Vol.2017-ARC-225 No.32
Vol.2017-SLDM-179 No.32
Vol.2017-EMB-44 No.32
2017/3/10

テンプレート記述を導入したアスペクト指向プログラミングによるリアルタイムＯＳファミリーの実現

原田 祐輔[1,a)]　石川 大貴[1,†1]　兪 明連[1,b)]　横山 孝典[1,c)]

概要：組込みシステムは様々な用途に使用されるため，必要となるリアルタイム OS(RTOS) の機能はアプリケーションによって異なることが多い．そこで，リソース制約のある組込みシステムでは，アプリケーションに応じて必要最小限の機能を備える RTOS を提供することが望まれる．我々はこれまでにアスペクト指向プログラミング (AOP) を用いて RTOS をカスタマイズする手法を提案し，分散処理機能の追加やスケジューリングアルゴリズムの置き換えを行い，RTOS のファミリー化を行ってきた．しかし既存の C 言語ベースのアスペクト指向言語を用いており，アスペクト間に重複が見られるなど改良の余地が見られた．そこで本論文では，重複の少ないモジュール化に優れた記述を可能とするテンプレートアスペクトを提案する．そして実際に RTOS ファミリーに適用し，重複記述を削減できることを示す．また AOP によるオーバヘッドが実用上問題ない程度に抑えられることも示す．

キーワード：組込みシステム，リアルタイム OS，アスペクト指向プログラミング，分散処理，並列処理，マルチコアプロセッサ，スケジューリング

Yusuke Harada[1,a)]　Hiroki Ishikawa[1,†1]　Myongryun Yoo[1,b)]　Takanori Yokoyama[1,c)]

1. はじめに

組込みシステムは様々な用途に使用されるため，必要となるリアルタイム OS(RTOS) の機能もアプリケーションによって異なることが多い．しかし，リソース消費量の制約が大きい組込みシステムにおいて，それら全ての機能を備えた RTOS を搭載する事は困難である．そこでアプリケーションの要求に応じて必要最小限の機能を備えた RTOS を提供することが望まれる．それには，必要な機能のみを備えた RTOS を選択できる RTOS ファミリーが有効である．また，それにともない，RTOS ファミリー化を効率的に実現する手法が求められている．

横断的関心事を分離してモジュール化できるアスペクト指向プログラミング (AOP)[1] を用いて，RTOS をカスタマイズする研究がなされている．Afonso らは，RTOS の同期 (排他制御) やロギングにアスペクトを適用している [2]．Park らは，プログラミング言語非依存な AOP 環境 AOX を開発し，カスタマイズ可能な RTOS への適用を提案している [3]．Beuche らは，AOP を用いてアーキテクチャ非依存な RTOS を実現する手法を提案している [4]．また Lohmann らは，RTOS ファミリーの開発には最初からアスペクトを意識した設計が必要として Aspect-Aware Design を提唱し，AUTOSAR OS[5] の機能の取捨選択が可能な RTOS ファミリーを開発している [6]．

我々は，OSEK OS[7] を対象に，分散処理機能を追加するアスペクト [8]，固定優先度スケジューリングを EDF(Earliest Deadline First)[9] スケジューリングまたは RMCL(Rate Monotonic Critical Laxity)[10] スケジューリングに置き換えるアスペクト [11], [12] を定義しカスタマイズを行った．しかし，実装に用いた C 言語ベースのアスペクト指向言語 ACC (AspeCt-oriented C)[13], [14] の制約から，横断的関心事のモジュール化は必ずしも十分でなく，アスペクト間に重複した記述があり改良の余地が見られた．

[1] 東京都市大学
Tokyo City University, Tokyo 158–8557, Japan
[†1] 現在，株式会社日立システムズ
Presently with Hitachi Systems, Ltd.
[a)] g1581519@tcu.ac.jp
[b)] myoo@tcu.ac.jp
[c)] tyoko@tcu.ac.jp

Vol.2017-ARC-225 No.32
Vol.2017-SLDM-179 No.32
Vol.2017-EMB-44 No.32
2017/3/10

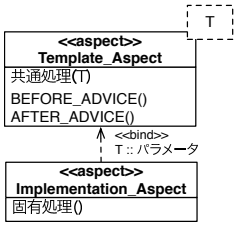

図 1 テンプレートアスペクトと実装アスペクト

そこで本論文では，よりモジュール化に優れたテンプレートアスペクトの記述法を提案する．そしてこれまで開発した上記アスペクト及びマルチコア並列処理機能を追加するアスペクトにテンプレートアスペクトを適用し，重複記述を削減できることを示す．

論文の構成は以下のとおりである．第 2 節ではテンプレートアスペクトの記述法について説明する．第 3 節では，テンプレートアスペクト記述による並列・分散 RTOS の実現及びスケジューラのカスタマイズについて述べる．第 4 節では記述削減の効果と AOP によるオーバヘッドを評価する．最後に 5 節で本研究の結論を述べる．

2. アスペクト記述法の拡張

2.1 アスペクト指向言語 ACC によるアスペクト記述

C 言語ベースの ACC は，ジョインポイントモデルに基づくアスペクト指向言語で，扱えるジョインポイントは関数の呼び出し (call) と実行 (execution)，変数への値の書き込み (set) と値の読み出し (get) である．アスペクトはポイントカット (pointcut) とアドバイス (advice) からなる．ポイントカットはジョインポイントの集合を指定するもので，アドバイスはポイントカットに合致するジョインポイントで実行する処理を記述したものである．ACC は before，after，around の 3 つのアドバイスをサポートしており，対象ジョインポイントの前後あるいはその代わりに，アドバイスコードを実行することができる．

しかし ACC にはオブジェクト指向言語ベースの AspectJ[15] におけるアスペクトの継承や抽象アスペクト等の機能はなく，扱えるジョインポイントも限られているため，重複して記述せざるを得ない場合がある．

2.2 テンプレートアスペクトの導入

本論文では，重複記述の削減に有効なテンプレートアスペクトを提案する．ただし，新たにアスペクト指向言語処理系を開発することは負担が大きいので，マクロによるテンプレートアスペクト記述を採用する．これにより，ACC の処理系をそのまま利用できる．

本手法では，複数アスペクトの共通部分をテンプレートアスペクトに記述し，それをベースに各アスペクト固有の部分を実装アスペクトに記述する．テンプレートアスペクトと実装アスペクトの関係を表した UML のクラス図を

```
#define \
  テンプレートアスペクト(パラメータ, …) \
アドバイス() : ポイントカット()        \
{                                      \
    BEFORE_ADVICE()                    \
    共通処理                           \
      ・・・                           \
    AFTER_ADVICE()                     \
}
#define BEFORE_ADVICE()
#define AFTER_ADVICE()
```

図 2 テンプレートアスペクト

```
#include"テンプレートアスペクト.h"
#define BEFORE_ADVICE() _前処理_
#define AFTER_ADVICE()  _後処理_
#define 固有処理()
テンプレートアスペクト(パラメータ, …)
```

図 3 実装アスペクト

図 1 に示す．テンプレートアスペクト Template_Aspect にパラメータの T を与えることでアスペクトを実装する．bind の下の T::部にパラメータを記述する．

図 2 にテンプレートアスペクトの記述形式を示す．テンプレートアスペクトはヘッダファイルとして定義する．マクロ関数のテンプレートアスペクト () に共通処理を記述する．引数に文字列を与え，トークン演算子##を用いて，アスペクトコード中の文字列を引数の文字列に置き換える．メインの処理の前後に処理を追記するための空のマクロ関数 BEFORE_ADVICE() と AFTER_ADVICE() も定義している．**図 3** に実装アスペクトの記述形式を示す．テンプレートアスペクトのヘッダファイルをインクルードし，テンプレートアスペクトの引数でパラメータの値を定義する．メインの処理の前後に処理を追加する場合は，マクロ関数 BEFORE_ADVICE() または，AFTER_ADVICE() に記述する．

ACC はトランスレータとして実装されており，はじめにベースとするプログラムの C ソースファイルとカスタマイズのためのアスペクトファイルおよび C ソースファイル内のマクロ記述を C コンパイラのプリプロセッサを用いて展開する．この時にテンプレートアスペクト () がマクロ展開され，ACC のアスペクトが生成される．マクロ展開されたファイルを ACC トランスレータに入力し，織り込みを行った後，C のソースファイルを出力する．出力されたファイルを C コンパイラによりコンパイルすることでオブジェクトコードを生成する．

3. RTOS ファミリーの実現

3.1 RTOS ファミリーの機能

本研究では，自動車制御分野の標準である OSEK OS 仕様 [7] に基づくオープンソースの RTOS である TOPPERS/ATK1[16] を対象として機能を拡張する．

RTOS ファミリーが提供する機能はフィーチャモデル [17] で表現することができ，その一部を**図 4** に示す．図の破線内が OSEK OS 仕様内のフィーチャである．

スケジューリングアルゴリズムのカスタマイズでは，オルタナティブフィーチャで表現されている固定優先度スケジューリング，EDF スケジューリング，RMCL スケジューリングのいずれかを選択可能で，それに応じてリソースア

Vol.2017-ARC-225 No.32
Vol.2017-SLDM-179 No.32
Vol.2017-EMB-44 No.32
2017/3/10

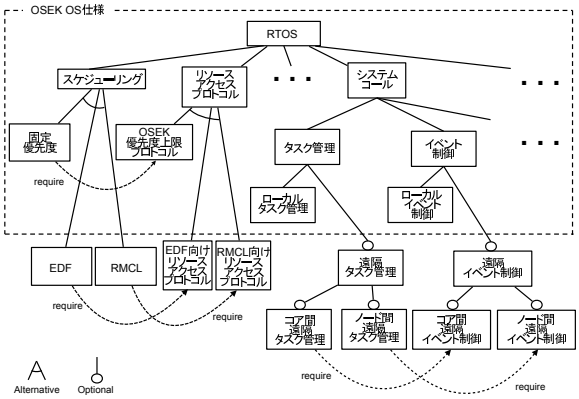

図 4　RTOS ファミリーのフィーチャモデル

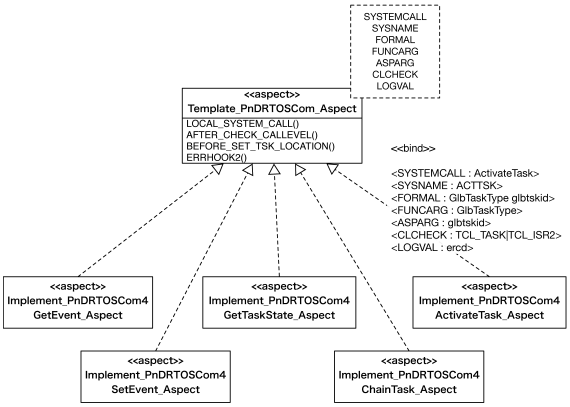

図 5　共通部テンプレートアスペクトと各実装アスペクト

クセスプロトコルの OSEK 優先度上限プロトコル，EDF 向けリソースアクセスプロトコル，RMCL 向けリソースアクセスプロトコルを選択する．

並列分散 RTOS 実現のためのカスタマイズでは，タスク管理のオプションフィーチャのコア間遠隔タスク管理，ノード間遠隔タスク管理を選択可能とする．また，イベント制御のオプションフィーチャのコア間遠隔イベント制御，ノード間遠隔イベント制御を選択可能とする．

3.2　並列・分散 RTOS 実現のためのアスペクト

OSEK OS が提供するシステムコールのうち，タスク管理の ActivateTask()，ChainTask()，GetTaskState() と，イベント制御の GetEvent()，SetEvent() をカスタマイズして，並列・分散 RTOS を実現する． **図 5** にシステムコールの共通部テンプレートアスペクトと，各実装アスペクトの関係を示す．共通部テンプレートアスペクトには，対象システムコールすべてに共通する，対象タスクの位置判定とローカルタスク向けのシステムコール処理を記述する．そして，これをベースにそれぞれのシステムコールで必要な処理を実装アスペクトに記述する．具体的には並列処理向けアスペクトとしてメモリ経由メッセージ送信を，分散処理向けアスペクトとしてネットワーク経由メッセージ送信を行う処理を記述する．

```
#define Template_PnDRTOSCom_Aspect(SYSTEMCALL, SYSNAME,                          \
                                   FORMAL, FUNCARG, ASPARG, CLCHECK, LOGVAL)  \
StatusType around(FORMAL):execution(StatusType SYSTEMCALL(FUNCARG))&&args(ASPARG)  \
{                                                                               \
  BEFORE_ADVICE()                                                               \
  ercd = E_OK;                                                                  \
  CHECK_CALLEVEL(CLCHECK);                                                      \
  AFTER_CHECK_CALLEVEL()                                                        \
  lock_cpu();                                                                   \
  if( NODE_ID(glbtskid) == MY_NODE_ID ) {                                       \
    LOCAL_SYSTEM_CALL()                                                         \
  } else {                                                                      \
    glbtskid_store = glbtskid;                                                  \
    BEFORE_SET_TSK_LOCATION()                                                   \
    tsk_location = check_target_location(glbtskid_store);                       \
  }                                                                             \
  AFTER_ADVICE()                                                                \
exit:                                                                           \
  unlock_cpu();                                                                 \
  LOG_##SYSNAME##_LEAVE(LOGVAL);                                                \
  return(ercd);                                                                 \
error_exit:                                                                     \
  lock_cpu();                                                                   \
d_error_exit:                                                                   \
  _errorhook_par1.tskid = TSKID(glbtskid);                                      \
  ERRHOOK2();                                                                   \
  call_errorhook(ercd, OSServiceId_ ## SYSTEMCALL );                            \
  unlock_cpu();                                                                 \
  goto exit;                                                                    \
}
#define BEFORE_ADVICE()
#define AFTER_ADVICE()
#define LOCAL_SYSTEM_CALL()
#define BEFORE_SET_TSK_LOCATION()
#define ERRHOOK2()
```

図 6　共通部テンプレートアスペクト記述

```
#include "Template_PnDRTOSCom_Aspect.h"
#include "_activateTask.h"
Template_PnDRTOSCom_Aspect(
/* システムコール名 */
      ActivateTask,
/* システムコール名略称 */
      ACTTSK,
/* アスペクト引数 */
      GlbTaskType glbtskid,
/* args引数 */
      GlbTaskType,
/* call引数 */
      gtbtskid,
/* コールレベルチェック */
      TCL_TASK|TCL_ISR2,
/* ログマクロ引数 */
      ercd
)
```

図 7　実装アスペクト記述

```
#define LOCAL_SYSTEM_CALL()                                                 \
if (tcb_tstat[TSKID(glbtskid)] == TS_DORMANT) {                             \
      if ((make_active(TSKID(glbtskid))) && (callevel == TCL_TASK)) {       \
            dispatch();                                                     \
      }                                                                     \
}                                                                           \
else if (tcb_actcnt[TSKID(glbtskid)] < tinib_maxact[TSKID(glbtskid)]) {     \
      tcb_actcnt[TSKID(glbtskid)] += 1;                                     \
} else {                                                                    \
      ercd = E_OS_LIMIT;                                                    \
      goto d_error_exit;                                                    \
}
```

図 8　ActivateTask 向けローカルシステムコール処理

図 6 に共通部のテンプレートアスペクト記述を示す．テンプレートアスペクトを表すマクロ関数 Template_PnDRTOSCom_Aspect に与えるパラメータは，システムコール名の SYSTEMCALL，ログマクロ名の一部に用いるシステムコール略称の SYSNAME，around アドバイスの引数に与える型と変数名の FORMAL，execution ポイントカット内でシステムコールに与える引数の FUNCARG，args ポイントカットに与える引数の ASPARGS，コールレベルチェックの条件の CLCHECK，ログマクロに与える引数の LOGVAL である．アスペクトのアドバイス部には，システムコール共通の処理を記述する．マ

Vol.2017-ARC-225 No.32
Vol.2017-SLDM-179 No.32
Vol.2017-EMB-44 No.32
2017/3/10

表 1 スケジューリングアルゴリズムとリソースアクセスプロトコルのためのアスペクト

アスペクト		EDF	RMCL
スケジューリング	レディキュー操作	x	x
	実行タスク選択	x	x
	プリエンプション	x	x
	スケジューラ呼出	x	x
	残り実行時間管理	-	x
	デッドライン更新	x	
	タスク初期化処理	x	x
リソース管理機能	リソース管理	x	x
	上限優先度管理	x	-
	共有リソース探索	-	x
	獲得リソース管理	x	
	リソース初期化処理	x	x

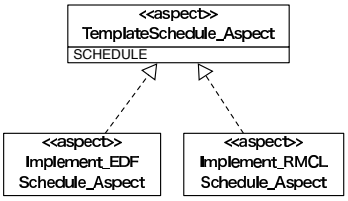

図 9 スケジューラテンプレートアスペクトと各実装アスペクト

クロ関数 LOCAL_SYSTEM_CALL() には自タスク向けのシステムコール処理，AFTER_CHECK_CALLEVEL() にはコールレベル以外の状態チェックを行うための処理，BEFORE_SET_TSK_LOCATION() にはタスク ID を以外の値の格納処理，ERRHOOK2() には追加のエラーフック処理を記述する．

図 7 に ActivateTask 向けの実装アスペクト記述を示す．Template_PnDRTOSCom_Aspect に，システムコール名 (SYSNAME) ActivateTask，システムコール略称 (SYSNAME)ACTTSK，around アドバイスの引数に与える型と変数名 (FORMAL) GlbTaskType glbtskid，システムコールの引数の型 (FUNCARG)GlbTaskType，args ポイントカットの引数 (ASPARG) glbtskid，コールレベルチェック (CKCHECK) TCL_TASK | TCL_ISR2，ログマクロ引数 (LOGVAL) ercd を引数として与える．マクロ関数 LOCAL_SYSTEM_CALL() には自タスク向けのタスク起動処理を記述するが，記述量が多いため見やすさを考慮し**図 8** に示すヘッダファイル_activateTask.h に記述する．Template_PnDRTOSCom_Aspect をマクロ展開することで，アスペクトコードを生成する．

3.3 スケジューラカスタマイズのためのアスペクト

OSEK OS の仕様で定められている固定優先度スケジューリングアルゴリズムを，EDF スケジューリングと RMCL スケジューリングにカスタマイズするためのアスペクトの一覧を**表 1** に示す．EDF スケジューリング向けに 10 個の，RMCL スケジューリング向けに 11 個のアスペクト (うち 2 個のアスペクトは共通) を定義する．

EDF と RMCL のアスペクトには，スケジューラ呼び出しシステムコール Schedule()，リソース管理システムコール GetResource() と ReleaseResource() に重複記述があるが，共通化できるのは主に引数のエラーチェックとエラー処理である．**図 9** にスケジューラのテンプレートアスペクトと各実装アスペクトの関係を示す．引数チェックやエラー処理をテンプレートアスペクトで共通化し，各スケジューリングの処理を実装アスペクトに記述する．

図 10 に EDF と RMCL の共通処理を記述するテンプレートアスペクトを示す．テンプレートアスペクトを表すマクロ関数 Template_Schedule_Aspect に与える引数はない．マクロ関数 SCHEDULE() に各スケジューラ向けの処理を記述する． **図 11** に EDF 向けスケジューラの実装アスペクト記述を示す．EDF スケジューラの処理を記述するマクロ関数 SCHEDULE() は記述量が多いため，見やすさを考慮し，**図 12** に示すヘッダファイル_edf_schedule.h に記述する．マクロ関数 Template_Schedule_Aspect を展開することでアスペクトコードを生成する．

```
#define Template_Schedule_Aspect()                                          \
StatusType around() : execution(StatusType Schedule(void))                  \
{                                                                           \
  BEFORE_ADVICE()                                                           \
  StatusType  ercd = E_OK;                                                  \
  LOG_SCHED_ENTER();                                                        \
  CHECK_CALLEVEL(TCL_TASK);                                                 \
  CHECK_RESOURCE(tcb_lastres[runtsk] == RESID_NULL);                        \
  lock_cpu();                                                               \
  SCHEDULE()                                                                \
  AFTER_ADVICE()                                                            \
exit:                                                                       \
  unlock_cpu();                                                             \
  LOG_SCHED_LEAVE(ercd);                                                    \
  return(ercd);                                                             \
error_exit:                                                                 \
  lock_cpu();                                                               \
  call_errorhook(ercd, OSServiceId_Schedule);                               \
  goto exit;                                                                \
}
#define BEFORE_ADVICE()
#define AFTER_ADVICE()
#define SCHEDULE()
```

図 10 スケジューラテンプレートアスペクト記述

```
#include "Template_Schedule_Aspect.h"
#include "_edf_schedule.h"
Template_Schedule_Aspect()
```

図 11 スケジューラ実装アスペクト記述

```
#define SCHEDULE()                                                          \
  TickType curval;                                                          \
  TickType dl;                                                              \
  if (nexttsk != TSKID_NULL ){                                              \
    curval = cntcb_curval[tskcnt];                                          \
    dl = tcb_deadline[nexttsk];                                             \
    if((curval < dl && (curval < tcb_deadline[schedtsk] &&                  \
     tcb_deadline[schedtsk] < dl )) ||                                      \
     (curval > dl && (curval < tcb_deadline[schedtsk] ||                    \
     tcb_deadline[schedtsk] < dl ))){                                       \
      preempt_edf();                                                        \
      dispatch();                                                           \
    }                                                                       \
  }
```

図 12 EDF 向けスケジューラ処理

表 2 アスペクト記述量

OS	テンプレートアスペクト記述	従来アスペクト記述 (重複記述量)
並列・分散 RTOS	272	257 (160)
スケジューラカスタマイズ	212	219 (100)

4. 評価

4.1 アスペクト記述量の評価

ソースコードの保守性を評価するために，テンプレート

Vol.2017-ARC-225 No.32
Vol.2017-SLDM-179 No.32
Vol.2017-EMB-44 No.32
2017/3/10

表 3 並列・分散 RTOS の実行時間評価

System Call	Optimizing Option	Execution Time[μsec]					
		Local		Inter-Node		Inter-Core	
		AOP	Rewrite	AOP	Rewrite	AOP	Rewrite
Activate Task()	none	1.83	1.89	6.28	5.89	5.58	5.26
	size	1.47	1.47	4.97	4.79	3.93	3.76
	speed	1.47	1.39	4.86	4.64	4.00	3.19
	size&speed	1.47	1.47	5.43	4.63	4.29	3.55
Chain Task()	none	2.28	2.92	6.06	6.06	6.12	6.12
	size	2.01	2.16	6.39	6.41	6.45	6.45
	speed	1.86	2.01	6.51	6.48	6.51	6.51
	size&speed	2.01	2.16	6.36	6.45	6.45	6.45
GetTask State()	none	0.91	0.93	6.24	5.86	5.62	5.23
	size	0.79	0.78	5.08	4.77	4.00	3.71
	speed	0.70	0.69	4.67	4.38	3.46	3.14
	size&speed	0.76	0.78	5.44	4.57	4.24	3.51
Set Event()	none	1.10	1.16	6.35	5.89	5.56	5.38
	size	0.78	0.83	5.14	4.91	4.00	3.81
	speed	0.70	0.83	4.67	4.59	3.43	3.28
	size&speed	0.78	0.77	5.31	4.80	4.11	3.65
Get Event()	none	0.86	1.01	6.31	5.88	5.62	5.33
	size	0.68	0.79	5.01	4.77	4.01	3.79
	speed	0.84	0.85	4.66	4.54	3.81	3.29
	size&speed	0.84	0.80	5.04	4.76	3.78	3.64

アスペクトを用いたアスペクト記述法と従来のアスペクト記述法について，記述量 (コードの行数) を比較する．

アスペクト記述量を**表 2** に示す．従来アスペクト記述のカッコ内は複数アスペクトで全く同じもしくは一部が重複している記述の行数である．両記述法を比較すると行数は同程度であるが，並列・分散 RTOS では 160 行，スケジューラでは 100 行の重複記述をテンプレートアスペクトを用いることで削減できた．

4.2 AOP によるオーバヘッドの評価

4.2.1 実験環境

並列・分散 RTOS の評価には，並列・分散 RTOS の開発 [18] に用いてきた，2 つの SH2A-FPU コアを持つデュアルコアプロセッサ SH7205 を搭載した評価ボード M3-HS50 を用いる．クロック周波数は 200MHz である．ネットワークには CAN を用いる．C コンパイラには C/C++ compiler package for SuperH RISC engine family 9.04.03 を用いる．最適化オプションによる影響を考慮し，最適化なし，サイズ優先の size，スピード優先の speed，両者優先の size&speed の場合について評価する．

スケジューラカスタマイズの評価には，これまでスケジューラのカスタマイズ [11], [12] で用いてきたマイクロコントローラ H8S/2638F を搭載した評価ボードを用いる．クロック周波数は 20MHz である．C コンパイラには GCC 3.4.6 を用いる．最適化オプションによる影響を考慮し，TOPPERS/ATK1 の makefile で指定している最適化オプション O2 に加え，最適化を行わない場合 (O0)，O1，O3 の場合について評価する．

4.2.2 CPU 実行時間

カスタマイズした機能に関わるシステムコールについて CPU 実行時間を測定し，ソースコードを直接書換えて実装した場合と比較することで AOP によるオーバヘッドを評価する．**表 3** に並列・分散 RTOS における CPU 実行時間 (発行から終了までの時間を 100 回測定した平均値) を示す．Local は対象タスクが同一ノード同一 CPU の場合，Inter-Node は他ノード上の CPU の場合，Inter-Core は同一ノード他 CPU の場合である．Inter-Node は CAN の通信時間を除いた値である．AOP は AOP でカスタマイズした場合，Rewrite はソースコードを直接修正した場合で両者の実行時間の差が AOP によるオーバヘッドを表す．最適化を行わない時に最もオーバヘッドが大きいのは他 Inter-Node の SetEvent() で，8%程度 (0.46μsec) の増大である．最適化を行う時に最もオーバヘッドが大きいのは Inter-Node の GetTaskState() で，19%程度 (0.87μsec) の増大に抑えることができる．

表 4 スケジューラカスタマイズの実行時間評価

System Call	Optimizing Option	Execution Time[μsec]				
		EDF		RMCL		Fixed
		AOP	Rewrite	AOP	Rewrite	
ActivateTask()	O0	140.7	117.8	263.6	220.8	75.4
	O1	92.6	81.3	172.9	139.1	51.0
	O2	84.0	67.5	163.4	137.4	53.9
	O3	77.6	62.7	154.1	108.3	44.3
ChainTask()	O0	161.4	129.2	361.3	296.7	93.1
	O1	110.0	88.9	237.6	185.5	64.4
	O2	99.6	85.1	218.5	177.7	62.8
	O3	93.5	68.2	217.2	151.4	53.9
TerminateTask()	O0	190.7	166.1	418.7	361.1	118.6
	O1	124.0	104.0	265.2	218.7	66.4
	O2	117.7	98.5	246.3	209.2	62.2
	O3	116.7	87.9	251.5	207.8	57.6
Schedule()	O0	106.5	90.6	201.5	170.9	27.3
	O1	72.2	57.5	133.3	109.1	13.0
	O2	68.3	55.5	125.2	104.0	15.3
	O3	68.3	49.2	124.6	97.9	12.4
GetResource()	O0	54.6	43.3	58.0	46.0	37.3
	O1	32.5	22.5	35.4	25.8	20.2
	O2	32.5	22.0	36.6	26.1	20.7
	O3	32.2	22.3	36.6	27.8	21.7
ReleaseResource()	O0	67.1	56.2	224.6	188.4	39.3
	O1	41.6	34.3	151.0	120.2	21.9
	O2	41.6	31.2	143.0	116.9	21.2
	O3	42.0	31.2	145.1	111.3	23.0
WaitEvent()	O0	57.0	49.1	175.6	152.6	63.8
	O1	42.5	35.8	115.2	96.7	45.5
	O2	41.5	35.0	107.5	95.6	42.8
	O3	40.6	30.8	119.7	92.2	38.6

表 4 にスケジューラカスタマイズにおける CPU 実行時間を示す．参考のため，オリジナルの TOPPERS/ATK1 の固定優先度スケジューリングの場合の実行時間を Fixed に示す．最適化を行わない時に最もオーバヘッドの大きいのは RMCL スケジューリングの ChainTask()，で 22%程度 (64μsec) 程度増大する．O2 最適化を行うと，23%程度 (40μsec) 程度の増加に抑えることができる．

以上の評価から，最適化を行える状況では，AOP のオーバヘッドは実用上問題ない範囲と考える．

4.2.3 メモリ消費量

AOP によるメモリ消費量のオーバヘッドを評価する．**表 5** に，並列・分散 RTOS でのコード領域，ROM 上のデータ領域，RAM 上のデータ領域のメモリ消費量を示す．最適化を行わない場合は，コード領域で 20%程度 (6.8kB 程度)，最適化を行うと 7%〜13%程度 (1.6kB〜2.9kB 程度)

表 5　並列・分散 RTOS のメモリ消費量評価

Optimizing Option	Memory Consumption [Byte]					
	Code		RAM		ROM	
	AOP	Rewrite	AOP	Rewrite	AOP	Rewrite
none	40432	33572	10988	10964	3839	3823
size	21060	18680	10988	10964	3819	3803
speed	25932	24244	10988	10964	3819	3803
size&speed	21570	19174	10988	10964	3819	3803

表 6　スケジューラカスタマイズのメモリ消費量評価

Scheduling	Optimizing Option	Memory Consumption [Byte]					
		Code		RAM		ROM	
		AOP	Rewrite	AOP	Rewrite	AOP	Rewrite
EDF	O0	22180	18424	868	868	1799	1797
	O1	18208	14560	856	856	1799	1797
	O2	17056	13760	856	856	1799	1797
	O3	18692	15188	856	856	1799	1797
RMCL	O0	24040	19816	891	891	1799	1796
	O1	18832	15448	879	879	1795	1796
	O2	17352	14240	879	879	1795	1796
	O3	19568	17120	879	879	1795	1796
FIXED	O0	17208		859		1793	
	O1	13804		847		1793	
	O2	12684		847		1793	
	O3	14456		847		1793	

の増加となる．データ領域については同程度で増大はない．

表 6 に，スケジューラカスタマイズでのコード領域，ROM 上のデータ領域，RAM 上のデータ領域のメモリ消費量を示す．最適化を行わない時に最もオーバヘッドの大きいのは RMCL のコード領域で 21%程度 (4kB 程度) 増加する．最適化を行うと，コード量は 22%程度 (3kB 程度) の増加となる．データ領域については同程度である．

アプリケーションを含めたシステム全体のメモリ消費量に対する増加量は 1%以下である．コード領域のメモリ消費量の増大は無視できる量ではないが，ACC 処理系によるもので改善の余地があり，今後の課題である．

5. おわりに

RTOS ファミリーの開発には，既存の RTOS のソースコードを直接修正することなくカスタマイズ可能な AOP が有効である．本論文では C 言語ベースのアスペクト指向言語における重複記述の削減を目的に，テンプレートアスペクトの記述法を提案した．そして，マルチコア並列処理機能および分散処理機能を追加するためのアスペクト，スケジューリングアルゴリズムを EDF や RMCL にカスタマイズするアスペクトに適用し，重複した記述を大きく削減できることを示した．

今後の課題として，メモリ消費量を削減する手法の検討が挙げられる．

謝辞　本研究で使用した TOPPERS/ATK1 の開発者と ACC の開発者に感謝する．本研究は JSPS 科研費 JP15K00084 の助成をうけたものである．

参考文献

[1] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. and Irwin, J.: Aspect-oriented programming, *ECOOP'97 — Object-Oriented Programming*, Vol. 1241, pp. 220–242 (1997).

[2] Afonso, F., Silva, C., Montenegro, S. and Tavares, A.: Applying Aspects to a Real-time Embedded Operating System, *6th Workshop on Aspects, Components, and Patterns for Infrastructure Software*, ACP4IS '07 (2007).

[3] Park, J. and Hong, S.: Building a Customizable Embedded Operating System with Fine-grained Joinpoints Using the AOX Programming Environment, *2009 ACM Symposium on Applied Computing*, SAC '09, pp. 1952–1956 (2009).

[4] Beuche, D., Fröhlich, A. A., Meyer, R., Papajewski, H., Schön, F., Schröder-Preikschat, W., Spinczyk, O. and Spinczyk, U.: On Architecture Transparency in Operating Systems, *9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System*, EW 9, pp. 147–152 (2000).

[5] AUTOSAR: *Specification of Multi-Core OS Architecture V1.1.0 R4.0 Rev 2* (2010).

[6] Lohmann, D., Spinczyk, O., Hofer, W. and Schröder-Preikschat, W.: Transactions on Aspect-Oriented Software Development IX, Berlin, Heidelberg, chapter The Aspect-aware Design and Implementation of the CiAO Operating-system Family, pp. 168–215 (2012).

[7] OSEK/VDX: *Operating System, Version 2.2.3* (2005).

[8] Saito, N., Yoo, M. and Yokoyama, T.: A distributed real-time operating system built with aspect-oriented programming for distributed embedded control systems, *2014 20th IEEE International Conference on Parallel and Distributed Systems*, pp. 436–443 (2014).

[9] Liu, C. L. and Layland, J. W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *J. ACM*, Vol. 20, No. 1, pp. 46–61 (1973).

[10] 加藤真平，山崎信行：Linux カーネル用リアルタイムスケジューリングモジュール，情報処理学会論文誌コンピューティングシステム (ACS)，Vol. 2, No. 1, pp. 75–86 (2009).

[11] Harada, Y., Abe, K., Yoo, M. and Yokoyama, T.: Aspect-Oriented Customization of the Scheduling Algorithms and the Resource Access Protocols of a Real-Time Operating System Family, *2015 IEEE International Conference on Smart City*, pp. 87–94 (2015).

[12] 原田祐輔，阿部一樹，兪明連，横山孝典：アスペクト指向プログラミングによるリアルタイム OS スケジューラのカスタマイズ，情報処理学会論文誌，Vol. 57, No. 8, pp. 1752–1764 (2016).

[13] Gong, M. Zhang, C. and Jacobsen, H. A.: Systems Development with AspeCt-oriented C (ACC), *Connections 2007 (ECE Graduate Symposium, University of Toronto) Talk 5.6* (2007).

[14] AsceCt-oriented C: *https://sites.google.com/a/gapp.msrg.utoronto.ca/aspectc/*.

[15] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W. G.: An Overview of AspectJ, *15th European Conference on Object-Oriented Programming*, ECOOP '01, pp. 327–353 (2001).

[16] TOPPERS Project: *http://www.toppers.jp/*.

[17] Kang, K., Cohen, S., Hess, J., Novak, W. and Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University (1990).

[18] Yokoyama, K., Saito, M., Yoo, M. and Yokoyama, T.: A real-time operating system with location-transparent inter-core and inter-node system calls for distributed embedded control systems, *TENCON 2015 - 2015 IEEE Region 10 Conference*, pp. 1–4 (2015).