情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-SLDM-144 No.11
Vol.2010-EMB-16 No.11
Vol.2010-MBL-53 No.11
Vol.2010-UBI-25 No.11
2010/3/26

RTOS のテスト自動生成システムに関する一考察

鴫原 一人[†] 森 孝夫[††] 本田 晋也[††]
山本 雅基[††] 高田 広章[††]

RTOS は組込みシステムの根幹をなすソフトウェアであるので，高い品質が求められる．RTOS のテストスイートは，使用する企業やその用途によって求められる範囲が異なる．したがって，RTOS のテストスイートは，テストの範囲を固定化したパッケージで提供しても，利用者が限られる．本論文では，利用者の求める範囲に応じて RTOS のテストスイートを生成する手法を提案する．RTOS のテストは，テストケースに直接影響するタスクやシステムをはじめとして取りうる条件が多く存在するので，テストの範囲に幅が出てしまう．そこで，テストケースに直接影響する条件をルール化し，直接影響しない条件をテスト範囲として与えることにより，利用者の求める範囲のテストスイートを自動生成する．

A Study about Automatic Generation Systems for RTOS Tests

Kazuto SHIGIHARA[†] Takao MORI[††] Shinya HONDA[††]
Masaki YAMAMOTO[††] and Hiroaki TAKADA[††]

As RTOS are the core of many embedded systems, they must be developed under high quality standards. Depending on the company or intended application, the scope of the test suite may differ. Test suites with a fixed scope are therefore limited to a few users. This paper proposes a method for generating RTOS test suites that can satisfy the demands of different users. RTOS test suites usually have many selectable parameters, apart from the test environment conditions. As a consequence, a variety of test cases appear. Our method generates automatically test suites to meet the demands of different users by defining rules for the test environment conditions and offering a variety of selectable parameters.

1. はじめに

名古屋大学大学院情報科学研究科附属組込みシステム研究センター(NCES)[1] は，TOPPERS プロジェクト[2]から ITRON 仕様[3]をベースとした組込みシステム向け RTOS をオープンソースで開発し公開している．2009 年度にはコンソーシアム型の研究組織を立ち上げ，複数の企業と団体の参加を得て，マルチプロセッサ対応 RTOS 対するテスト手法の検討とテストスイートの開発を実施している．

テストスイートを開発する過程で，RTOS のテストスイートは使用する企業やその用途によって求めるテストの範囲が異なることから，テスト範囲を固定化したパッケージでは利便性に欠け，利用者が限られるので，実用的ではない可能性が判明した．さらに，テスト範囲を固定化して作業すると，パッケージに含まれるドキュメントとデータファイルの開発や保守が困難になることも明確になった．そこで，我々は，RTOS 特有である抽象的な仕様の表現や API の対称性とテスト条件の共通パターンから，テスト範囲を動的に変更する開発と管理を行う手法を検討した．

本論文では，我々がコンソーシアム型の研究組織において実施したテストスイート開発で明確化した問題と，その問題を解決するためのテストスイート開発手法と，それに基づき作成したテスト自動生成システムについて考察する．

2. テスト範囲を固定化したテストスイート開発

本章では，我々が開発したテストスイートについて述べる．

2.1 テスト対象の RTOS

TOPPERS プロジェクトでは，信頼性と安全性とソフトウェアポータビリティを向上させるために，ITRON 仕様に改良を加えた TOPPERS 新世代カーネルを公開しており，その仕様を「TOPPERS 新世代カーネル統合仕様書[4]」としてまとめている．

TOPPERS 新世代カーネルには，シングルプロセッサ対応の TOPPERS/ASP カーネル(以下，ASP カーネル)と，マルチプロセッサ対応の TOPPERS/FMP カーネル(以下，FMP カーネル)がある．FMP カーネルのテストを目的とする我々は，その前段階として，最初に ASP カーネルに対するテストスイートの開発を行った．

2.2 テストスイート開発の概要

ASP カーネルが提供している 121 個の API が統合仕様書に定められている通りに正

† 富士ソフト株式会社
FUJISOFT INCORPORATED
†† 名古屋大学
Nagoya University

Vol.2010-SLDM-144 No.11
Vol.2010-EMB-16 No.11
Vol.2010-MBL-53 No.11
Vol.2010-UBI-25 No.11
2010/3/26

しく振舞うことのテストを,我々はAPIテストと呼ぶことにする.テストスイートは,APIテストの実施を目的として開発する.APIテストは,API発行前のシステム状態(前状態)を定義し，その状態でテスト対象となるAPIを発行(処理)し，API発行後のシステム状態(後状態)を確認するテストプログラムを実行するものである．API テストの作業フローを図 1 に示す.

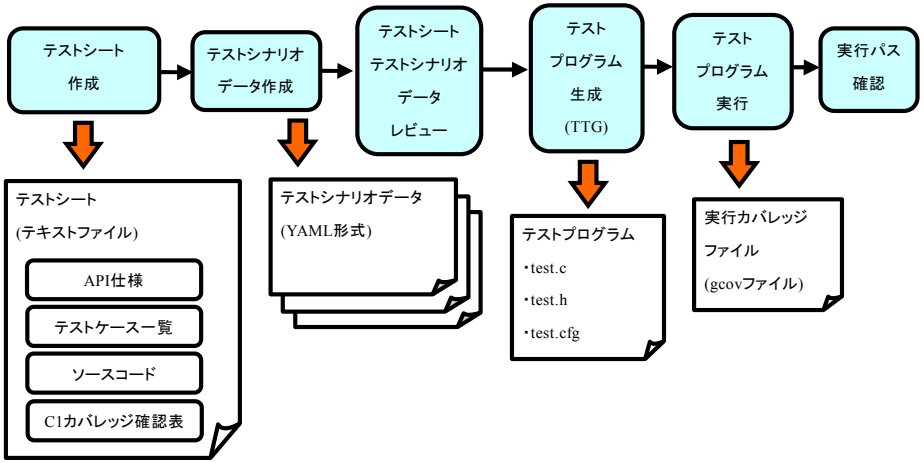

図 1　APIテスト作業フロー

APIテストでは，最初に，統合仕様書からテスト対象であるAPIの仕様と振舞いを理解し，テストケースを抽出する．統合仕様書ではAPIの仕様を，API毎の仕様をまとめた章(図 2)とカーネル全体の共通的な振舞いについてまとめた章(図 3)に分散して定義している．そのために，テストスイートの開発は，統合仕様書の全体的な理解が必要である．加えて，タスクの状態やシステム状態が取り得る組み合わせは膨大な量であるので，テストの生産性を高めるためには，テスト品質を保証しながら，テストをする条件を絞り込む必要がある．我々は，コンソーシアム型研究に参加している各企業の経験を持ち寄り，最低限テストすべき項目のみ実施できるよう，ポリシーを策定し，テストケース数増加を抑止している．図 4 に，テストケース抽出ポリシーから作成したテストケースの例を示す．さらに，各テストケースを実行することによるAPIのソースコードカバレッジを手作業で確認する．これらの情報をテストシートと呼ぶテキストファイルにまとめる.

次に，テストケース毎に YAML と呼ばれる形式化したフォーマットに従い，前状態(pre_condition)と，処理(do)と，後状態(post_condition)を定義したテストシナリオデータを作成する(図 5).

テストケースから手作業でテストプログラムを作成すると，作成者によってテストの実現方法がばらつくことが考えられる．ばらつきにより，テストプログラムレビュー時の可読性や修正時の保守性が著しく低下する．この問題を回避するために，我々はテストシナリオデータを階層型データ形式言語である YAML 形式で表現したテストケースから,テストプログラムを生成するツール TTG(TOPPERS Test Generator)を開発した．図 5 の例では，優先度の低い TASK1 と高い TASK2 が存在して，前状態では TASK1 を実行状態に TASK2 を休止状態にする．処理は，TASK1 が act_tsk を TASK2 に対して発行することである．処理の結果，後状態として，TASK2 が実行状態に，TASK1 が実行可能状態となることをテストすることを表している．TTG を用いて,テストシナリオデータを入力値として,テストプログラムをテストケース毎に生成する.

最後に，テストプログラムを実行して APIテストを行う．同時にコードカバレッジを取得するツールである gcov を用いてテストプログラムのカバレッジを取得し,手作業で求めたカバレッジと照らし合わせ，テストプログラムがテストケースで想定したパスを通過していることを確認する.

```
00: act_tsk     タスクの起動
01: 【C言語API】
02:     ER ercd = act_tsk(ID tskid)
03: 【パラメータ】
04:     ID          tskid       対象タスクのID番号
05: 【リターンパラメータ】
06:     ER          ercd        正常終了(E_OK)またはエラーコード
07: 【エラーコード】
08:     E_CTX               コンテキストエラー(非タスクコンテキストからの呼出し,
                            CPUロック状態からの呼出し)
09:     E_ID                不正ID番号(tskidが不正)
10:     E_QOVR              キューイングオーバフロー
                            (起動要求キューイング数がTMAX_ACTCNTに一致)
11:
12: 【機能】
13: tskidで指定したタスク(対象タスク)に対して起動要求を行う．具体的な振舞いは以下の通り.
14: 対象タスクが休止状態である場合には，対象タスクに対してタスク起動時に行うべき初期化処理
    が行われ，対象タスクは実行できる状態になる.
15: 対象タスクが休止状態でない場合には,対象タスクの起動要求キューイング数に1が加えられる.
16: 起動要求キューイング数に1を加えるとTMAX_ACTCNTを超える場合には，E_QOVRエラーとなる.
17: act_tskにおいてtskidにTSK_SELF(=0)を指定すると，自タスクが対象タスクとなる.
```

図 2　API(act_tsk)の仕様例

Vol.2010-SLDM-144 No.11
Vol.2010-EMB-16 No.11
Vol.2010-MBL-53 No.11
Vol.2010-UBI-25 No.11
2010/3/26

```
00: 第２章　主要な概念と共通定義
01: 2.1 仕様の位置付け
02: この仕様は，TOPPERS 新世代カーネルに属する各カーネルの仕様を，統合的に記述することを目
    標としている．また，TOPPERS 新世代カーネル上で動作する各種のシステムサービスに共通に適
    用される事項についても規定する．
      :
03: 2.3.1 オブジェクトと処理単位
04: オブジェクトは，種類毎に，番号によって識別する．カーネルまたはシステムサービスで，オブ
    ジェクトに対して任意に識別番号を付与できる場合には，1 から連続する正の整数値でオブジェ
    クトを識別する．
      :
05: 2.6.3 タスクのスケジューリング規則
06: 実行できるタスクは，優先順位の高いものから順に実行される．すなわち，ディスパッチ保留状
    態でない限りは，実行できるタスクの中で最も高い優先順位を持つタスクが実行状態となり，他
    は実行可能状態となる．
07: タスクの優先順位は，タスクの優先度とタスクが実行できる状態になった順序から，次のように
    定まる．優先度の異なるタスクの間では，優先度の高いタスクが高い優先順位を持つ．優先度が
    同一のタスクの間では，先に実行できる状態になったタスクが高い優先順位を持つ．
      :
08: 2.5.8 ディスパッチ保留状態
09: 非タスクコンテキストの実行中，全割込みロック状態，CPU ロック状態，割込み優先度マスクが
    全解除でない状態，ディスパッチ禁止状態では，ディスパッチが保留される．これらの状態を総
    称して，ディスパッチ保留状態と呼ぶ．
      :
```

図 3　カーネル全体の共通仕様例

```
(a) 非タスクコンテキストから呼び出して，E_CTX エラーが返ること．
(b) CPU ロック状態で呼び出して，E_CTX エラーが返ること．
(c) tskid が不正の時に E_ID が返ること．
    (c-1) tskid が許容される最小値-1(=-1)の時に E_ID が返ること．
    (c-2) tskid が許容される最大値+1(=TNUM_TSKID+1)の時に E_ID が返ること．
(d) 対象タスクの起動要求キューイング数が TMAX_ACTCNT(=1)に一致している場合．
    (d-1) tskid に TSK_SELF(=0)を指定しない場合，E_QOVR が返ること．
    (d-2) tskid に TSK_SELF(=0)を指定すると，自タスクが対象タスクとなり，E_QOVR が返ること．
(e) 対象タスクが休止状態である場合には，対象タスクに対してタスク起動時
    に行うべき初期化処理が行われ，対象タスクは実行できる状態になること．
    (e-1) 対象タスクの優先度が実行状態のタスクより高い場合．
          (e-1-1) 実行状態になること．
          (e-1-2) ディスパッチ禁止状態の場合，実行可能状態になること．
          (e-1-3) 割込み優先度マスクが全解除でない場合，実行可能状態になること．
    (e-2) 対象タスクの優先度が実行状態のタスクより低い場合は，
          実行可能状態となり，同じ優先度のタスクの最後につながれること．
    (e-3) 対象タスクの優先度が実行状態のタスクと同じ場合は，
          実行可能状態となり，同じ優先度のタスクの最後につながれること．
(f) 対象タスクが休止状態でない場合，対象タスクの起動要求キューイング数に 1 が加えられること．
    (起動要求キューイング数が TMAX_ACTCNT(=1)に一致していない場合)
    (f-1) tskid に TSK_SELF(=0)以外を指定する場合．
          (f-1-1) 対象タスクが実行可能状態の場合．
          (f-1-2) 起床待ちの場合．
          (f-1-3) 時間経過待ちの場合．
          (f-1-4) セマフォの資源獲得待ち(タイムアウト無)の場合．
          (f-1-5) セマフォの資源獲得待ち(タイムアウト有)の場合．
          (f-1-6) 対象タスクが強制待ち状態の場合．
          (f-1-7) 二重待ち(起床待ち)の場合．
          (f-1-8) 二重待ち(時間経過待ち)の場合．
    (f-2) tskid に TSK_SELF(=0)を指定する場合．
```

図 4　act_tsk のテストケース

```
ASP_task_manage_act_tsk_e_1_1:
  pre_condition:                          ┐
    TASK1:                                │
      type   : TASK                       │
      tskpri : MID                        │
      tskstat: running                    │  API 発行前の
                                          │  システム状態
    TASK2:                                │
      type   : TASK                       │
      tskpri : HIGH                       │
      tskstat: dormant                    ┘

  do:                                     ┐
    - id: TASK1                           │
      syscall: act_tsk(TASK2)             │  API 発行処理
      ercd: E_OK                          ┘

  post_condition:                         ┐
    TASK1:                                │
      tskstat: ready                      │  API 発行後の
                                          │  システム状態
    TASK2:                                │
      tskstat: running                    ┘
```

図 5　テストシナリオデータの例

3. テスト範囲を固定化した開発の問題点

(1) テストケース抽出ポリシーの固定化

今回開発したテストスイートでは，策定したテストケース抽出ポリシーに従いテストの範囲を固定している．しかし，テストしたい項目つまり求めるテストの範囲は RTOS の利用者によって異なるので，個々の要求への対応が困難である．

(2) テストケース間での抽出範囲の不一致

RTOS では，異なる API でもタスクを同じ状態へ遷移をさせるものが存在する．例えば，タスクを起床待ちから実行可能状態へ遷移させる API は，wup_tsk(タスクの起床)と rel_wai(タスクの待ち強制解除)の 2 つがある．つまり API 間には対称性があり，同じ振舞いを確認するテストケースは，API が異なっても同じテストケースが必要である．

しかしながら，テストポリシーを策定したとしても，担当者毎の認識のずれや理解不足で，テストケースの抽出範囲の不一致が発生する恐れがある．

(3) テストシート，テストシナリオデータの保守性

テストケースを識別するテスト ID は，各 API のテストケース毎に(a)からアルファ

Vol.2010-SLDM-144 No.11
Vol.2010-EMB-16 No.11
Vol.2010-MBL-53 No.11
Vol.2010-UBI-25 No.11
2010/3/26

ベット順にテスト番号を付与しているので，テストケースの増減が発生した場合に，テストケースやテストシナリオデータのすべての番号を付与し直す必要があり，保守性が低下する．さらに，テストシートとテストシナリオデータはテキストファイルで管理されているので，テストポリシーの変更があると，対応するすべての API への対応が必要となる．

例えば，システム状態の 1 つとして定義されている「ディスパッチ保留状態」には，「CPU ロック状態」,「ディスパッチ禁止状態」,「割込み優先度マスクが全解除でない」などの複数の要因が存在するが，この内どれをテストケースとして抽出するかというポリシーが変わるだけで，該当するすべての API のテストシート，テストシナリオデータに対する修正作業が発生してしまう．

(4) テストケースとテストシナリオデータの一貫性確保

テストケースから実際にテストを実行するためのテストシナリオデータを作成するという作業を行っているので，テストケースに変更があると，テストシナリオデータを修正する必要があり，修正忘れが起きる可能性がある．

(5) テストケースの冗長性

複数の異なる前状態において同じ API を発行すると，一つの後状態になる場合がある．これらの異なる前状態に対してテストを行う場合には，状態の数だけのテストケースとテストシナリオデータが必要である．しかし，これらの異なる状態を抽象化して管理し，テストケース数を限定することが可能である場合では，テストケースが冗長となり保守性が悪化する．

(6) テストケース間の文言の不一致

API 毎に担当者を割り振り，日本語を用いてテストシートの作成を行うので，同一内容のテストケースであっても，文言が一致しない場合がある．日本語の文言の不一致により，テストシナリオデータが異なる内容になる危険性を排除するために，文言の統一が必要である．しかし，API は 121 個存在するので，全ての文言を人手で一致させることには，手間がかかることが予測される．

(7) マルチプロセッサ対応 RTOS のテストケース

マルチプロセッサ対応 RTOS の場合に，プロセッサの組み合わせまで考慮してテストケースを作成すると，テストポリシーで抑制しても膨大なテストケース数となってしまう．さらに，プロセッサが異なるだけでテスト内容が同じであるテストケースを，別のテストケースとして開発し管理することは非効率的である．

(8) 統合仕様書とテストスイートのマッピング

各テストケースが統合仕様書のどの記載に対応しているのかマッピングできていないので，統合仕様書が改訂された場合に，どのテストケースに修正が必要か確認しなければならず，生産性が低下する一因となっている．

4. 問題解決のための手法

本章では，3 章で取り上げた問題点を解決するため，統合仕様書，テストスイートを分析し，抜本的解決を図るための手法を提案する．

4.1 統合仕様書とテストスイートの分析

統合仕様書は，対称性のある API に対する仕様を効率良く説明するため，抽象的な表現を使用する場合がある．例えば，図 2 の「休止状態でない場合」は，具体的には，実行可能状態や待ち状態など複数のタスクの状態を意味する．また，図 3 の「非タスクコンテキストの実行中」は，具体的には，アラームハンドラ実行中や CPU 例外実行中などの可能性がある．

具体的なテストプログラムを作成するために，作成担当者は，統合仕様書に暗に記述されている複数の状態から一つを選択し，具体的なテストケースを作成する必要がある．ここに，作成担当者間のばらつきが発生する可能性がある．我々は，ばらつきを抑止するためにテストケース抽出ポリシーを定めたが，3 章で示した(2)や(6)の問題を 100%防ぐことは困難である．

また，(1)に挙げたようにテストケース抽出ポリシーが固定されているので，ポリシーを変更するたびに，(3)，(4)，(5)の問題が発生する．さらに，(7)と(8)の問題はこれらの原因の延長上にあるものであり，現状のテストスイート開発の手法では，改善する術がない．

4.2 仕様とテストケースのルール化

前節の分析は，すべての問題の主たる原因は抽象的に表現された仕様を具体化する作業にあることを示唆する．そこで我々は，仕様を抽象化したままでテストケースを作成する手法の検討を実施し，この問題に対処することとした．例えば，図 4 の(f-1-1)～(f-1-8)は「休止状態でない場合」を具体化するために発生したテストケースであり，「休止状態でない場合」を定義できれば管理上のテストケースは 1 つで済む．

表 1 は，図 2 と図 3 に例示した仕様のテストケースを抽出するのに必要な要素を示す．ここで，識別コードとは各仕様に割り当てた便宜上の名前であり，抽象化コードとは対応する各識別コードを包括するものである．例えば，抽象化コードの

Vol.2010-SLDM-144 No.11
Vol.2010-EMB-16 No.11
Vol.2010-MBL-53 No.11
Vol.2010-UBI-25 No.11
2010/3/26

ABST_SYS_ALL は仕様の「ディスパッチ保留状態」を，ABST_NOT_DORMANT は「休止状態でない場合」を示している．なお，全割込みロック状態はターゲット依存であるので，ディスパッチ保留状態の対象外としている．

表 1 で定義した仕様はテストケースを抽出する上での要素となるので，テストケースエレメントと呼ぶ．本質的にテストするべき項目の抽出作業はテストケースエレメントの組み合わせ作業となるはずであり，テストケースエレメントを組み合わせた抽象化されたテストケースのことをテストケースモデルと呼ぶ．テストケースモデルの抽象化コードを包括される識別コードに展開し，具体的な状態や数値に置き換えて展開したものが，テストケースとなる．

表 1　act_tsk のテストで使用するテストケースエレメント

種別	状態，条件	識別コード	抽象化コード	
システム状態	通常状態	-		
	非タスクコンテキスト実行中	SYS_NON_TASK	ABST_SYS_1	ABST_SYS_ALL
	CPU ロック状態	SYS_CPU_LCK		
	ディスパッチ禁止状態	SYS_DIS_DSP	ABST_SYS_2	
	割込み優先度マスクが全解除でない	SYS_INT_MASK		
タスク	自タスク(API を発行するタスク)	TASK_SELF	ABST_TASK_S_OA	
	他タスク(A)	TASK_OTHER_A		
	他タスク(B)	TASK_OTHER_B		
アラームハンドラ	動作中	ALARM_STA		
	停止中	ALARM_STP		
	実行中	ALARM_ACT		
タスク [状態]	実行状態	TSTAT_RUNNING		
	休止状態	TSTAT_DORMANT	ABST_TSTAT_ALL	
	実行可能状態	TSTAT_READY		ABST_NOT_DORMANT
	起床待ち状態	TSTAT_SLEEP		
	時間経過待ち状態	TSTAT_DELAY		
	セマフォ待ち(タイムアウト無)	TSTAT_SEM_N		
	セマフォ待ち(タイムアウト有)	TSTAT_SEM_T		
	強制待ち状態	TSTAT_SUSPEND		
	二重待ち状態(起床待ち)	TSTAT_SUS_SLEEP		
	二重待ち状態(時間経過待ち)	TSTAT_SUS_DELAY		
タスク	1	TACTCNT_1	ABST_TACTCNT_ALL	
[起動要求キューイング数]	0	TACTCNT_0		
タスク [優先度]	高	TPRI_HIGH	ABST_TPRI_ALL	
	中	TPRI_MID		ABST_TPRI_M_L
	低	TPRI_LOW		
タスク [優先順位]	1	TORDER_1		
	2	TORDER_2		
	3	TORDER_3		
引数	許容される最小値-1	ARG_MIN_MINUS_1	ABST_ARG_LIMIT	
	許容される最大値+1	ARG_MAX_PLUS_1		

4.3 act_tsk による手法の検証

本節では，act_tsk のテストケースモデルの作成を例にして，前節で定義したテストケースエレメントの有効性を図 4 に示した現状のテストケースとの比較により行う．表 2 は，図 2 と図 3 の仕様に仕様番号を付番して整理しなおしたものを示す．ただし，記述箇所の”:”記号に続く数字は各図中の行番号を示す．

表 2　act_tsk に関する仕様のナンバリング

仕様番号	仕様	記述箇所
[1-1]	正常終了(E_OK)	図 2：06
[1-2]	E_CTX コンテキストエラー(非タスクコンテキストからの呼出し，CPU ロック状態からの呼出し)	図 2：08
[1-3]	E_ID 不正 ID 番号(tskid が不正)	図 2：09
[1-4]	E_QOVR キューイングオーバフロー(起動要求キューイング数が TMAX_ACTCNT に一致)	図 2：10
[1-5]	対象タスクが休止状態である場合には，対象タスクに対してタスク起動時に行うべき初期化処理が行われ，対象タスクは実行できる状態になる．	図 2：14
[1-6]	対象タスクが休止状態でない場合には，対象タスクの起動要求キューイング数に 1 が加えられる．	図 2：15
[1-7]	起動要求キューイング数に 1 を加えると TMAX_ACTCNT を超える場合には，E_QOVR エラーとなる．	図 2：16
[1-8]	act_tsk において tskid に TSK_SELF(=0)を指定すると，自タスクが対象タスクとなる．	図 2：17

Vol.2010-SLDM-144 No.11
Vol.2010-EMB-16 No.11
Vol.2010-MBL-53 No.11
Vol.2010-UBI-25 No.11
2010/3/26

[2-1]	オブジェクトに対して任意に識別番号を付与できる場合には，1 から連続する正の整数値でオブジェクトを識別する．	図 2：13
[2-2]	ディスパッチ保留状態でない限りは，実行できるタスクの中で最も高い優先順位を持つタスクが実行状態となり，他は実行可能状態となる．	図 3：06
[2-3]	優先度の異なるタスクの間では，優先度の高いタスクが高い優先順位を持つ．優先度が同一のタスクの間では，先に実行できる状態になったタスクが高い優先順位を持つ．	図 2：07
[2-4]	非タスクコンテキストの実行中，全割込みロック状態，CPU ロック状態，割込み優先度マスクが全解除でない状態，ディスパッチ禁止状態では，ディスパッチが保留される．これらの状態を総称して，ディスパッチ保留状態と呼ぶ．	図 2：09

表 3 に，act_tsk のテストケースモデルと対応する仕様番号をまとめたものを，(A)から順に ID を振り列挙する．

表 3　act_tsk のテストケースモデルと対応する仕様番号

テストケースモデル	対応仕様番号	補足
(A)	[1-1]，[1-5]，[2-2]	-
(B)	[1-1]，[1-5]，[2-3]	-
(C)	[1-1]，[1-5]，[2-4]	-
(D)	[1-1]，[1-6]	-
(E)	[1-1]，[1-6]，[1-8]	「休止状態でない」は実行状態も含まれるので，自タスクに対するテストも必要である．
(F)	[1-2]	「E_CTX」が返る条件は，「非タスクコンテキストからの呼出し」と「CPU ロック状態からの呼出し」だけであり，他の条件について仕様は言及していない．
(G)	[1-3]，[2-1]	境界値分析法から，不正 ID 番号は「許容される最小値-1」と「許容される最大値+1」とする．
(H)	[1-4]，[1-7]	「E_QOVR」は対象タスクが「休止状態でない」場合に発生する．
(I)	[1-4]，[1-7]，[1-8]	「休止状態でない」は実行状態も含まれるので，自タスクに対するテストも必要である．

表 3 で作成したテストケースモデルを，表 1 のテストケースエレメントを用いて記述したものを表 4 に示す．また，それぞれのテストケースモデルに対応する現状のテストケース番号も記載する．

表 4　act_tsk のテストケースモデル

テストケースモデル	前状態	処理 (引数/戻り値)	後状態 (変化のない状態は記載しない)	現状のテストケース
(A)	TASK_SELF: TSTAT_RUNNING | TPRI_MID TASK_OTHER_A: TSTAT_DORMANT | TPRI_HIGH	TASK_OTHER_A E_OK	TASK_SELF: TSTAT_READY TASK_OTHER_A: TSTAT_RUNNING	(e-1-1)
(B)	TASK_SELF: TSTAT_RUNNING | TPRI_MID TASK_OTHER_A: TSTAT_DORMANT | ABST_TPRI_M_L TASK_OTHER_B: TSTAT_READY | ABST_TPRI_M_L	TASK_OTHER_A E_OK	TASK_OTHER_A: TSTAT_READY | TORDER_2 TASK_OTHER_B: TORDER_1	(e-2) (e-3)
(C)	ABST_SYS_2 TASK_SELF: TSTAT_RUNNING | TPRI_MID TASK_OTHER_A: TSTAT_DORMANT | TPRI_HIGH	TASK_OTHER_A E_OK	TASK_OTHER_A: TSTAT_READY	(e-1-2) (e-1-3)
(D)	TASK_SELF: TSTAT_RUNNING | TACTCNT_0 TASK_OTHER_A: ABST_NOT_DORMANT | TACTCNT_0	TASK_OTHER_A E_OK	TASK_OTHER_A: TACTCNT_1	(f-1-*)
(E)	TASK_SELF: TSTAT_RUNNING | TACTCNT_0	TASK_SELF E_OK	TASK_SELF: TACTCNT_1	(f-2)
(F)	ABST_SYS_1 TASK_SELF: TSTAT_RUNNING TASK_OTHER_A: TSTAT_DORMANT	TASK_OTHER_A E_CTX		(a) (b)
(G)	TASK_SELF: TSTAT_RUNNING TASK_OTHER_A: TSTAT_DORMANT	ABST_ARG_LIMIT E_ID		(c-1) (c-2)
(H)	TASK_SELF: TSTAT_RUNNING TASK_OTHER_A: ABST_NOT_DORMANT | TATT_ACTCNT_1	TASK_OTHER_A E_QOVR		(d-1)
(I)	TASK_SELF: TSTAT_RUNNING | TATT_ACTCNT_1	TASK_OTHER_A E_QOVR		(d-2)

表 4 は，テストケースエレメント導入前の現状のテストケースが，新しいテストケースモデルで定義されていることを示す．しかし，テストケースモデル(D)では「休止状態でない」場合について(f-1-*)のテストケースですべて実施しているのに対し，(H)については，(d-1)の 1 つしか実施していない．これはテストケース抽出ポリシーに不備があり，等しい観点でテストケースを抽出できていないことを示している．テストケースモデル(H)から抽象化コードを展開すると表 5 のように 8 個のテストケースモ

Vol.2010-SLDM-144 No.11
Vol.2010-EMB-16 No.11
Vol.2010-MBL-53 No.11
Vol.2010-UBI-25 No.11
2010/3/26

デルとなる．しかし，実際の(d-1)のテストシナリオデータは，対象タスクを実行可能状態としているので，表 5 の(H-1)しか実施できていないことになる．

表 5　テストケースモデル(H)の抽象化コード展開

テストケースモデル	前状態	処理（引数/戻り値）	後状態（変化のない状態は記載しない）	現状のテストケース
(H-1)	TASK_SELF: TSTAT_RUNNING TASK_OTHER_A: TSTAT_READY | TATT_ACTCNT_1	TASK_OTHER_A E_QOVR		(d-1)
(H-2)	TASK_SELF: TSTAT_RUNNING TASK_OTHER_A: TSTAT_SLEEP | TATT_ACTCNT_1	TASK_OTHER_A E_QOVR		-
(H-3)	TASK_SELF: TSTAT_RUNNING TASK_OTHER_A: TSTAT_DELAY | TATT_ACTCNT_1	TASK_OTHER_A E_QOVR		-
(H-4)	TASK_SELF: TSTAT_RUNNING TASK_OTHER_A: TSTAT_SEM_N | TATT_ACTCNT_1	TASK_OTHER_A E_QOVR		-
(H-9)	TASK_SELF: TSTAT_RUNNING TASK_OTHER_A: TSTAT_SEM_T | TATT_ACTCNT_1	TASK_OTHER_A E_QOVR		-
(H-10)	TASK_SELF: TSTAT_RUNNING TASK_OTHER_A: TSTAT_SUSPEND | TATT_ACTCNT_1	TASK_OTHER_A E_QOVR		-
(H-11)	TASK_SELF: TSTAT_RUNNING TASK_OTHER_A: TSTAT_SUS_SLEEP | TATT_ACTCNT_1	TASK_OTHER_A E_QOVR		-
(H-12)	TASK_SELF: TSTAT_RUNNING TASK_OTHER_A: TSTAT_SUS_DELAY | TATT_ACTCNT_1	TASK_OTHER_A E_QOVR		-

4.4　手法による問題点の解決

本節では 3 章で挙げた問題点を，提案した手法によって解決する方法について述べる．

(1) テストケース抽出ポリシーの固定化

テストケースモデルを定義したことで，テストケース抽出ポリシーに変更があった場合でもテストケースの変更と追加が容易に可能である．つまり，テストケース抽出ポリシーの不備が発見された場合にはポリシーが変更されるが，その変更にも柔軟に対応し得る．例えば，前節で説明したように表 5 の(H-1)しか実施できていなくても，TSTAT_READY を ABST_NOT_DORMANT へ変更するだけで，必要とするテストケース，テストシナリオデータの動的な作成が可能である．

(2) テストケース間での抽出範囲の不一致

仕様番号[1-2]～[1-4]のようなエラーコードに対するテストや，[1-5]のような対象タスクを実行可能状態へ遷移させる振舞いに対するテストは，対称性のある他の API のテストケースモデルから漏れないように，テストケース抽出ポリシー自体もルール化し，テストケースモデルとの関連付けを行うことで(2)の問題も解決可能である．

(3) テストシート，テストシナリオデータの保守性

テスト ID はテストケースモデルを作成後に，テストケースとテストシナリオデータを生成する時点で動的にナンバリングすれば手作業で修正する必要はない．

(4) テストケースとテストシナリオデータの一貫性確保

テストケースモデルの情報を用いてテストケースとテストシナリオデータを生成することにより，両者の一貫性を常に保つ．

(5) テストケースの冗長性

テストケースモデル(D)や(H)のように抽象化コードを用いて，タスクの状態が異なるだけの複数のテストケースを 1 つのテストケースモデルにまとめ，情報の冗長を排除する．

(6) テストケース間の文言の不一致

テストケースエレメントの定義文(表 1)を利用すれば，日本語のテストケースを動的に表現することは可能であり，文言が一致する．

(7) マルチプロセッサ対応 RTOS のテストケース

表 1 の「自タスク」「対象タスク」というテストケースエレメントのように，新たに「自プロセッサ」「他プロセッサ」というテストケースエレメントを組み合わせることや，「自プロセッサ」「他プロセッサ」を抽象化しておくことで，具体化するときに取りうるすべてのプロセッサの組み合わせを実施することが可能である．

(8) 統合仕様書とテストスイートのマッピング

統合仕様書側の全記述をナンバリングし，さらにテストケースモデルと関連付けることにより，仕様が変更された時に影響を受けるテストケースモデルの検出が容易となる．

Vol.2010-SLDM-144 No.11
Vol.2010-EMB-16 No.11
Vol.2010-MBL-53 No.11
Vol.2010-UBI-25 No.11
2010/3/26

5. テスト自動生成システム実現に向けた取り組み

5.1 TTE

本論文で取り上げた手法を実現するテスト自動生成システムは規模が大きいだけではなく，既に開発済みのテストスイートの修正も必要である．ただし，今後のテストスイートのメンテナンスや拡張性の観点から，コンソーシアム型研究では当初の目的であるテストスイート開発と並列して，本手法の考え方を取り入れたツール開発を段階的に実施する．

開発するツールは TTE(TOPPERS Test Expander)と呼称し，既に開発済みのテストシナリオデータと，新たに作成するポリシーファイルを入力し，複数の条件に対応したテストシナリオデータを出力する．図 6 は，TTE の処理イメージを示す．

現在 TTE は「OJL による最先端技術適応能力を持つ IT 人材育成拠点の形成」[5) 6)] の OJL(On the Job Learning)の開発テーマとして開発中である．

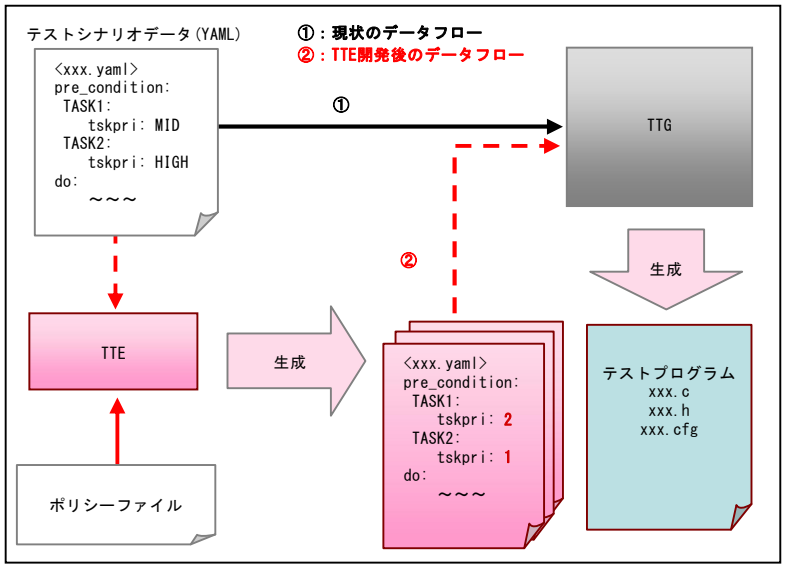

図 6　TTE の処理イメージ

5.2 今後の課題

本論文では，(2)と(8)以外の問題点を解決する手法を提案した．しかし，(2)のテストケース抽出ポリシーとテストケースモデルとの関連付けと，(8)の統合仕様書ナンバリングは具体的な手法については，今後の課題として残される．さらに，統合仕様書のすべての仕様に必要なテストケースエレメントの定義を，本論文で行った act_tsk の事例を参考として行う必要がある．

6. おわりに

本論文では，テスト範囲を固定化したテストスイート開発における課題と，課題の解決策となる手法の提案とテストスイート開発システムの実現可能性について述べた．テスト自動生成システムの実現に向けて，継続して TTE を開発する．RTOS の API テストは，RTOS を新しいターゲットにポーティングした際，ターゲット依存部が正しく実装されていることを確認する上で，必ず実施するべきテストである．つまり複数回実施される可能性の高いテストであるので，テストを自動化する対象に適している．本論文で提唱した手法，システムが RTOS のテストスイートとして標準的に利用されるようになることを願っている．

謝辞　テストポリシーが定まらず日々テストシートとテストシナリオデータの修正という非生産的な作業が続いていた時に，本手法の検討に参加し，さらに論文執筆にご協力下さった NCES 研究員の皆様に謹んで感謝の意を表する．

参考文献

1)　名古屋大学大学院情報科学研究科附属組込みシステム研究センター http://www.nces.is.nagoya-u.ac.jp/
2)　TOPPERS プロジェクト http://www.toppers.jp/
3)　坂村健監修，高田広章編，“μITRON4.0 仕様 Ver.4.02.00” トロン協会, 2004
4)　TOPPERS 新世代カーネル統合仕様書 http://www.toppers.jp/docs/tech/ngki_spec-110.pdf
5)　OJL による最先端技術適応能力を持つ IT 人材育成拠点の形成, http://www.ocean.is.nagoya-u.ac.jp/
6)　小林隆志，沢田篤史，山本晋一郎，野呂昌満，阿草清滋:On the Job Learning: 産学連携による新しいソフトウェア工学教育手法,電子情報通信学会信学技報 SS2009-28, (Vol.109, No.170, pp.95-100), 2009