Vol.2019-ARC-235 No.11
Vol.2019-SLDM-187 No.11
Vol.2019-EMB-50 No.11
2019/3/17

優先度付コマンドインタフェースによる組込み制御ソフトウェアの構造化手法

藤平達[†1]　趙立晴[†1]　吉田泰三[†2]　山田裕樹[†2]　岩崎力[†2]

概要：組込み制御システムは，年々，多機能化が進んでいる．システムの複数の機能が同一のアクチュエータを制御する為，ソフトウェアで機能間の競合を解決する必要がある．一般的な競合解決手段として，OS によるタスクスケジューリングや共有資源排他制御等の機構が知られているが，各機能の優先度がアクチュエータによって変化する場合，対応が困難であり，ソフトウェアが複雑化する傾向にある．そこで，本稿では，各機能モジュールからアクチュエータへ優先度を付したコマンドインタフェースを設け，競合解決を行うソフトウェア構造化手法を提案する．本手法を空調機制御ソフトウェアについて試行した結果，複雑度を低減できることを確認した．

キーワード：ソフトウェア構造化，機能競合解決，組込み制御システム

Structuring Method of Embedded Control Software by Prioritized Command Interface

TORU FUJIHIRA[†1]　LIQING ZHAO[†1]
TAIZO YOSHIDA[†2]　YUKI YAMADA[†2]　CHIKARA IWAZAKI[†2]

Abstract: Multi-functionalization of embedded control system has been progressing over time. It is more often the case that multiple functions of system control single actuator and software needs to resolve conflicts among functions. In general OS features such as task scheduling and exclusion control of common resource are known as conflict resolution methods. But in case that priority of each function varies according to actuator to control, it is difficult to apply OS features and software tends to be complicated. Therefore, we propose software structuring method which resolves conflicts by utilizing prioritized command interface from each function modules to actuator modules. We tried to apply the method to control software of air conditioner and confirmed that complexity can be reduced.

Keywords: Software structuring, Function conflict resolution, Embedded control system

1. はじめに

携帯機器，家電製品，産業機器，輸送機器等の広い分野で用いられる組込みシステムは，IoT 化等の高付加価値化のための新機能が追加されるようになり，高機能化・多機能化が進んでいる．組込みシステムの機能を実現する組込みソフトウェアは，大規模化・複雑化が著しく，開発費全体の約４割を占めている[1]．そのため，ビジネス観点からソフトウェア開発プロジェクトの信頼性及び生産性の向上の両立が求められている[2]．

信頼性及び生産性の向上には多くのアプローチが考えられるが，その１つはソフトウェア構造の改善による複雑度の低減である．一般的にはソフトウェアを疎結合かつ高凝集な構造[3]とすることにより保守性や再利用性を向上し，不具合発生の低減と開発期間の短縮を図る．例えばプラットフォームの導入やアーキテクチャの見直し等の対応がとられる．

しかしながら，一口に組込みシステムといっても，応用分野によって大きく特性が異なっている[4]．対象システムの特性に応じた，より具体的なソフトウェア構造化手法が望まれる．組込みシステムの大まかな枠組みとして，ネットワーク通信や画面操作を主とする組込み情報システムと，センサやアクチュエータを用いて物理対象を制御する組込み制御システムがある．本稿では，組込み制御システムの制御を担う組込み制御ソフトウェアを対象とした構造化手法を検討する．

組込み制御ソフトウェアの構造に着目すると，複雑化の一因として，システムのアクチュエータを制御する機能が多数に上ることが挙げられる．同一のアクチュエータを制御する複数の機能が同時に有効となる場合，所定の優先度に従ってソフトウェアで機能間の競合を解決する必要がある．多機能化が進むほど，優先度解決のためのロジックが複雑化していく傾向にある．そこで，本稿では，各機能モジュールからアクチュエータ制御モジュールへ優先度を付したコマンドインタフェースを設け，競合解決を行うソフトウェア構造化手法を提案する．

以下，2 章では組込み制御システムにおける機能競合について述べる．3 章で多機能化による組込み制御ソフトウェアの複雑化について述べる．4 章で複雑度低減について

†1 株式会社日立製作所
Hitachi, Ltd.
†2 日立ジョンソンコントロールズ空調株式会社
Hitachi-Johnson Controls Air Conditioning, Inc.

Vol.2019-ARC-235 No.11
Vol.2019-SLDM-187 No.11
Vol.2019-EMB-50 No.11
2019/3/17

検討する．5 章で優先度付コマンドインタフェースによるソフトウェア構造化手法について述べ，6 章で空調制御ソフトウェアに適用試行した結果について述べる．

2. 組込み制御システムにおける機能競合

2.1 組込み制御システム

図 1 に組込み制御システムの概念例を示す．図中のブロック（方形）は実体を，楕円は処理を，接続線はデータの流れを表している．コントローラは MCU[a]等によりセンサから取得した制御対象の物理量などの情報を用いて制御対象への処理を決定する．そして，決定した制御処理に応じてアクチュエータを動作させることにより制御対象への操作を行う．

1 章で述べたように近年の組込み制御システムは，多数のセンサ･アクチュエータと機能を備える傾向がある．図 1 では，簡単な例として各 3 つのセンサ，アクチュエータ，機能を備えたシステムを示している．機能毎に利用するセンサ・アクチュエータは異なるが，重複している場合が多い．図 1 の例では，機能 A･C がセンサ U の入力を共有し，全ての機能 A・B・C がアクチュエータ X を制御する．

2.2 機能状態

多くの場合，機能は，状態を持ったステートマシンである．機能の状態は，例えば，単純な場合は，機能の有効／無効状態などである．各機能の状態に応じて制御処理内容は異なっている．図 1 の組込み制御システムにおける機能と，その状態に応じた制御処理の例を表 1 に示す．

システムに複数の機能がある場合，システム全体の状態（以降，システム状態と呼ぶ）は，それらの機能の状態を組み合わせたものとなる．ある機能の状態と同時に起こり得る別の機能の状態との組合せにおいて，各々の制御処理を実行する．

2.3 機能競合

システム状態における複数機能の制御処理が同一のアクチュエータを対象としながら，その動作が機能間で異なる場合，同時に実行することはできない．つまり，アクチュエータで機能競合が発生することになる．機能競合を回避するために，優先度を決めて，どちらかの制御処理を選択して実行する必要がある．

表 2 に機能競合における優先度の例を示す．機能毎に制御するアクチュエータは異なっており，競合は，アクチュエータ単位で発生する．また，機能の主目的である安全性や快適性などへの各アクチュエータ動作の寄与の度合いが，機能の状態によって変動する．そのため，優先度は，機能そのものではなく，各々のアクチュエータに対して決まる．

各アクチュエータの優先度からシステム状態におけるアクチュエータ制御が一意に決まる（表 3）．

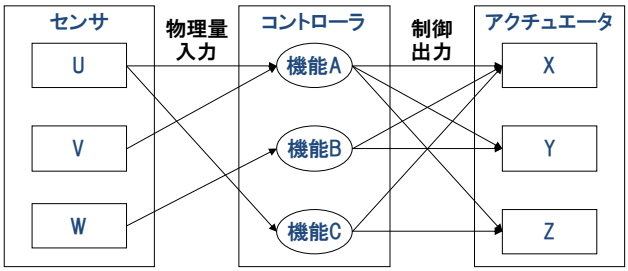

図 1　組込み制御システムの例

Figure 1　An example of embedded control system

表 1　機能状態による制御処理の例

Table 1　An example of control process by features' status

機能	機能状態	アクチュエータ制御		
		X 動作	Y 動作	Z 動作
A	A0	X0	Y0	Z0
	A1	X1	Y1	Z1
	A2	X2	Y2	Z2
B	B0	なし	なし	なし
	B1	X3	Y3	なし
C	C0	なし	なし	なし
	C1	X4	なし	Z3

表 2　機能競合における優先度の例

Table 2　An example of feature conflict resolution priority

優先度	アクチュエータ X		アクチュエータ Y		アクチュエータ Z	
	機能状態	動作	機能状態	動作	機能状態	動作
1	A0	X0	A2	Y2	A1	Z1
2	B1	X3	B1	Y3	C1	Z3
3	A2	X2	A1	Y1	A0	Z0
4	C1	X4	A0	Y0	A2	Z2
5	A1	X1	-	-	-	-

表 3　組合せ状態における制御処理の例

Table 3　An example of control process by combination status

組合せ状態	アクチュエータ制御		
	X 動作	Y	Z 動作
A0/B0/C0	X0 (A0)	Y0 (A0)	Z0 (A0)
A0/B0/C1	X0 (A0)	Y0 (A0)	Z3 (C1)
A0/B1/C0	X0 (A0)	Y3 (B1)	Z0 (A0)
A0/B1/C1	X0 (A0)	Y3 (B1)	Z3 (C1)
A1/B0/C0	X1 (A1)	Y1 (A1)	Z1 (A1)
A1/B0/C1	X4 (C1)	Y1 (A1)	Z1 (A1)
A1/B1/C0	X3 (B1)	Y3 (B1)	Z1 (A1)
A1/B1/C1	X3 (B1)	Y3 (B1)	Z1 (A1)
A2/B0/C0	X2 (A2)	Y2 (A2)	Z2 (A2)
A2/B0/C1	X2 (A2)	Y2 (A2)	Z3 (C1)
A2/B1/C0	X3 (B1)	Y2 (A2)	Z2 (A2)
A2/B1/C1	X3 (B1)	Y2 (A2)	Z3 (C1)

a) Micro Controller Unit

Vol.2019-ARC-235 No.11
Vol.2019-SLDM-187 No.11
Vol.2019-EMB-50 No.11
2019/3/17

2.4 機能競合解決の要件

組込み制御システムにおける機能競合の特徴を踏まえた機能競合解決の要件を以下に示す.

(1) アクチュエータ毎の機能優先度設定

アクチュエータ毎に機能の優先度が設定可能であることが必要である.

(2) 機能状態遷移に伴う機能優先度変更

アクチュエータに対する機能の優先度は，機能状態に応じて変更可能であることが必要である.

3. 多機能化による組込み制御ソフトウェアの複雑化

本章では，組込み制御システムにおける機能競合解決のための一般的なソフトウェア構造と，多機能化による複雑化の詳細について述べる.

3.1 機能競合解決のための一般的なソフトウェア構造

(1) アクチュエータ優先処理に基づく構造

表 2 に示したアクチュエータ毎の機能競合における優先度にしたがって，制御処理を決定するアクチュエータ優先処理モジュールを設ける構造である．アクチュエータ優先処理モジュールは，アクチュエータを制御する各機能の状態を参照し,アクチュエータの制御処理を決定する.図 2 にアクチュエータ優先処理に基づくソフトウェア構造の例を示す．アクチュエータ優先モジュールの処理内容をC言語に則って記述した例を図 3 に示す.

(2) システム状態管理に基づく構造

表 3 に示したように，システム状態に対して各アクチュエータ制御は一意に決まる．したがってシステム状態をシステム全体の単一の状態遷移として扱うシステム状態管理モジュールを設ける[5]ことにより，直接アクチュエータを制御することができる．機能競合はシステム状態管理モジュールで暗黙的に解決される．図 4 にシステム状態管理に基づくソフトウェア構造の例を示す.

3.2 多機能化による複雑化

3.1 節の各構造において機能競合解決に起因するソフトウェアの複雑度について検討する．本稿ではソフトウェアの複雑度の指標として McCabe の循環的複雑度[6]を用いる.

構造(1)では，各機能状態管理モジュールとアクチュエータ優先処理モジュールが対象となる．構造(2)では，システム状態管理モジュールが対象となる．構造(1)の機能状態管理モジュールと構造(2)のシステム状態管理モジュールは，状態遷移を行うモジュールである．状態数やモジュール数は異なるが，モジュール単体の複雑度は同様に計算できるため,これらについては,状態管理モジュールとして扱う.

(1) 状態管理モジュールの複雑度

実際の遷移条件によって複雑度は変化するが，ここでは，状態遷移処理において各状態毎に自状態を含む全状態を遷移先とする条件分岐を行うものとして試算する（図 5）．機

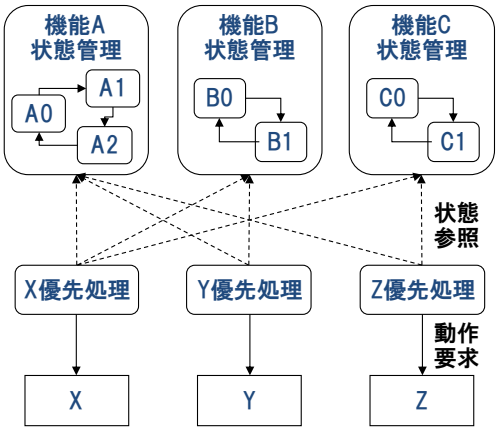

図 2 アクチュエータ優先処理に基づくソフトウェア構造の例

Figure 2 An example of software structure based on actuator arbitration process

```
int priorX(int A,B,C) { // アクチュエータX優先処理
        int X = X0;             // アクチュエータX動作
        if (A == A0) {          // 優先度1の機能状態
                X = X0;
        } else if(B == B1) {    // 優先度2の機能状態
                X = X3;
        } else if(A == A2) {    // 優先度3の機能状態
                X = X2;
        } else if(C == C1) {    // 優先度4の機能状態
                X = X4;
        } else if(A == A1) {    // 優先度5の機能状態
                X = X1;
        }
        return X;
}
```

図 3 アクチュエータ優先処理の例

Figure 3 An example of priority based actuator process

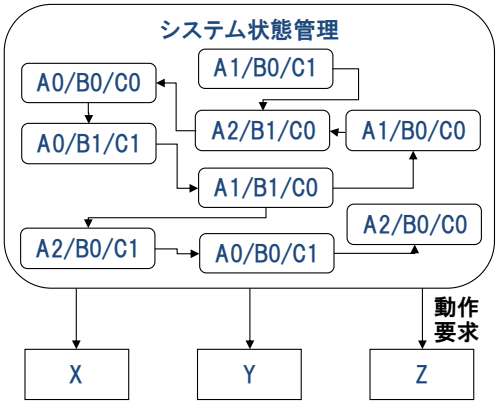

図 4 システム状態管理に基づくソフトウェア構造の例

Figure 4 An example of software structure based on system status management

Vol.2019-ARC-235 No.11
Vol.2019-SLDM-187 No.11
Vol.2019-EMB-50 No.11
2019/3/17

```
int transitionA(int A) { // 機能A状態遷移
        int next;                   // 遷移後の状態
        if (A == A0) {              // A0状態からの遷移条件
                if(…) {
                        next = A2;
                } else if (…) {
                        next = A1;
                } else {
                        next = A0;
                }
        } else if(A == A1) {    // A1状態からの遷移条件
                …
        } else if(A == A2) {    // A2状態からの遷移条件
                …
        }
        return next;
}
```

図 5　状態遷移処理の例
Figure 5　An example of state transition process

能 f の状態数を s(f)とすると，状態管理モジュールの循環的複雑度 M_f は，

$$M_f = \Sigma_f((1 + s(f)) \times s(f))$$

となる．

(2) アクチュエータ優先処理モジュールの複雑度

アクチュエータ a から機能 f の状態 s への参照の有無（有の場合 1，無の場合 0）を r(a,f,s)とすると，アクチュエータ優先処理モジュールの循環的複雑度 M_a は，

$$M_a = \Sigma_a(1 + \Sigma_f \Sigma_s r(a,f,s))$$

となる．

全体の循環的複雑度 M は，M_f と M_a の和となる．M_f，M_a とも，機能数および機能状態数が増えるほど複雑度が増大する．これは，状態遷移の条件分岐や優先度解決のロジックがハードコーディングされることによる．特に M_f では，機能状態数の 2 乗のオーダーで増大する．構造(2)では，システム全体の機能状態の組み合わせにより状態数が爆発的に増大するため，影響が大きい．2 章に示した例では，構造(1)は M = 40，構造(2)は M = 156 である．

4. 複雑度低減の検討

3.2 節で述べたように，組込み制御システムの機能競合解決のための一般的なソフトウェア構造では，ロジックのハードコーディングにより機能数や機能状態数が増えるほど複雑度が増大する．そこで，ロジックの影響を低減する為，汎用的な OS 機構やアルゴリズム・データ構造を組込み制御システムの機能競合解決に利用することが可能か検討した．

4.1 OS 機構の利用

組込み制御システムでは，複数のタスクでシステム資源を効率的に利用するためにリアルタイム OS が使われることが多い．アクチュエータは，システム資源であると捉えられる．そこで，リアルタイム OS の一般的な資源管理機構であるタスクスケジューリングと排他制御について組込み制御システムの機能競合解決に利用できるか検討した．

(1) タスクスケジューリング

OS の主要な機能であるタスクスケジューリングは，CPU 時間資源をタスク間で分配する機構である．タスクスケジューリングには，いくつかの方式があるが，リアルタイム OS では，一般的に図 6 に示す優先度方式が用いられる[7]．各タスクに要求される実時間性に応じた優先度を付与し，待ち行列にキューイングする．スケジューラは待ち行列から優先度の高いタスクを優先して実行する．

組込み制御システムの機能競合解決が優先度に基づくことから，優先度方式タスクスケジューリングを利用することが考えられる．この場合，2.4 節で述べた機能競合解決の要件(1)より，各機能の制御処理をアクチュエータ毎に分割してタスクに割り付ける必要がある．しかし，タスクスケジューラでは，システムの全てのタスク間で優先度によるスケジューリングが実行されるため，アクチュエータ毎の機能優先度を独立させることができない．つまり，本来，アクチュエータ間の関係は並列であり，優先度が無いが，タスクスケジューラによる優先度の影響を受け，特定のアクチュエータのタスクに切り替わらなくなる問題が生じる．

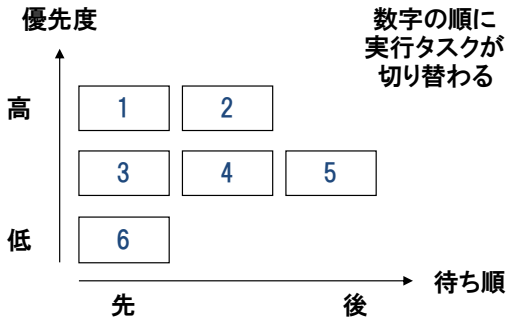

図 6　優先度方式タスクスケジューリング
Figure 6　Priority based task scheduling

(2) 排他制御

タスクの利用するメモリ等の共有資源について，タスク間の競合を避ける仕組みとして排他制御がある[8]．排他制御は 1 つのタスクが共有資源を利用している間は，他のタスクがその共有資源を利用できないようにすることで競合を回避する．

アクチュエータはシステムの共有資源とみなせるので，各機能をタスクに割り付け，機能タスク間で排他制御を行うことが考えられる．表 4 に主な排他制御方式を示す．いずれも排他制御自体は，優先度制御の機構を持たない．したがって，優先度方式タスクスケジューリングと組み合わせて使用する必要がある為，前述の問題が同様に生じる．

Vol.2019-ARC-235 No.11
Vol.2019-SLDM-187 No.11
Vol.2019-EMB-50 No.11
2019/3/17

表 4　排他制御方式

Table 4　List of exclusive control methods

排他制御方式	内容
ロック	資源へのアクセスを制限・解除する機構.
セマフォ	利用可能な資源数と資源獲得を待つタスクの待ち行列を持つ.
ミューテックス	単一資源の場合のセマフォと同等の機能に加えて，排他制御に伴うタスク優先度逆転を防ぐためのプロトコルがサポートされる.

4.2 優先度付待ち行列の利用

一般的に知られている計算アルゴリズム・データ構造に優先度付待ち行列[9]がある．優先度付待ち行列は以下の操作をサポートしている．

(1) **要素の追加**

待ち行列に優先度のついた要素を追加する．

(2) **要素の削除**

待ち行列から最も優先度の高い要素を取り出し，待ち行列から削除する．

(3) **要素の参照**

待ち行列から最も優先度の高い要素を参照する．

(4) **優先度の変更**

待ち行列の指定した要素の優先度を変更する．

操作(1)(2)が優先度付待ち行列の基本機能であり，操作(3)(4)はオプション機能として追加することが可能な操作である．優先度付待ち行列をアクチュエータ毎に用意することで 2.4 節で述べた機能競合解決の要件(1)を満たすことができる．また，操作(4)によって要件(2)を満たすことができる．

優先度付待ち行列は，C++や Java 等の言語で標準ライブラリ化されている[10][11]. 組込み制御システムで主に使用される C 言語では，標準ライブラリ化されていないが，二分ヒープデータ構造やソート等を用いて実装することができる．また，OS に依存しない為，OS を使用しないシステムでも使用することができる．

5. 優先度付コマンドインタフェースによるソフトウェア構造化

4 章の検討結果から，組込み制御システムの機能競合解決に優先度付待ち行列の機能を利用することができ，その結果，ソフトウェアの複雑度を低減することが可能と考えられる．

5.1 ソフトウェア構造

図 2 におけるアクチュエータ優先処理モジュールを，優先度付待ち行列を利用した優先度付コマンドインタフェースモジュールに置換する．図 7 に優先度付コマンドインタフェースに基づくソフトウェア構造の例を示す．本構造では，以下のように動作する．

(1) 機能状態管理モジュールが機能状態に応じたアクチュエータ動作コマンドを優先度とともに優先度付コマンド待ち行列に発行（追加）する．

(2) アクチュエータは，優先度付コマンド待ち行列から最も優先度の高いコマンドを参照し，指定された動作を実行する．

(3) 機能状態管理モジュールにおいて機能状態が遷移した場合，コマンドの優先度と制御処理内容を変更する．

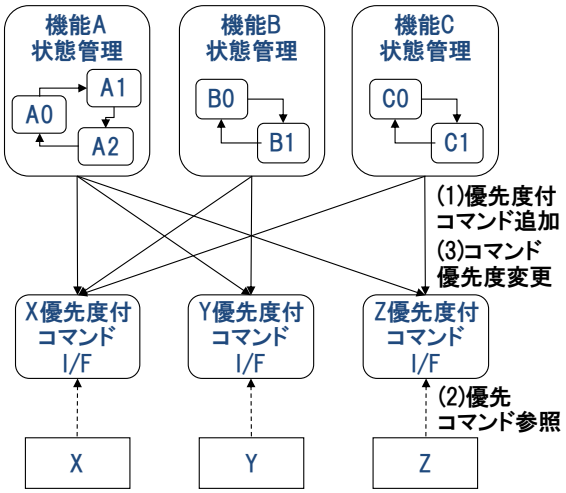

図 7　優先度付コマンドインタフェースに基づくソフトウェア構造の例

Figure 7　An example of software structure based on prioritized command interface

5.2 複雑度

5.1 節の構造におけるソフトウェア複雑度を検討する．3.2 節で述べた状態管理モジュールの複雑度 M_f は変わらない．新たに導入する優先度付コマンドインタフェースモジュールの複雑度 M_p は，優先度付待ち行列の複雑度を p とすると，

$$M_p = \Sigma_a p$$

となる．

ここで，p は定数となる為，M_p は，アクチュエータ数に比例する．一方，元のアクチュエータ優先処理モジュールの複雑度 M_a は，アクチュエータ数に加えて機能状態数が増えるほど増大する．したがって，機能状態数が多い場合には，アクチュエータ優先処理モジュールを優先度付コマンドインタフェースモジュールに置換することによって複雑度を低減することができる．

6. 空調制御ソフトウェアへの適用試行

5 章で述べたソフトウェア構造化手法の評価のため，空調制御ソフトウェアを対象として適用試行した．適用前の構造は，3.2 節で述べたアクチュエータ優先処理に基づくソフトウェア構造である（図 8）．アクチュエータのうち，

Vol.2019-ARC-235 No.11
Vol.2019-SLDM-187 No.11
Vol.2019-EMB-50 No.11
2019/3/17

ファンとルーバについてアクチュエータ優先処理モジュールから優先度付コマンドインタフェースモジュールへの置換を行った（図 9）．ファン，ルーバとも適用前後で動作に変更がないことを確認した．表 5 に評価結果を示す．本手法の適用により複雑度が低減できることを確認した．

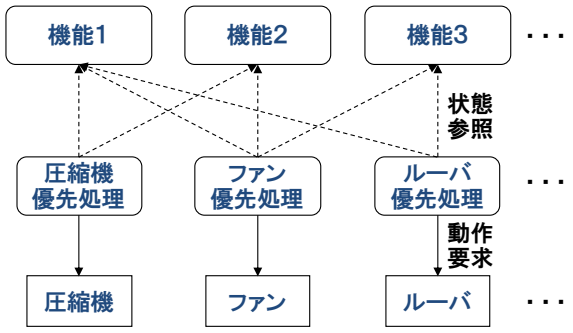

図 8　空調制御ソフトウェア構造

Figure 8　Air-conditioning control software structure

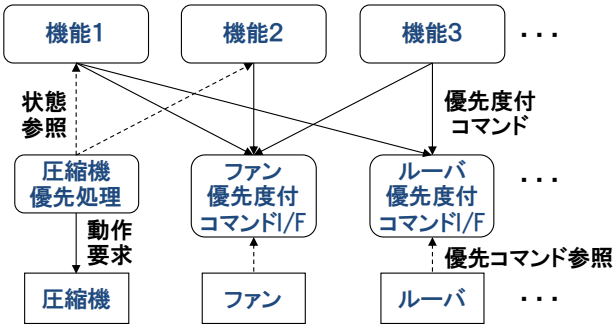

図 9　空調制御ソフトウェア構造の修正

Figure 9　Modification of air-conditioning control software structure

表 5　評価結果

Table 5　Result of evaluation

適用アクチュエータ	機能数	機能状態数	循環的複雑度			
			最大		平均	
			適用前	適用後	適用前	適用後
ファン	14	42	18	11	8.4	4.1
ルーバ	8	15	14	10	4.4	2.3

7. おわりに

本稿では，組込み制御システムにおける機能競合を分析し，その特徴を明らかにした．機能競合解決のための一般的なソフトウェア構造では，ロジックのハードコーディングにより機能数と機能状態数が増えるほど，複雑化する問題がある．そこで，ロジックの影響を低減する為，汎用的な OS 機構やアルゴリズム・データ構造を機能競合解決に利用する検討を行った．OS による優先度方式タスクスケジューリングや共有資源排他制御等の機構では対応が困難である．そこで，優先度付待ち行列の機能を応用し，各機能モジュールからアクチュエータへ優先度を付したコマンドインタフェースを設け，競合解決を行うソフトウェア構造化手法を提案した．空調制御ソフトウェアを対象として試行適用した結果, 循環的複雑度を約 3～5 割程度削減できた．本手法により，組込み制御ソフトウェアの複雑度を低減し，信頼性と生産性を向上することができる．

今後の課題としては，さらなる開発効率化を目指して機能優先度等の仕様決定作業の容易化が挙げられる．

参考文献

[1] 情報処理推進機構．“2017 年度組込みソフトウェアに関する動向調査”，2018
[2] 情報処理推進機構．“ソフトウェア開発データが語るメッセージ 2017”，2018
[3] Glenford J. Myers. Composite/Structured Design. Litton Educational Publishing, 1978
[4] 中本幸一，高田広章，田丸喜一郎．組込みシステム技術の現状と動向．情報処理学会論文誌. 1997, vol. 38, no. 10, p. 871-878.
[5] 花井志生．モダンＣ言語プログラミング．アスキー･メディアワークス, 2013, 103p.
[6] Thomas J. McCabe. A Complexity Measure. IEEE Transactions. 1976, vol. SE-2, no. 4, p. 308-320
[7] “リアルタイムＯＳのしくみ｜トロンフォーラム”. https://www.tron.org/ja/onwebseminar/chap3/, (参照 2019-02-04).
[8] 社団法人日本システムハウス協会エンベデッド技術者育成委員会．組込みソフト技術．電波新聞社, 2005, 153p.
[9] Gerth Stølting Brodal. Space-Efficient Data Structures, Streams, and Algorithms: A Survey on Priority Queues. Springer, 2013, p.150-163
[10] “priority_queue – cpprefjp Ｃ＋＋日本語リファレンス” https://cpprefjp.github.io/reference/queue/priority_queue.html, (参照 2019-02-04).
[11] “PriorityQueue (Java Platform SE 8)” https://docs.oracle.com/javase/jp/8/docs/api/java/util/PriorityQueue.html, (参照 2019-02-04).