Vol.2015-SLDM-170 No.27
Vol.2015-EMB-36 No.27
2015/3/7

# 組込みコンポーネントシステムの呼び出しフロー解析ツールの開発

成瀬 有美[1,a)]　石川 拓也[1]　安積 卓也[2]　大山 博司[3]　高田 広章[1]

**概要**：組込みソフトウェア向けのコンポーネントベース開発を支援するコンポーネントシステムに TECS がある．TECS は, コンポーネント記述に独自のコンポーネント記述言語（TECS CDL）, 実装言語に C 言語を採用している．一般的にソフトウェアは, フロー解析を用いて検証を行うことで, 信頼性の向上が期待できる．TECS を利用して開発したシステム全体のフロー解析を行うためには, C 言語のフロー解析に加え, CDL の解析を行って C 言語の解析結果と統合を行う必要がある．そのため, TECS 向けのフロー解析を行う開発支援ツール TECS-flow を開発した．本ツールによって, TECS を使用して開発したシステム全体のフローを可視化し, コンポーネント間の関数呼び出し順序を知ることができる．加えて, 排他制御が必要な箇所の検出やデッドコード検出も可能である．なお, 本研究の一部は enPiT プロジェクトの一環として実施した．

Naruse Yumi[1,a)]　Ishikawa Takuya[1]　Azumi Takuya[2]　Oyama Hiroshi[3]　Takada Hiroaki[1]

## 1. はじめに

ソフトウェアを効率的に開発する手法の一つとして, ソフトウェアを機能の集合に分割した, コンポーネントと呼ばれるソフトウェア部品を組み合わせることで開発を行う, コンポーネントベース開発がある．この中に, メモリなどのリソース制約が厳しい組込みシステムに適したコンポーネントベース開発支援システムとして, TOPPERS プロジェクトで開発された TOPPERS Embedded Component System（TECS）[4] がある．

一般的にソフトウェアは, 静的解析を用いて検証を行うことで, 信頼性の向上が期待できる [2]．静的解析のひとつにフロー解析がある．テスト段階前にフロー解析を行うことで, 動作フロー依存エラーの検出や, コードが設計通りに正しく実装されていることの確認ができる．

TECS は C 言語で実装を行うため, 実装コードに対しては既存の C 言語解析ツールでフロー解析を行うことができる．しかし, TECS は実装コードをコンポーネント単位で分割しており, またコンポーネント間の関数呼び出しは TECS の提供するコンポーネントごとのポートを介して行われる．このため, 既存の C 言語解析ツールではコンポーネント内部のみのフローは解析できるが, コンポーネント間の呼び出し関係を含めたソフトウェア全体のフローは知ることができない．さらに, TECS コンポーネントの結合関係記述には独自の記述言語を採用しており, こちらは対応するフロー解析ツールが存在しない．そのため, TECS を用いて開発したソフトウェアのフロー解析を行うためには, TECS コンポーネント記述言語を解析してコンポーネント間の呼び出し関係を知り, コンポーネント内部呼び出し関係を表す C 言語の解析結果と統合する必要がある．これにより, TECS を利用して開発したソフトウェアが設計通りに正しく実装されているかどうかを検証することができる．

既存の C 言語解析ツールは, フリーソフトウェアから商用ソフトウェアまで多くのものが存在する．また, コンポーネントの生成・結合を動的に行う, 既存のコンポーネントソフトウェアは形式手法で解析・検証を行う [5]．これに対し, TECS のコンポーネント記述言語は独自のものであるため, 既存の解析ツールでは対応できない．しかし, TECS はコンポーネントの生成・結合を静的に行うため, TECS コンポーネント記述言語を解析し, その結果を C フロー解析結果と統合することで, 静的フロー解析が可能になると考えた．

[1] 名古屋大学大学院情報科学研究科
[2] 大阪大学大学院基礎工学研究科
[3] オークマ株式会社
[a)] yumi_n@ertl.jp

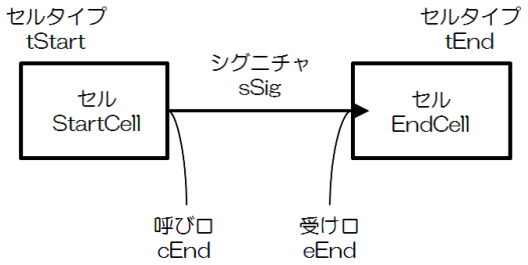


図 1　TECS のコンポーネントモデル

本研究の目的は, TECS を対象にしたフロー解析ツールを開発し, 組込みソフトウェア開発の効率を向上させることである. そのため, 本研究において, TECS コンポーネント記述言語と C 言語それぞれの解析結果を統合して, ソフトウェア全体のフロー解析結果を示すことのできるツールを開発する. また, フロー解析による情報を利用して, 排他制御が必要な箇所の検知や, 過剰な実装の検知を行い, 検証の効率化を図る機能も実装する. そして開発したツールを利用し, ソフトウェア全体が期待通りに実装されているかどうかを検証する.

## 2. TECS

TECS は, TOPPERS プロジェクトにより開発された, 組込み用途に特化したコンポーネントシステムである. TECS が既存のコンポーネントシステムと異なる点は, コンポーネントを静的に生成・結合している点である. TECS では, コンポーネント生成と結合の記述に TECS コンポーネント記述言語（TECS CDL）, コンポーネントの振舞いの記述に C 言語を採用している. 図 1 に TECS のコンポーネント図の例を示す.

TECS では生成すなわちインスタンス化されたコンポーネントをセルと呼び, セルの型, すなわちセルの基本的な構成要素をセルタイプと呼ぶ. セルは, 自身の機能を提供するためのインタフェースである受け口, 他のセル（自セルも可）が提供する機能を利用するためのインタフェースである呼び口を利用して結合する. セルは付随情報を表す属性（定数）, セルの内部状態を表す変数（以下, 内部変数と呼ぶ）を持つ.

### 2.1 TECS コンポーネント記述

セルタイプの定義およびセルの結合関係は, TECS CDL で記述する. TECS CDL 記述は, シグニチャ記述, セルタイプ記述, 組上げ記述からなる.

シグニチャは, 関数ヘッダの集合であり, 各セルが提供する機能または要求する機能の集合を表す [3]. シグニチャは呼び口および受け口に対応付けられ, セル間のインタフェースを規定するものである. すなわち, 他セル（自セルも可）の機能（受け口）を利用するためには, 機能を要求するセルに, 同じシグニチャの呼び口が必要であり, その呼び口を介することが必要となる. 呼び口と受け口を一対多で結合させる場合には, 呼び口を配列にして使用する. これを呼び口配列と呼ぶ. 一方, 受け口は任意の数の呼び口と結合することができる. これを合流と呼ぶ. 複数の呼び口と結合した受け口が呼び元を判別するためには, 受け口配列を用いる. 呼び口配列と受け口配列は, 配列添え字によって結合先の呼び口を判別する.

```
1: cell  tMotor MotorMain {
2:       cController  = Sample.eController;
3:};
```

リスト 1　組上げ記述例

```
1:ER
2:eBridge_connect(CELLIDX idx, int32_t con){
3:      /* 中略 */
4:      cMain_calc(con); /* 受け口関数の呼び出し */
5:      VAR_param = con*100; /* 内部変数の利用 */
6:      printf(”%d”, i); /* 非受け口関数の呼び出し */
7:      return(ercd);
8:}
```

リスト 2　セルタイプコード例

セルタイプ記述において, 1 つのセルタイプ情報として, 結合に用いるシグニチャと対応する受け口名および呼び口名, 属性の型と名前, 内部変数の型と名前が定義される. また, 複数のセルを組み合わせて, 新しいセルタイプを生成することもできる. これを複合セルタイプと呼ぶ. 複合セルタイプのインスタンスを複合セルと呼ぶ.

組上げ記述は, セルタイプをインスタンス化してセルを生成し, セル同士の結合関係を定義してソフトウェアを構成するものである. 同じセルタイプから生成したセルでも, 組上げ記述によってそれぞれ異なる結合先セルを指定できる. 内部変数は, 組上げ記述においては初期値を与えることができない, セル単位のインスタンス変数である. 組上げ記述の例をリスト 1 に示す. リスト 1 は, セル MotorMain の呼び口 cController とセル Sample の 受け口 eController との結合関係を表す.

### 2.2 TECS を用いた開発の流れ

TECS を用いたソフトウェア開発では, TECS CDL で記述したコンポーネント定義ファイルを, TECS ジェネレータと呼ばれるツール（以下, tecsgen と呼ぶ）に入力することで, インタフェースコードとヘッダファイル（以下, この 2 つをまとめてヘッダと呼ぶ）, テンプレートコードが生成される. このテンプレートコードをもとに, コンポーネントの提供する機能を C 言語で実装したものがセルタイプコードである. ヘッダとセルタイプコードをコンパイル・リンクすることでアプリケーションモジュールが完成する.

セルタイプコードのテンプレートは, セルタイプ記述で定義された各受け口に含まれる関数（受け口関数）に対して生成される. セルタイプコードでは, セルの結合にかかわる処理の記述は呼び口の名前のみを指定し, 呼び先セル名は記述しない. そのため, セルタイプコードだけではセルの結合関係を知ることはできない. ヘッダに記述されているセルの呼び口関数マクロを利用することで, 結合先のセルの受け口関数を呼び出すことができる [6]. また, 同じくヘッダ情報である関数テーブルや受け口ディスクリプタの情報を利用し, 呼び口・受け口配列の結合情報を保持する. セルタイプコードの例をリスト 2 に示す. リスト 2 では, 受け口 eBridge から利用できる関数 connect が, 呼び口 cMain から関数 calc を呼び出すことを表す.

## 3. TECS-flow の仕様と実現方針

### 3.1 利用目的

ソフトウェアにおける動作フローに対して静的解析を行うことで, ソフトウェア検証が容易になる可能性があるが, TECS にはソフトウェア全体の動作フローをモデル化する手段は存在しない. そこで, TECS を用いたソフトウェアのフロー解析を行い可視化するツール TECS-flow を開発する. フロー解析ツールを用いることで, ツールによる動作検証が可能となり, さらに検証を効率化できると考える. 本研究では, 動作検証の一例として, 排他制御の不足箇所やデッドコードの検出を TECS-flow の提供する機能として開発する.

本ツールの利点は, TECS を使用して開発されたソフトウェアを, セルタイプコード単位ではなく, セルの呼び出し関係と合わせたソフトウェア全体のフローを可視化できる点である. セルタイプコードはセル同士の呼び出しに関する情報として呼び口名しか保持しておらず, 結合先セルの判別に必要な情報はヘッダファイルやインタフェースコードが保持している. さらに, セルタイプの構成要素やセルごとの設定などの情報は コンポーネント定義ファイルが保持している. このように, TECS を利用して開発したソフトウェアは, 情報を様々な形式に分けて保持している. そのため, 既存の C 言語解析ツールでは TECS を利用して開発したソフトウェアのフロー解析を行うことができない. これに対し, TECS-flow を利用することで, セル間の呼び出し関係と関数呼び出し関係を統合し, それぞれの呼び出し関係の実装が正しく行われていること, またそれらを統合した結果が正しく実装されていることの確認を行うことができる. 加えて, 内部変数の競合アクセスが起きうる場所, すなわち排他制御が必要な箇所や, デッドコード検出といったオプション機能によって, ソフトウェアの検証を効率化することができる.

### 3.2 動作フローの可視化

TECS-flow の基本機能は, TECS を用いて開発したソフトウェア全体の動作フローをテキスト形式で表示することである.

アクティブセルを動作フローの起点セルとする. これをルートセルと呼ぶ. フロー表示の際はルートセル名の前に '*' を表示して, 他セルと区別する. ルートセルを含めた呼び元セルの呼び口関数呼び出しから, 呼び先セルの受け口関数への呼び出し情報を, "呼び元セル名", "呼び口名", "->", "呼び先セル名", "受け口名", "受け口関数名" の順に表示する. 呼び元セル名の前に, それぞれのセル呼び出しの深さ×半角スペース 4 つをインデントとして表示する. セル呼び出しの深さとは, どこからも呼ばれていないセルをルートセルとした時に, そのルートセルから経由するセルの数を表す. 呼び口配列・受け口配列の場合は配列名の後ろに配列添字を付加する. 複合セルとの結合部分は, 複合セル名とその呼び口または受け口名と, 内部セル名とその呼び口または受け口名を "=>" で繋げて記述する. 複合セル内部セル名は, "複合セル名", '_', "内部セル名" の形式で表す.

なお, コンポーネント図上や組上げ記述上では結合関係にあっても, セルタイプコード内すなわち関数レベルで呼び出しが行われていない場合, 呼び出しフローは表示されない.

### 3.3 動作フローの検証

TECS-flow は, フロー表示に加えて, 実装の検証効率化機能も持つ. なお, セルタイプコードやコンポーネント定義記述の修正を必要とする問題を検知する場合は, 問題の検知のみにとどまり, 実際のセルタイプコードやコンポーネント定義記述の修正作業は開発者が行う方針とする.

#### 3.3.1 排他制御を要する場所の検知

内部変数は同一のセル内部で参照でき, 実行時に書き換え可能な変数であるため, 複数のアクティブセルから同時にアクセスされる場合に, アクセス競合が発生する可能性がある. そのため, アクセス競合が起こりうる場所において, 排他制御を行うことが必要になる. アクセス競合発生の可能性がある場所を知るためには, ソフトウェア中に存在するすべての処理単位がどのタイミングでどの変数にアクセスするかを把握する必要があるため, 人手で発見することは困難である. 本機能によって, アクセス競合が発生する可能性のある箇所かつ, 排他制御が不足している関数名を判別し, 警告する. 既に排他制御が行われていると判断した場合は警告しない.

#### 3.3.2 デッドコード検出

デッドコードは, バグのもとになる場合や, リソースの無駄遣いになる場合があるため, 好ましくない. 本機能によって, どこからも呼び出されないコードを判別し, 警告する.

### 3.4 TECS-flow 実現方針

TECS-flow を実現するために, 次の方針を考えた. まず, コンポーネント定義ファイルからセル呼び出し関係情報を得る. 次に, ヘッダとセルタイプコードから, C フロー情報を得る. 最後に 2 つの情報を統合し, ソフトウェア全体のフローを出力する. この方針を基に行った実装の詳細を次章で述べる.

## 4. TECS-flow の開発

### 4.1 開発方針

TECS-flow の主機能は, コンポーネント結合情報と関数フロー情報を入力し, これら 2 つの情報を統合して出力することである. この情報統合機能部分が, TECS-flow の本体である. コンポーネント結合情報は コンポーネント記述の構文解析, 関数フロー情報は C 言語の構文解析を行う必要がある. 構文規則は言語ごとに固有のものであり, 構文解析機能自体はツールごとに大きな違いはない. そのため, コンポーネント記述ファイルと C ファイルの解析には既存のツールを利用することにした. よってツールの新規開発部分は情報統合機能部分のみにとどめ, コンポーネント定義ファイルと C ファイルを解析してツールへの入力ファイルを生成するために, tecsgen と TCFlow という, 2 つの既存ツールに対し変更開発を行うことで, ツールの開発コストを低減する.

TCFlow は, C 言語のソースファイルから関数の呼び出しツリーおよび参照グローバル変数のリストを出力する, C 言語解析ツールである. TCFlow 実行時に, C プリプロセッサを呼び出すことができ, ここでヘッダファイル等をインクルードできる. このため, TCFlow 実行時にセルタイプコードに加えてヘッダをインクルードすることで, ヘッダの持つセル呼び出し関係情報を TCFlow 実行結果に付加できる.

### 4.2 tecsgen の変更開発

tecsgen はコンポーネント定義ファイルを入力とし, ヘッダ, セルタイプコードのテンプレートを出力する. tecsgen はこれらのファイル生成の際にコンポーネント定義ファイルの解析結果を出力するオプション機能を持つ. この解析結果の中に, ソフトウェア全体の動作フローを解析するために必要なコンポーネント結合情報やコンポーネント定義情報が含まれている. そのため, tecsgen で行うコンポーネント定義ファイルの解析結果から, TECS-flow の入力として呼び元セル名と呼び口名, 呼び先セルと受け口名の組, セルとセルタイプの組などの情報を抽出し, 整形してファイルに出力するよう変更する. これをコンポーネント結合情報ファイルと呼ぶ. 同時に, 通常のヘッダとは別に, TECS-flow 向け情報生成のために TCFlow にインクルードする特別なヘッダも生成するように変更した. これは, TECS-flow に必要なコンポーネント結合に関する情報が, 通常のヘッダでは最適化 [7] によって省略されてしまうためである.

### 4.3 TCFlow の変更開発

ソフトウェア全体のフローを知るためには, コンポーネント同士の結合情報に加えて, コンポーネント内部の関数フローを知る必要がある. そのため, TCFlow の関数呼び出しツリー表示機能を利用して, 関数の呼び出しフローや関数ごとのインデントを抽出・整形し, TECS-flow 向けファイルを出力する機能を追加した. また, 第 3.3 項で述べた機能を実現するためには, 関数単位における内部変数のアクセス情報が必要となる. この情報は TCFlow の参照グローバル変数リスト出力で得られるため, こちらも TECS-flow 向けに変数名・読み出し回数・書き込み回数などを抽出・整形し, 関数フロー情報とともにファイルへ出力する機能を追加する.

### 4.4 TECS-flow の新規開発

tecsgen から出力されるコンポーネント結合情報ファイルと, TCFlow から出力される関数フロー情報ファイルの 2 つの入力情報を統合し, ソフトウェア全体のフロー情報を出力するのが TECS-flow である.

まず, フロー表示の起点となるルートセルを決定する. セル情報からアクティブフラグが 1 のセルを一つ選び, さらにそのセルが複合セルの内部セルかどうかを確認し, 内部セルの場合は複合セル名を保持する. 未表示のルートセルが無い場合, ツールは動作を終了する. ルートセルが決定したら, ルートセルから呼び出されるセルと関数名を確認する. この時, 呼び先セルが複合セルかどうかも確認する. 呼び元セルと呼び先セルの関係によって, その行の表示形式が決定する. 呼び出し深さ 1 以上のセル呼び出し関係表示の場合も, 同様に呼び元セルと呼び先セルの種類から表示形式を決定する. そのセルから別のセル（自セルも可）への受け口関数呼び出し表示がすべて完了した後, 非受け口関数を表示する. ひとつのルートセルからのフローを表示し終えたら, 次のルートセルを探す. 未表示のルートセルが無い場合, ツールは動作を終了する.

#### 4.4.1 排他制御必要性判定

排他制御が行われているかどうかは, TECS-flow 実行時に, 入力対象のコード内で排他制御用ロックとして使用したキーワードを指定する. 対象の関数内でこれらのキーワードを含んだ関数名が呼び出されているかどうかを調べる. 関数名のみを探索対象としているため, 名前にそのキーワードを含んだ関数が実際にロック処理を行っているかどうかまでは判断しない. これは, TCFlow が同じ深さの関数の場合, 実行順ではなく名前順に関数名を出力するためである.

#### 4.4.2 デッドコード検出

TECS-flow に入力された関数フロー情報のうち, TECS-flow が通常フロー表示を終了するまで表示されなかった関数をデッドコードと判断して一覧表示する.

## 5. ケーススタディ

本章では, TECS-flow を複数のプログラムに適用した結果を示す. 対象のプログラムは, 表 1 の 3 つを利用した.

### 5.1 動作フロー可視化

sample1_skyeye の動作フロー表示結果の一部をリスト 3 に示す.

セル MainTask をルートセルとした呼び出しフローが表示された. セル Sample1 から複合セル SerialPort への結合は "=>" を使用した, 複合セル特有の表示となっている. また, Sample1 から 4 種類のセルへの呼び出しを表す呼び口配列 cTask の結合も正しく表示された.

### 5.2 動作フロー検証機能

sample1_skyeye で各機能を実行した結果を述べる.

#### 5.2.1 排他制御が必要な場所の検知

本機能の検証は, 排他制御用キーワードに "ooo" という, 無意味な単語を指定した場合と, ディスパッチ禁止処理 "dis_dsp" を指定した場合の結果を比較する. このプログラムでは, 競合アクセスの可能性がある内部変数として, openFlag, sendStopped, receiveStopped, receiveFlowControl の 4 つが挙がる. この内部変数に対し, 前者では 4 つとも警告が出るが, 後者では dis_dsp でロック処理を行っている変数 openFlag の警告が消え, 警告は 3 つ分になった. よって, ロック処理が施された変数アクセス処理を区別し, 排他制御が必要な場所を検知することができた.

#### 5.2.2 デッドコード検出

この項では tTask のセルタイプコードを対象に説明する. tTask セルタイプコードで定義された受け口関数のうち, eTask_activate, eTask_cancelActivate, eTask_calcelWakeup がフロー表示結果として示された. tTask セルタイプコードでは, 上記の 3 つの関数に加え, eTask_refer, eiTask_activate, eiTask_wakeup, eiTask_releaseWait, eiTask_raiseException の 5 つの関数が定義されているが, これら 5 つの関数がどこからも呼ばれないデッドコードとして検出された.

## 6. おわりに

### 6.1 まとめ

本論文では, TECS を用いて開発したソフトウェアのフロー解析を行うツール TECS-flow の開発について述べた.

フロー解析を行うことで, テストを行う前に不具合を発見してテストプロセスのコスト削減や, テストでは表出しない不具合を発見することができ, またコードの最適化も行うことができる. このため, TECS-flow の仕様決定にあたり, 基本的なフロー解析だけでなく, コードの最適化を行うための機能も搭載することにした. TECS-flow の開発では, 2 つの既存ツール tecsgen と TCFlow に対する変更開発と, コンポーネントの結合情報と C 言語のフロー情報を統合する部分の完全新規開発の 2 種類の開発作業を行った.

開発した TECS-flow を利用して呼び出しフロー解析を行った結果, 本論文で用いたすべてのプログラムでも, コンポーネントと関数の呼び出し関係を統合した, プログラム全体の呼び出しフローが正しく表示されていることを確認した. また, 基本となるフロー表示に加え, 排他制御が必要な箇所の検知やデッドコード検出といった動作フロー検証機能の適用結果も示した. この結果から, 動作フロー検証機能を適用することで, 不具合の原因となりうる箇所の検知を行うことが可能であり, TECS-flow によってコンポーネントベース開発されたソフトウェアのコード最適化が行えることを示した.

### 6.2 今後の課題

今後の課題として, TECS-flow のオプション機能拡張と, 動作フロー表示の可読性向上が挙げられる. 拡張オプション機能として, デッドロック検知などが考えられる. 現在のオプション機能である, 排他制御が必要な箇所の検知機能を使用して警告が出た箇所にロック処理を追加した結果, デッドロックになってしまう可能性がある. このような場合に対応するため, 既存の検知機能にはロックの順番まで指定する機能を追加し, デッドロック検知機能を新たに実装することが考えられる.

現在の TECS-flow は複数回呼び出される関数の内容を, その都度すべて表示している. これは, 表示結果を部分的に確認する場合は有効であるが, 全体のフロー表示結果では, 同じ表示が何度も繰り返されることになるため, 結果表示量が多くなり, 可読性が低下する. このため, 繰り返し表示する回数に上限を設け, 上限を超えたら関数の中身は表示しないといった表示方法にすることが考えられる. 加えて, 関数の表示順は名前順ではなく, より実際の動作に近くなるように実行順で表示した方が有用性が高くなり, 排他制御が正しく行われているかの検証も正確になると考えられる.

## enPiT

本研究の一部は enPiT Emb[1] の一環として行った. enPiT は社会の課題を解決できる情報技術人材の育成を目的としたプロジェクトであり, enPiT Emb は組込みシステム分野の人材を育成するプログラムである. 名古屋大学では発展型 OJL（On the Job Learning） の形で実施した. これは, 大学と企業が連携し, 学生が企業に準じた開発スタイ

表 1　TECS-flow 適用対象プログラム

| プログラム名 | 対象 | 概要 |
|---|---|---|
| EX09_JOYSTICK + LCD + SD-CARD | STM32Primer2 | TECS 教材 |
| sample1_skyeye | Skyeye シミュレータ | ASP+TECS パッケージ同梱物 |
| sample_etrobo | MINDSTORMS NXT 走行体 | ET ロボコン用走行体向け |

```
 1: *MainTask.cBody->Sample1.eMainTask.main
 2: /* 中略 */
 3:     Sample1.cSerialPort->SerialPort1.eSerialPort=>SerialPort1_SerialPortMain.eSerialPort.control
 4:         sns_dpn
 5:     Sample1.cSerialPort->SerialPort1.eSerialPort=>SerialPort1_SerialPortMain.eSerialPort.open
 6:         SerialPort1_SerialPortMain.cSIOPort=>SIOPortMain.eSIOPort->SIOPortTarget.eSIOPort=>
     SIOPortTarget_SIOPortMain.eSIOPort.enableCBR
 7:         /* 内部関数省略 */
 8:         SerialPort1_SerialPortMain.cSIOPort=>SIOPortMain.eSIOPort->SIOPortTarget.eSIOPort=>
     SIOPortTarget_SIOPortMain.eSIOPort.open
 9:         /* 内部関数省略 */
10:     Sample1.cSerialPort->SerialPort1.eSerialPort=>SerialPort1_SerialPortMain.eSerialPort.read
11:         SerialPort1_SerialPortMain.cReceiveSemaphore->SerialPort1_ReceiveSemaphore.eSemaphore.signal
12:             sig_sem
13:/* 中略 */
14:     Sample1.cTask[0]->MainTask.eTask.activate
15:         act_tsk
16:     Sample1.cTask[1]->Task1.eTask.activate
17:         act_tsk
18:     Sample1.cTask[2]->Task2.eTask.activate
19:         act_tsk
20:     Sample1.cTask[3]->Task3.eTask.activate
21:         act_tsk
```

リスト 3　sample1_skyeye の実行結果（一部）

ルで企業が持ち込んだ開発テーマを, プロジェクトマネージャー（以下, PM）の指導の下に実践するものである. 具体的には, 開発目標達成のための開発プロセス設定, 開発スケジュール立案, 要求仕様書や開発書といった文書作成などを実施する.

本研究では, 開発ツールの要求分析から基本機能開発までを OJL の形式で行った. 最初に開発プロセスとスケジュールを設定し, 企業担当者や PM とのミーティングと文書作成を定期的に行いながら開発を進めた. OJL によって, 企業と大学の開発スタイルの違いを体感し, 文書作成の難しさと重要性, 柔軟なリスケジュールの必要性を学んだ.

## 参考文献

[1] enPiT Emb. http://emb.enpit.jp.
[2] IEEE Computer Society. *The Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) V3.0* . 2014.
[3] NPO 法人 TOPPERS プロジェクトコンポーネント仕様ワーキンググループ. 組込みコンポーネントシステム TECS 仕様書, may 2011.
[4] TOPPERS プロジェクト:TECS. http://www.toppers.jp/tecs.html.
[5] Saddek Bensalem, Marius Bozga, Thanh-Hung Nguyen, Joseph Sifakis. Compositional Verification for Component-based Systems and Application. *Software, The Institution of Engineering and Technology*, 2010.
[6] 安積卓也. 組込みシステムに適したコンポーネントシステム. PhD thesis, 名古屋大学, 2009.
[7] 安積卓也, 山本将也, 小南靖雄, 高木信尚, 鵜飼敬幸, 大山博司, 高田広章. tecsgen の持つ，コンポーネント結合の最適化機能. 組込みシステムに適したコンポー ネントシステムの実現と評価, コンピュータソフトウェ ア, Vol. 26, No. 4(2009), pp. 3955.