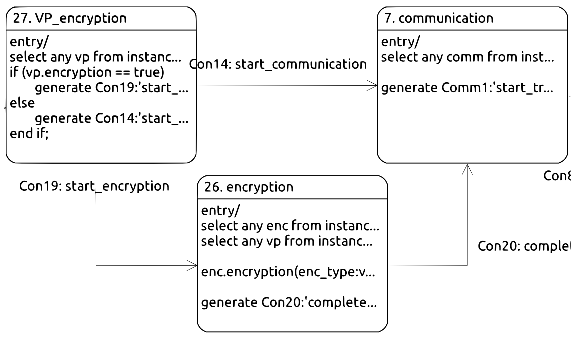
We will describe the self-adaptive software development procedure using DFEAM. It is assumed that the requirements of the application are determined for development.

3.1.1 Variability Analysis

First, we analyze the variability of the application that realizes the requirement and express it with a feature model. The feature model assigns features to software functions and describes the variable functions at runtime as variable features. The feature model consists of common features, Alternative features, Optional features and Parameterized features. The Alternative feature is a feature that selects one, and the Optional feature is a feature that selects valid or invalid. The Parameterized feature is a feature having continuous values as variable points. Since there are Alternative features with Parameterized features and multiple choices, there arises a problem that innumerable variations are present.

3.1.2 xtUML Description

Next, we describe a behavior model (Fig. 2:*Behavior model*) with xtUML. The behavior model consists of *Common Behavior* and *Variable Behavior*. Application behavior is represented by continuous state transitions. Common operations not based on variations are described by one state transition without branch. On the other hand, the variable operation needs to be described so that only the state transition which realizes the behavior selected by the variable point is executed. We use a state transition diagram with variable points for associating features with states. It can set the guard conditions for state transitions, and the state transitions according to the active features can be managed by them. In the variable behavior part, we set a state for each variable point to realize the guard conditions. In the transition management state,



```
27. VP_encryption
entry/
select any vp from instanc...
if (vp.encryption == true)
    generate Con19:'start_...
else
    generate Con14:'start_...
end if;
```

```
7. communication
entry/
select any comm from inst...

generate Comm1:'start_tr...
```

```
26. encryption
entry/
select any enc from instanc...
select any vp from instanc...

enc.encryption(enc_type:v...

generate Con20:'complete...
```

Figure 3: A state machine description at variable point.

a conditional branch is described in an action language, and similar to the guard condition, state transition management corresponding to the active feature is realized. Figure 3 shows a description of the state machine diagram at the variable point related to Optional feature. When *encryption* feature is active, a state transition *start_encryption* is fired. On the other hand, the feature is inactive, another state transition *start_communication* is fired.

The power consumption monitor (*Monitor*) and the variable point manager (*Manager*), which are components of the adaptive concept, are also written in xtUML. First, in xtUML, one variable point class holds the status of all the variable points held by the application. The status of the variable point changes according to the context. Therefore, a mechanism for determining the optimum variation from the dynamic power consumption is necessary. The role of the variable point manager is to determine the optimum variation based on the required power consumption derived by the monitor and is to update the status of the variable point according to the estimated power consumption. This manager is handled as another class in xtUML. The power consumption monitor, which is another component is a mechanism for acquiring the dynamic power consumption. The monitor performs the role of deriving the required power consumption of the software (*Required Power*) according to power limitation (*Thresholds*). It is described as part of the state machine diagram held by the variable point manager class. We use a simple state machine for power consumption monitoring and describe behaviors to periodically collect resource consumption of devices (*Resource Consumption*). It obtains the estimated power consumption (*Estimated Power*) at runtime by using the power consumption estimation model prepared beforehand from the resource consumption.

3.1.3 Search for Optimum Variation

After describing by xtUML, we create a model for finding indices for searching for optimal variation. The main novelty of this method is that it is possible to add QoS consideration to self adaptation by comparing the quality of each variation by creating a QoS model on a common scale. Figure 4 shows the relationship between the software and the factors affecting its quality. This is a metamodel of the manager's planning and execution action in Figure 2. The QoS depends on the feature's weight, quality function, and activity. The feature's activity affects the application's behavior via the transition constraint.

The QoS model (*QoS model*) is created using weights (*Weight*) and quality functions (*Quality*) for each variable feature. The weight is a value indicating how important the variable function corresponding to the variable point is as the function of the application. Quality function is a function expressing the influence of variable point on QoS. We normalize the effect on software quality by variability change from 0 to 1. These weights and quality functions are values whose designers arbitrarily reflect the degree of importance and the degree of influence on the quality in service of variable functions. The QoS model is created by reflecting the tree structure of the feature tree with weights and quality functions. In this model, the QoS value of the entire application is obtained by following the tree recursively from root to leaf. The formulation reflecting the tree structure will be described below.

We define the QoS value for a feature tree whose root is feature k as Q_k, therefore a QoS value of the entire application is Q_{root}. Feature k has n_k number of child features ($n_k \geq 0$). When feature k has any child feature, we specify its child feature as feature k_i ($1 \leq i \leq n_k$). QoS model for the entire application is as follows.

$$Q_{root} = \sum_{i=1}^{n_{root}} Q_{root\_i}$$

Q_{root} :	the QoS value for entire application
$root\_i$:	i th child of root feature

In addition, variable feature k has variability v_k, weight w_k and quality function $F_k(v_k)$. The domain of v_k and the expression of F_k depend on the type of feature. The quality functions $F_k(v_k)$ of the Alternative feature and Parameterized feature are a function arbitrarily set by the developer. The function of the Optional feature is a function that returns 0 or 1. For simplicity we assume that the Parameterized feature is a leaf of the feature tree. We formulate each

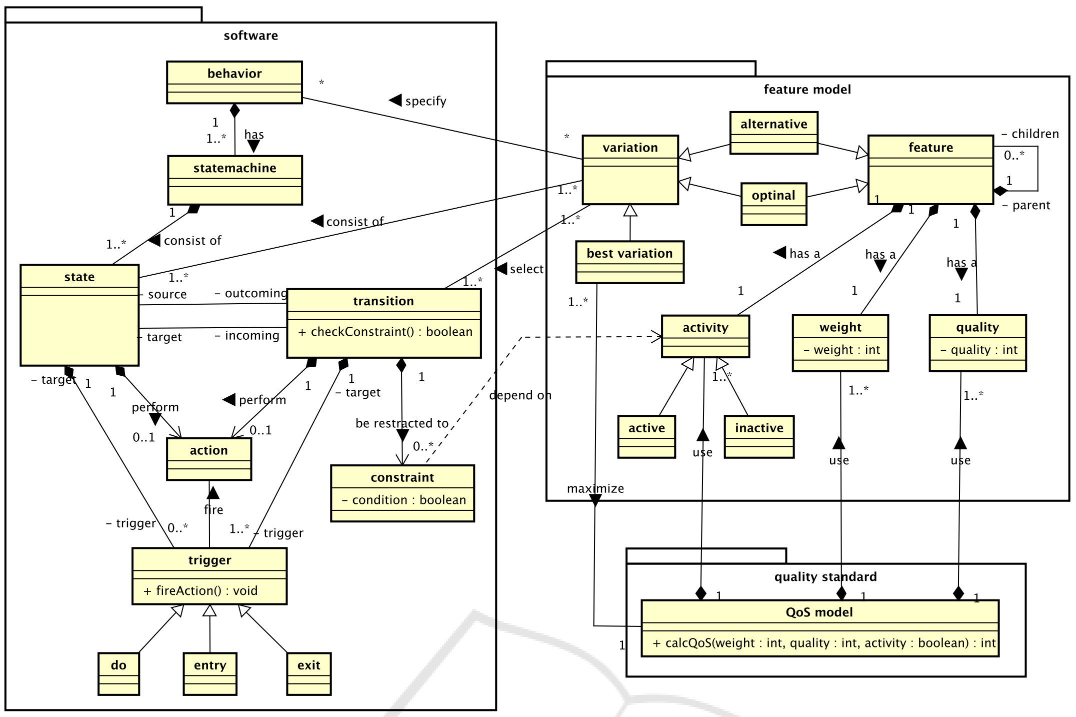


Figure 4: A metamodel of relation between application and quality standard.

of the variable features with the characteristics Common, Alternative, Optional and Parameterized.

$$
\begin{aligned}
Com: &\quad Q_k = \sum_{i=1}^{n_k} Q_{k\_i} \\
Alt: &\quad Q_k = w_k F_k(v_k) + Q_{k\_v} \qquad v_k = \{1, 2, \ldots n_k\} \\
Opt: &\quad Q_k = F_k(v_k)(w_k + \sum_{i=1}^{n_k} Q_{k\_i}) \qquad v_k = \{0, 1\} \\
Par: &\quad Q_k = w_k F_k(v_k) \qquad (v_k \geq 0)
\end{aligned}
$$

Q_k :	the QoS for a feature k tree
k_i :	i th child of feature k
w_k :	the wight of feature k
$F_k(v_k)$:	the quality function of feature k

Next, we create a power model to estimate the power consumption of each variation. This model is a polynomial with each variable point as a parameter. We perform model fitting using a realistic number of variations obtained by randomly changing values. The formula for creating a Power model for an application with n variable points v_i is as follows.

$$Power = C + \sum_{i=1}^{n} a_i v_i$$

We describe the procedure of searching for the optimal variation. First, QoS value and estimated power consumption of all variations are obtained using QoS model and Power model. Next, all the variations are sorted by the estimated power consumption and grouped by the estimated power consumption within a certain range. The range depends on the power consumption required at runtime. The developer extracts variations with the highest QoS value in the group for the group in which the variation corresponding to the range exists in the group. For the extracted representative variation group, if there are other variations with a higher QoS value with lower power consumption, the variation is excluded from the representative variation group. The finally obtained variation is used for self application.

3.2 Adaptation Flow

We describe the behavior change flow of an application developed using DFEAM at runtime. At runtime, the power consumption monitor operates in parallel with the behavior of realizing the application. The power consumption monitor periodically estimates the power consumption at the time of execution by acquiring the resource consumption, and the power consumption required for the application is determined from the estimated power consumption and the power limit. The variable point manager selects variations that meet the requirements from the adopted variations and updates the status of variable points. Since the state transition permitted by changing the status of the variable point changes, the operation of the application also changes. Therefore, it is possible to change the behavior of the application according to the power consumption. In addition, the behavior at that time is changed so as to be optimal in the QoS value according to the defined criteria.

4 CASE STUDY

We do case studies to demonstrate the usefulness of DFEAM. We modeled the variability xtUML using BridgePoint. BridgePoint can automatically generate code from xtUML and is utilized as a tool in model-driven development. In BridgePoint, developers describe class diagrams and can set state machines in classes where behavior exists. Furthermore, if there is behavior within the state, the detailed behavior can be described by using an action language.

4.1 Case Study Design

We designed an application that acquires and stores simple environmental information, as a case study. The outline of the application is presented below.

> The application obtains environmental information using a temperature sensor or an illuminance sensor operating on a battery-driven device. The sensing intervals of both sensors are the same. The environmental information is obtained by converting the acquired data into temperature and illuminance and adding a time stamp. The information is accumulated locally on the device and is sent to the server after a certain number of sensing operations. The server holds the data transmitted from the device and provides services using real-time environmental information. It is conceivable that the server communicates with a plurality of devices.

4.2 Implementation

The feature model of the case study is shown in Figure 5. We describe behavior model, power consumption monitor and variable point manager with xtUML.

We set the weights and quality functions for the variable features of the application as shown in Table 1. For SensingInterval, TransmitSize, Brightness, it is defined by the step function shown in (a)∼(c) of Figure 6. When VP Number is i, the variable point value is v_i, the weight is w_i, and the quality function is

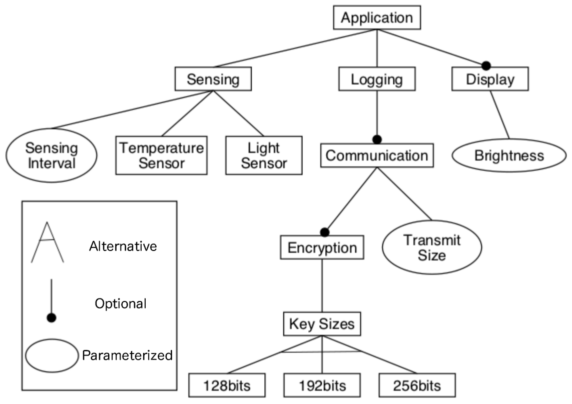


Figure 5: Feature model in case study.

$F_i(v_i)$, the QoS model in the case study is determined as follows. Incidentally, the expression is described in three parts.

$$\begin{aligned} Q_{part1} &= F_1(v_1)\{w_1 + F_2(v_2)(w_2 + w_4F_4(v_4)) \\ &\quad + w_5F_5(v_5)\} \\ Q_{part2} &= w_3F_3(v_3) \\ Q_{part3} &= F_6(v_6)(w_6 + w_7F_7(v_7)) \\ Q_{max} &= \sum_{i=1}^{7} w_i \\ Q_{root} &= \frac{Q_{part1} + Q_{part2} + Q_{part3}}{Q_{max}} \end{aligned}$$

Next, we create a power model to estimate the power consumption of each variation. We perform model fitting using 80 kinds of variations obtained by randomly changing values. The power model equation is shown below.

$$Power = C + \sum_{i=1}^{7} a_i v_i$$

The procedure for finding the optimum variation is as follows. First, we obtain QoS values and estimated power consumption of all variations using QoS model and Power model. Since the quality function of the parameterized feature is discretized when setting the quality function, the number of variations that can be selected practically is 180 in total. A scatter diagram of QoS values and estimated power consumption of 180 all variations is shown in Fig. 7.

Next, all the variations are sorted by the estimated power consumption and grouped by the estimated power consumption within a certain range. In this case study, 26 groups were created every 0.001 A, and there were variations corresponding to the range of 7 groups among them. The table 2 shows the result of extracting the variation with the highest QoS value for the group with variations in the group. From the table, variations of #4 and #6 show that variations with

Table 1: Weight and quality function.

VP Number	1	2	3	4	5	6	7
VP Name	Communication	Encryption	SensingInterval	KeySizes	TransmitSize	Display	Brightness
Weight	10	8	7	6	5	3	1
Quality	0/1	0/1	Fig. 6(a)	0.5/0.8/1	Fig. 6(c)	0/1	Fig. 6(b)

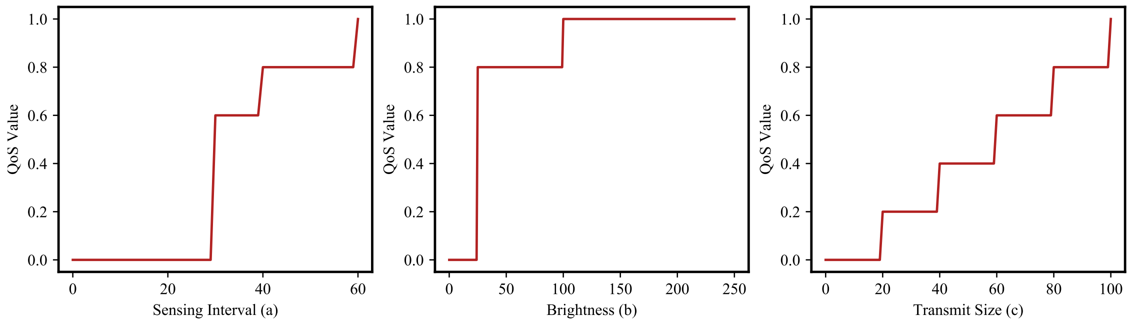


Figure 6: Quality function.

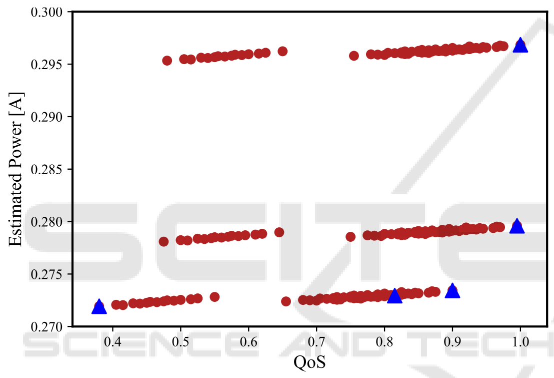


Figure 7: Scatter diagram of QoS and estimated power.

Table 2: Variations with the highest QoS value in each group.

#	Power	QoS
1	0.27193	0.380
2	0.27294	0.815
3	0.27344	0.90
4	0.27896	0.885
5	0.27960	0.995
6	0.29599	0.835
7	0.29685	1.0

higher QoS values with lower power consumption exist in other groups. Since these variations are variations that do not satisfy the requirement of operating with maximum QoS under limited power, these are excluded. As a result, five variations were selected as optimal variations under specific limited power. The five triangular points of the Figure 7 are the five variations obtained as a result of the search.

5 EVALUATION

In this section, we evaluate the usefulness of the DFEAM using the GQM model (Basili et al., 1994), for the five variations obtained in the case study. The GQM model is a measurement framework, which establishes a clear target and associates metrics necessary for achieving a goal. As many previous software measurements have used the GQM model, we decided that it would be suitable for the evaluation in our research (Alfred et al., 2014; Chen et al., 2003). Table 3 shows the GQM model used for evaluation. Q1 indicates the trade-off between the power consumption and QoS, for the five variations selected, and it confirms whether the premise of trade-off optimization is satisfied or not. By confirming this, it can be shown that the arbitrary QoS model used in this method is valid. To estimate Q1, we use the estimated power consumption (M1) and QoS value (M2) of each variation. Q2 meets the requirement to develop software that provides the maximum QoS under a certain limited power. We use the QoS value (M 3) of each variation to evaluate Q2.

Figure 7 shows the estimated power and QoS values of all variations. In Figure 7, the five triangle points represent the selected variations. For Q1, we can see that the QoS value is higher for variations with larger power consumptions than in the figure, and the trade-off relationship between the power consumption and service quality does not disappear in the selected five variations. Therefore, the prerequisite for the optimization of the trade-off relation, which is the main aim of this method, is satisfied, and it can be said that the selected variation group is a reasonable group. Furthermore, with respect to Q2, since the

Table 3: GQM model.

Goal	Question	Metrics
Adaptation is appropriate	Is there a trade-off in the selected variation group?(Q1)	Estimated power(M1)
		QoS value(M2)
	Does it provide maximum QoS under limited power? (Q2)	QoS value(M3)

trade-off relationship of the selected variation group can be confirmed, the higher the power consumption is, the higher the QoS value will be. Therefore, QoS is also the maximum when selecting one variation that uses the maximum power consumption under a limited power. From the above results, it can be said that Q1-2 is achieved and appropriate adaptation is possible. Thus, we show that this method is useful as a self-adaptation methodology to optimize the tradeoff.

In this case study, it is shown that the runtime variation determined using DFEAM provides the maximum QoS at the priority defined under the power limitation. However, it is impossible to show how much effect is expected for the demand of power-consumption reduction. In addition, DFEAM arbitrarily determines the importance of functions by taking the developers' use cases of applications into consideration. Therefore, the selected variations are subject to the constraints of the developers and use cases.

6 CONCLUSION

First, we stated that embedded systems have restrictions on the hardware in many cases and according to these restrictions. It is necessary to reduce the power consumption to meet the operating time requirements. However, there is a tradeoff between power consumption and QoS, and there are demands for solutions to balance these, in software development. In this research, we proposed a self-adaptive modeling method, DFEAM, which realized software that performed self-adaptation to solve this trade-off, using model-driven development based on xtUML. The proposed method modeled an executable model that performed self-adaptive behavior to solve a trade-off by searching for an optimal variation using a self-adaptive concept incorporating the concept of MAPE-K and a model showing quantitative QoS. In a case study, this method was applied to applications with multiple variable points and evaluated using the GQM model. From the evaluation, it was observed that appropriate adaptation had been made to provide the maximum software quality under the given power limitations. Therefore, it can be said that the DFEAM is useful as a self-adaptive method to optimize the trade-off between the power consumption and the QoS.

ACKNOWLEDGEMENTS

This work was supported by JSPS KAKENHI Grant Number 15H05708.

REFERENCES

Abdallah, F. B., Trabelsi, C., Atitallah, R. B., and Abed, M. (2017). *Model-Driven Approach for Early Power-Aware Design Space Exploration of Embedded Systems*, volume 87.

Alfred, G. H., Pelechano, V., Mazo, R., Salinesi, C., and Diaz, D. (2014). *Dynamic adaption of service compositions with variability models*, volume 91.

Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). Goal question metric paradigm. In *Encyclopedia of Software Engineering 1*, pages 528–532.

Chen, T., Far, B. H., and Wang, Y. (2003). Development of an intelligent agent-based gqm software measurement system. In *Proceedings of the ATS 2003 Conference*, pages 188–197.

Kephart, J. and Chess, D. (2003). *The vision of autonomic computing*, volume 36.

Macías-Escrivá, F. D., Haber, R., del Toro, R., and Hernandez, V. (2013). *Self-adaptive systems: A survey of current approaches, research challenges and applications*, volume 40.

Mens, K., Capilla, R., Hartmann, H., and Kropf, T. (2017). *Modeling and Managing Context-Aware Systems' Variability*, volume 34.

Tanaka, F., Hisazumi, K., Ishida, S., and Fukuda, A. (2017). A methodology to develop energy adaptive software using model-driven development. In *Proceedings of the 2017 IEEE Region 10 Conference (TENCON)*, pages 769–774.

Weyns, D., Iftikhar, M. U., Gil, D., and Ahmad, T. (2012). A survey of formal methods in self-adaptive systems. In *C3S2E*, pages 67–79.