組込みコンポーネントシステム TECSの概要

TOPPERSプロジェクトコンポーネント仕様WG主査
オークマ株式会社 FA システム本部
大山　博司

TECSとは

- TOPPERS Embedded Component System
  - TOPPERS プロジェクトコンポーネント仕様WGにより開発された、
  - 組込みシステムに適したコンポーネントシステム

コンポーネント仕様WGのこれまでのあゆみ

2004年　1月　コンポーネント仕様 WG 開始

2006年　3月　TECSと命名（方向性がほぼ固まる）

2006年10月　セキュリティコンポーネントIPA未踏に採択

2007年12月　組込みZINE連載開始(現CodeZine)

2008年　6月　会員向けリリース開始

2008年11月　キャッツとヴィッツがサポートを表明

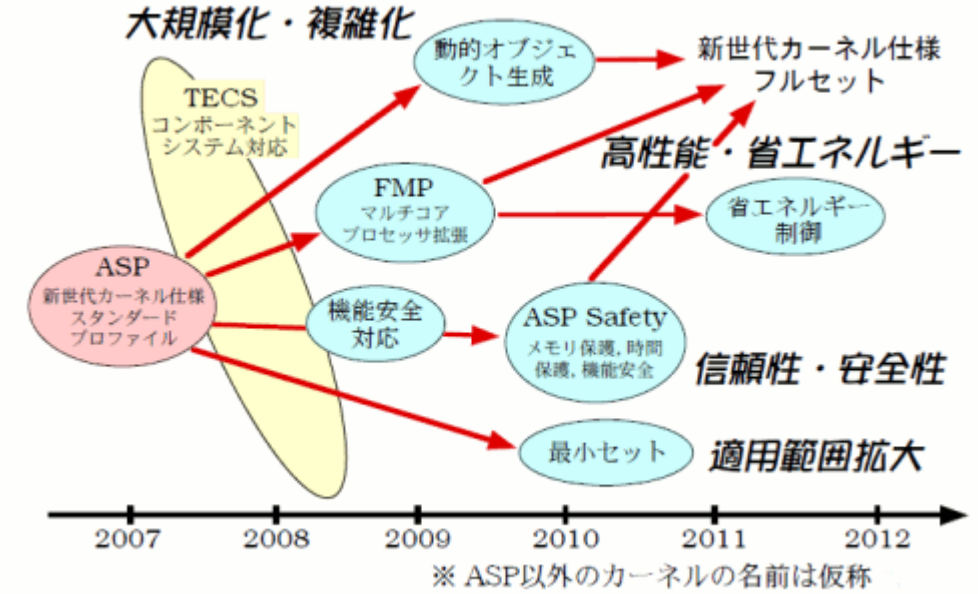
2009年　5月　一般向けリリース開始

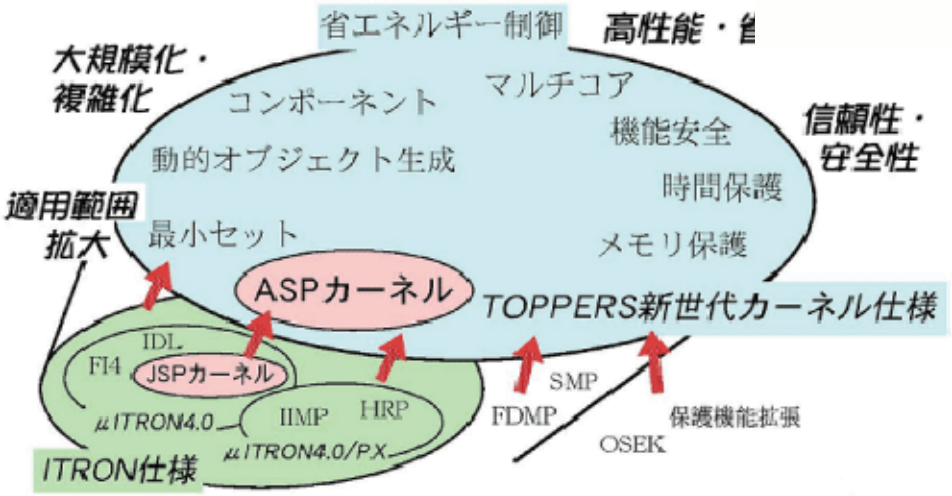
組込みプレスに記事を掲載

TOPPERS

TECSの位置づけ(1)
TOPPERSプロジェクトでの位置づけ

(1)TECS対応





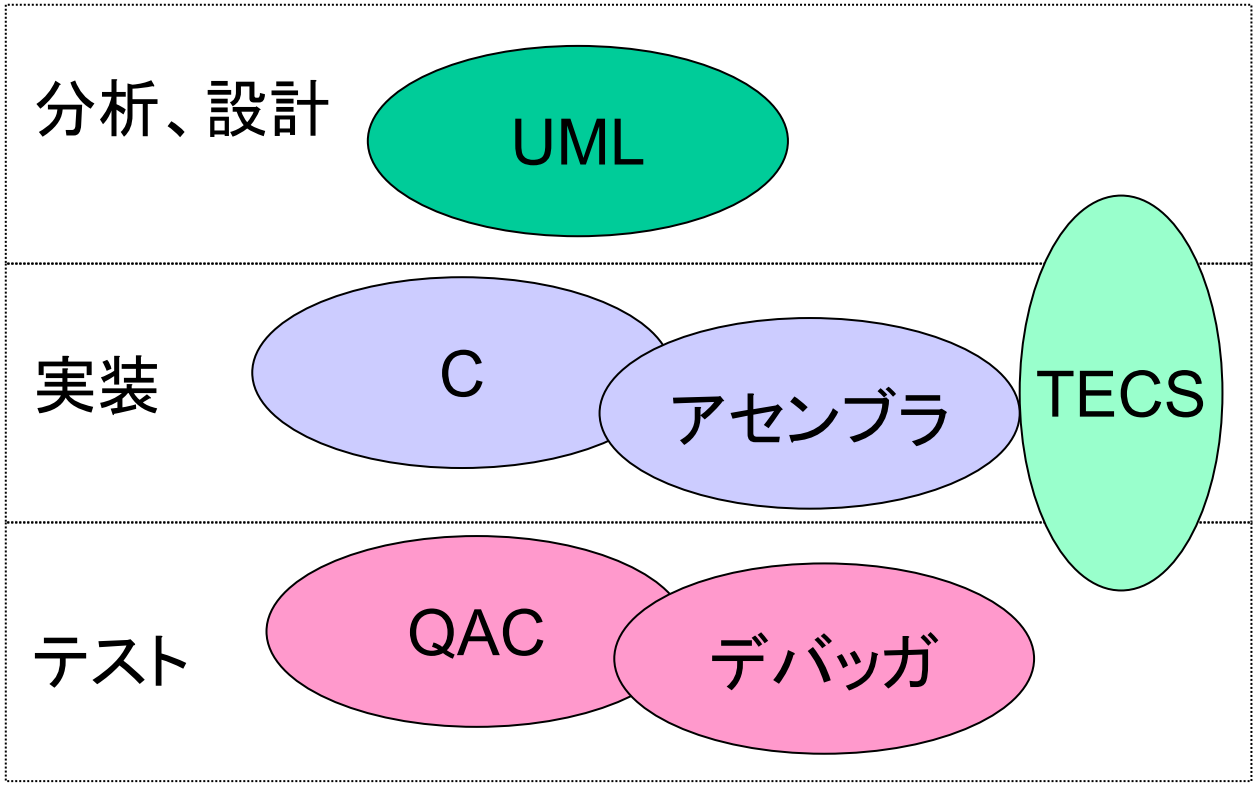
(2)新世代カーネル仕様

TECSの位置づけ(1)
TOPPERSプロジェクトでの位置づけ

- ソフトウェア部品、ミドルウェア提供の仕組みとして、各カーネルに共通した基盤
  - 移植性の向上
    - カーネルオブジェクトのコンポーネント化
  - プロセッサ間通信機構として

TECSの位置づけ(2)
ツールチェーンにおける位置づけ

- TECSがカバーできる設計領域の範囲



※Cやアセンブラが実装のほとんどすべてをカバーできるのに対し、他はすべてをカバーするものではない

組込みコンポーネントとは何か？

- 期待感はあるが、実のところコンポーネントとは何か、よくわからない
- 人によって定義は（感覚的なものを含めて）まちまち
  - コンポーネントをどのように定義づけるか、難しい課題
- 期待する役割もさまざま
  - 実際に使われる場面を想定し、「使える」仕様にすることが重要
- 既に非組込み系では、かなり普及している
  - 10数年ほど前から各種のソフトウェアコンポーネントフレームワークが提案されていて、広く普及している
    - CORBA　CCM、JavaBeans、MS COM
    - UMLのコンポーネント図

TECSでできること

- ソフトウェア構造の見える化
  - 大規模化するソフトウェアの見通しをよくする
  - 開発の分担を容易化
- インタフェースの見える化
  - 人と人とのインタフェースミスを低減
  - 再利用の促進
  - インタフェース標準化をしやすくし部品流通を促進
- 分散フレームワーク(RPC)
  - プロセッサ間の機能呼び出しを容易化
- その他
  - セキュリティ機能
  - 検証の支援などなど

TECS 仕様の設計方針(1)

- 静的なコンポーネント生成と結合
  - 柔軟な組上げと最適化によるオーバーヘッドの最小化
- すべてのものをコンポーネントとして扱う
  - アロケータ、RPCチャネルなどを隠さない
- インタフェース定義の曖昧さをなくす
  - 型、入出力方向、配列要素数
- オブジェクト指向技術を直接的には取り入れない
  - C言語を基本とする(想定する利用者の知識)

TECS 仕様の設計方針(2)

- プラグインによるフレームワーク・フレームワークの実現
  - 分散フレームワーク、セキュリティフレームワークなどフレームワークを構築するフレームワークに使える
- ソースコードからビルド
  - 種々の最適化を可能にする
  - バイナリ供給は別途考慮
- ROM/RAM 支援
- その他
  - 動作コンテキストの指定

TECS仕様の構成

- TECS コンポーネントモデル
- TECS コンポーネント図
- TECS コンポーネント記述言語(CDL)
- TECS コンポーネント実装モデル

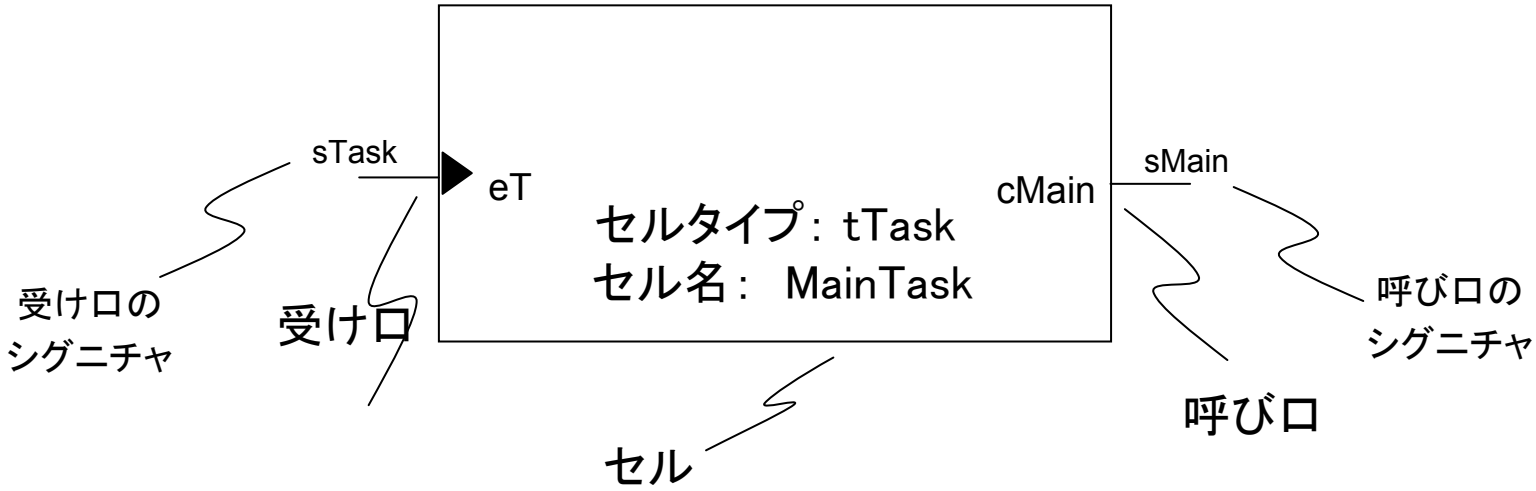
- WGでは仕様の確認と早期の普及のため以下を実施
  - 参照実装
  - 実例

参照実装

- TECSジェネレータ¶ tecsgenの実装
- TECSジェネレータのプラグインの実装
  - トレースプラグイン
  - RPC プラグイン（分散フレームワークの実現）
  - （セキュリティプラグイン）…IPA 未踏ソフトウェア開発事業採択

¶単にジェネレータと書いた場合、インタフェースジェネレータのことを指します

TECSコンポーネントモデル、コンポーネント図



セル（コンポーネント インスタンス）は

- 呼び口、受け口を持つ
- （図示しないが） 属性、変数を持つ
- セルタイプに属する（セルタイプコードとして振舞いを記述）
- 名前が付けられる
- （イベント、メッセージを扱うようなものはない.時間制約等もない）

セルタイプ　*部品の型式のコンポーネント記述*

- セルタイプは CDL で以下のように表現される
  - セルタイプの名前には接頭文字 t を付加する

```
celltype tTask {
  entry sTask    eT;      /* タスク受け口 */
  entry siTask   eiT;     /* 非タスクコンテキスト用タスク受け口 */

  call  sMain    cMain;   /* メイン関数呼び口 */

  attribute {
    ID                    id = C_EXP( "TASKID_$id$" );
    [omit]ATR             tskatr;
    [omit]PRI             itskpri;
    [omit]SIZE            stksize = 4096;
  };
  …
};
```

- この例では var は存在しない
- C_EXPはジェネレータが解釈しないで字面だけを扱うことの指定
- omitはCB/INIBに含めないことの指定

TOPPERS

シグニチャ *部品間のインタフェースのコンポーネント記述*

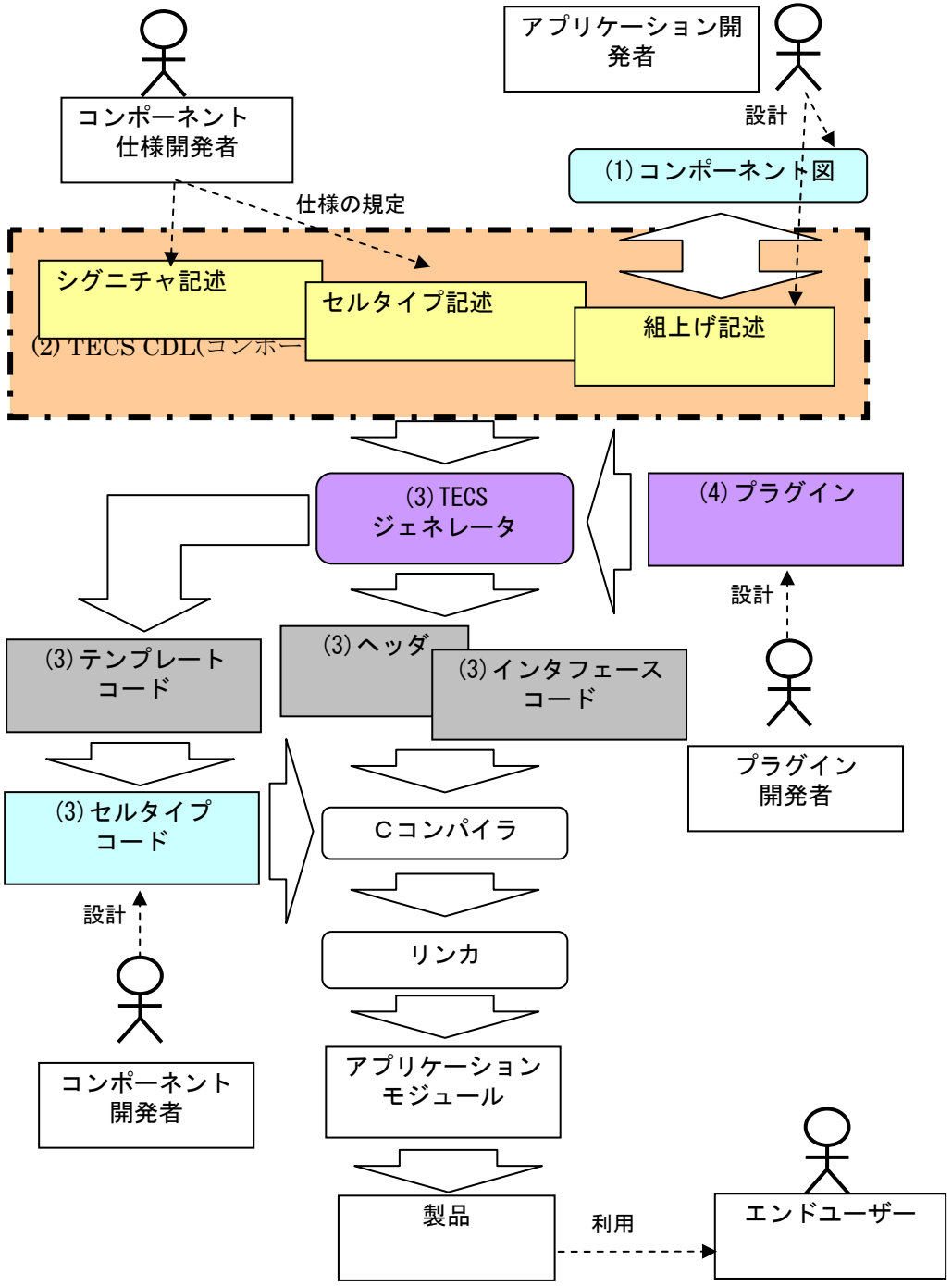
- 関数ヘッダの集合
  - 引数の入出力方向の指定 in, out, inout, send, receive
  - 引数のポインタは配列か、非配列か．配列ならばサイズを指定
    - ハンドリングされるデータ構造を明確化

- シグニチャの CDL 表現
  - シグニチャの名前には接頭文字 s を付加

```
signature sFile {
  ER open( [in,string]char_t *name, [in]int16_t mode );
  ER close(void);
  ER read( [out,size_is(len),count_is(*count)]int8_t *buf,
                [in]int32_t len, [out]int32_t *count );
  ER write( [in,size_is(len)]int8_t *buf,
                [in]int32_t len, [out]int32_t *wrote_len );
  ER seek( [in]int32_t offset );
};
```

  - []内を取り除くと、正当なC言語の文法になる

開発の流れ

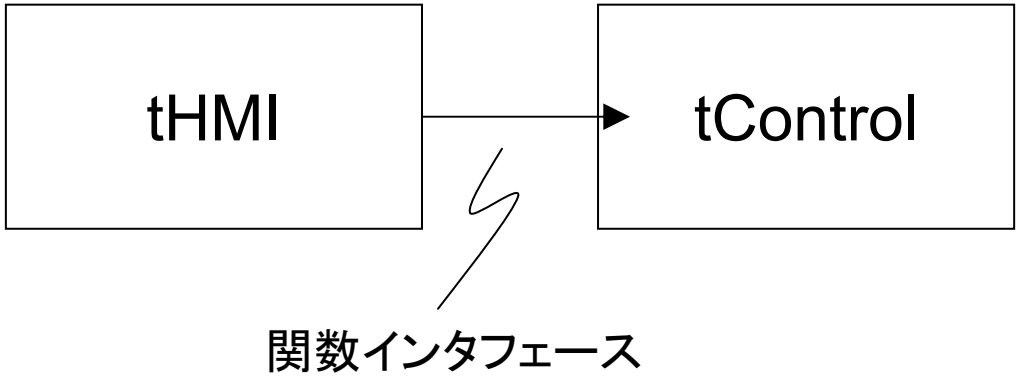

アプリケーション開発者
コンポーネント仕様開発者
設計
(1) コンポーネント図
仕様の規定
シグニチャ記述
セルタイプ記述
組上げ記述
(2) TECS CDL(コンポー
(3) TECS ジェネレータ
(4) プラグイン
設計
(3) テンプレートコード
(3) ヘッダ
(3) インタフェースコード
プラグイン開発者
(3) セルタイプコード
Cコンパイラ
設計
リンカ
コンポーネント開発者
アプリケーションモジュール
製品
利用
エンドユーザー

TECSの使い方の例

I. 大規模化、複雑化対応
   1. 構造を作りたい、見える化したい
   2. 関数インタフェースを正確に定義したい
   3. 一部を取替え可能としたい
II. 部品流通
   4. 部品を提供したい
   5. 部品を利用したい
III. 分散フレームワーク、テスト支援
   6. 密結合マルチプロセッサでRPCしたい
   7. 疎結合マルチプロセッサのRPCしたい
   8. トレースを使ってテストを自動化
IV. その他
   9. C++ の柔軟性をローコストで使いたい
   10. C++の柔軟性を持つROM化コード

1.構造を作りたい、見える化したい

- ソフトウェアに基本構造を与える
  - 大規模化、複雑化するソフトウェアの構造を可視化
  - 関数インタフェースにより独立性の高いモジュールに分離



2.関数インタフェースを正確に定義したい

```
signature sSig {
    ER open( [in,string]char_t *name);
    ER read(
        [out,size_is(sz),count_is(*len)]int8_t *buf,
        [in]int32_t sz, [out]int32_t *len);
    ER write(
        [in,size_is(*len)]int8_t  *buf,
        [inout]int32_t  *len);
    ER close();
};
```

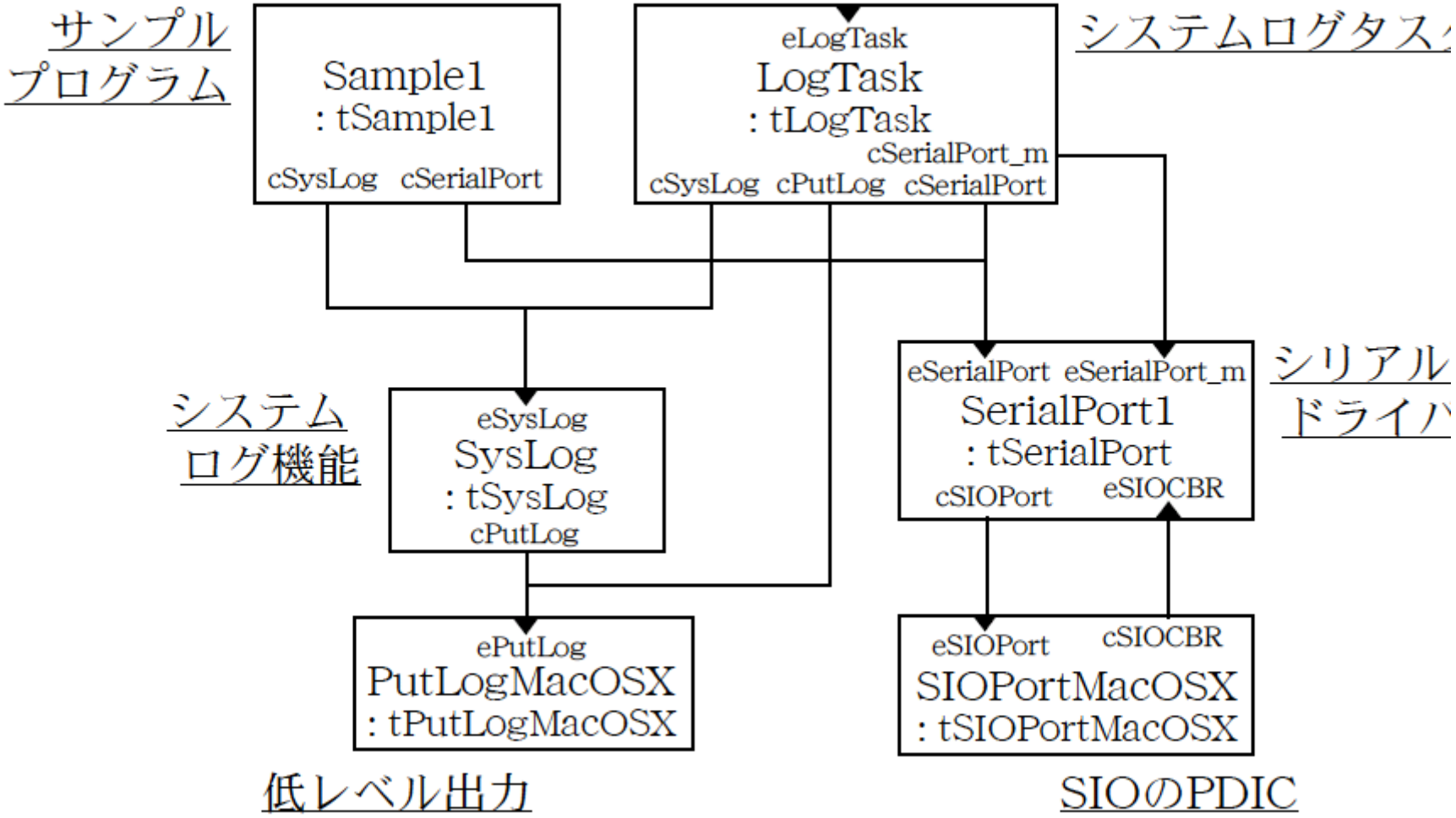
- C 言語の曖昧性をTECSがカバー
  - 曖昧さがなければ、もっと便利にできる
    - 曖昧さがとてもうれしいときもある

3.一部を取替え可能としたい

- プロダクトライン的
  - 組換えによるバリエーション展開
- TOPPERS/ASPのログタスク周り
  - 実装により変更したい

取替え可能な実装の例

ASPカーネルの周辺ドライバ等のTECS図

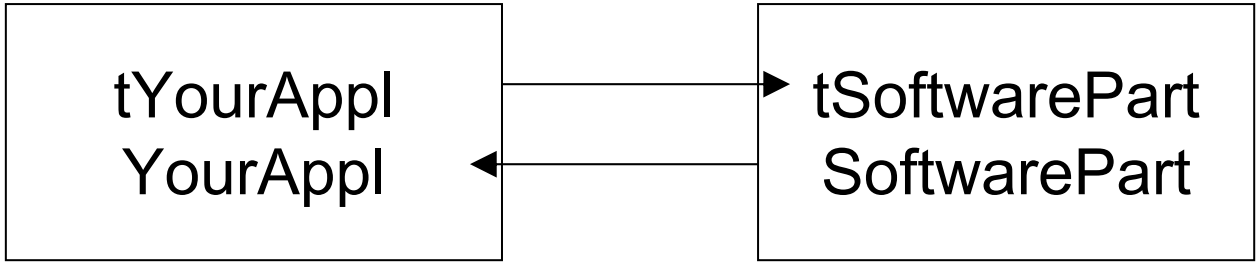


4.部品を提供したい

- 部品提供はTECSの重要なポイントの一つ
  - TECSの特性は組込み用の部品提供に最適
- シグニチャが提案されて共通化されることを期待
  - 標準化の進まない組込み向きAPIを提案しやすくする
- テスト自動化支援機能により期待した使われ方になっているかの検証ができる
  - TECS仕様で部品提供することのメリット

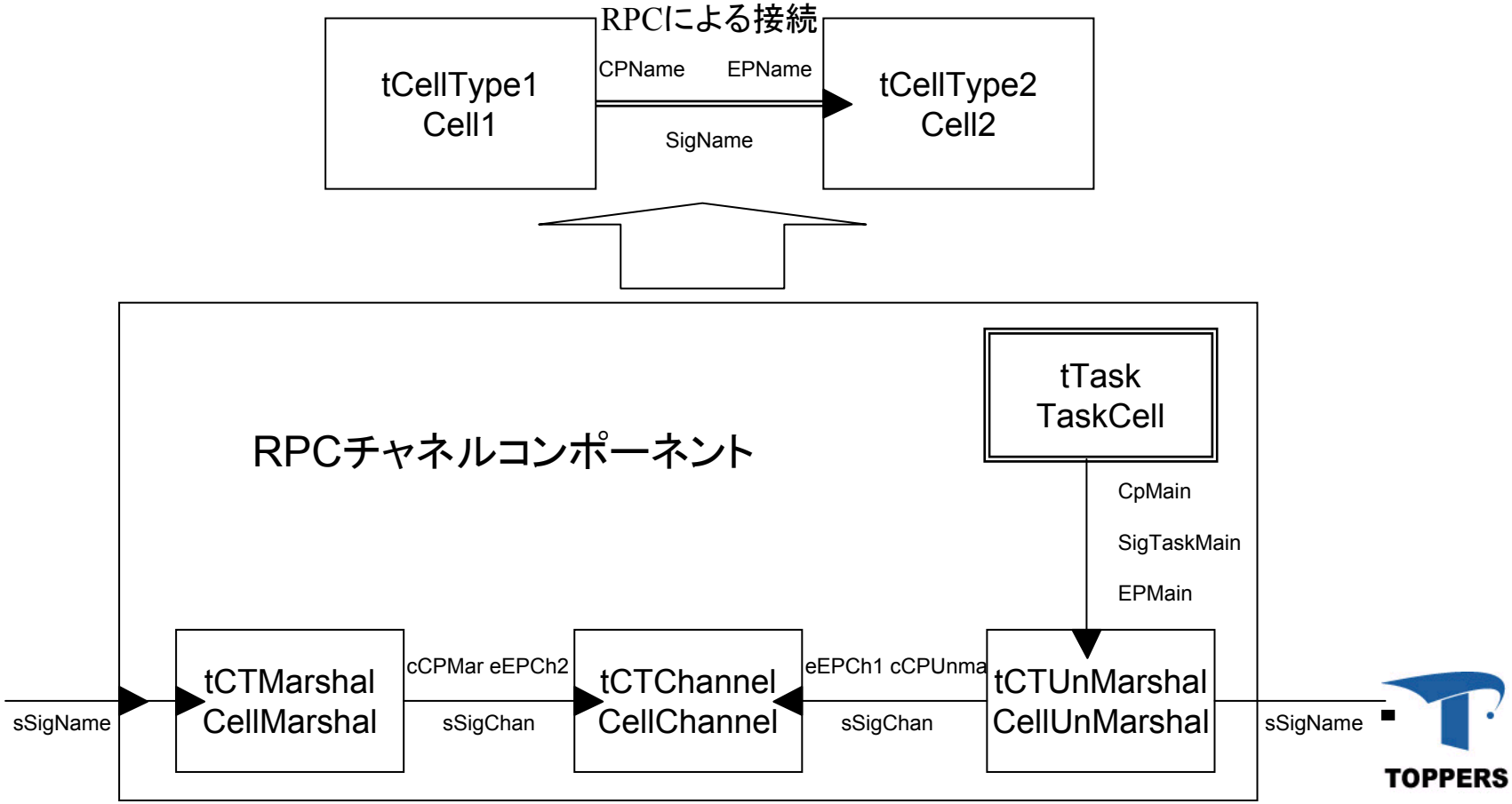
5.部品を利用したい

- 部品を利用したいけどTECS対応しないといけないの?
  - 部品だけを導入することは、容易に可能
  - 部品以外の部分を、(仮想の)アプリケーションコポーネントとして扱う
  - することは、以下のことだけ
    - tYourAppl_tecsgen.h を include
    - 呼び口関数を使って呼出す
    - 部品が必要とする受け口関数を用意してやる



6.密結合マルチプロセッサでRPCしたい

- トランスペアレントRPCチャネル
  - 単一プロセッサのタスク間通信
  - 対称マルチプロセッサのタスク間通信



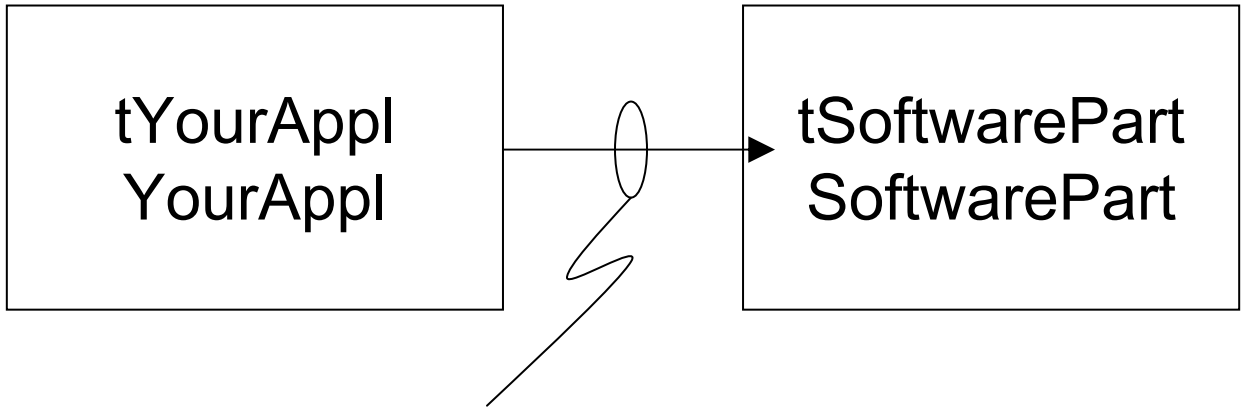
TOPPERS

7.疎結合マルチプロセッサのRPCしたい

- オペイクRPCチャネル
  - 疎結合マルチプロセッサ
    - 非対称マルチプロセッサ
    - 小さな共有メモリ
    - 通信チャネル

8.トレースを使ってテストを自動化

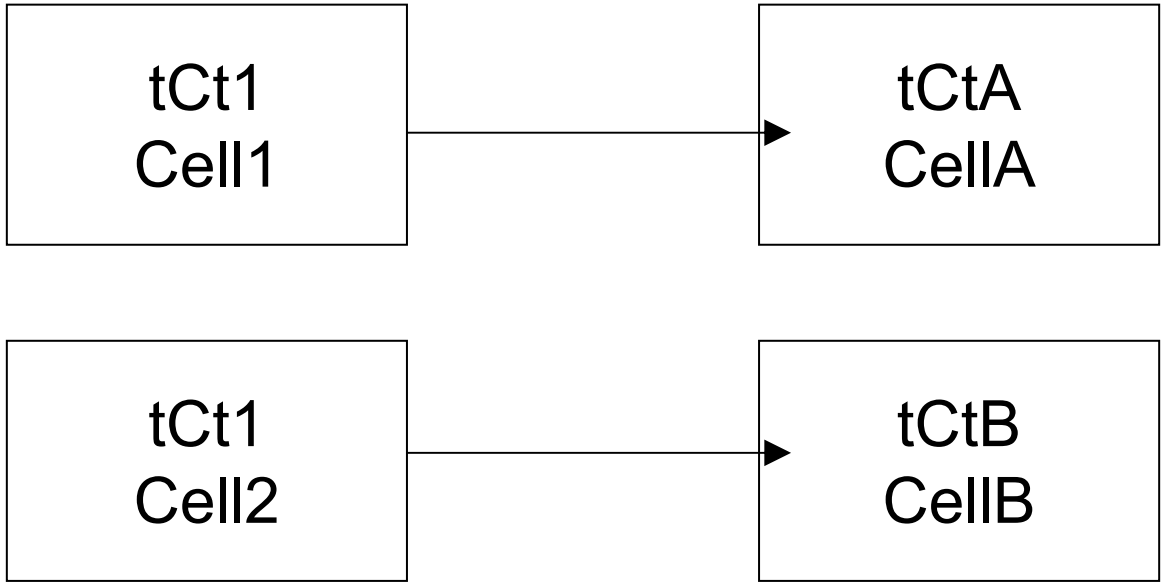
- ソフトウェアプローブ



- リグレッションテストの自動化
- 部品の動作が思わしくないときの情報収集手段

9.C++ の柔軟性をローコストで使いたい

- TECSコンポーネント図で以下のようなものを実現しようとするとC++ではvirtual関数実装が必要
- TECSでも同様
  - しかし、上半分だけでよい場合には、自動的にvirtualではなくなる(最適化)



10.C++の柔軟性を持つROM化コード

- インスタンスをROM化したい
- C++の場合　×

```
class ROMable{
    const int32_t ROval;
    int32_t       RWval;
}
```

  - ROval はROM に配置されることはなく動的に初期化
- TECSの場合

```
celltype tCt {
  attribute { int32_t  ROval; };    // ROM配置
  var{ int32_t  RWval = ROval; };       // RAM配置
};
```

  - TECSではポインターつで両方扱える（つなぐポインタは隠される）
    - ROMオンリ、RAMオンリの場合、自動でポインタが繋がれる
  - 起動を早くするのに有効と考える

ダウンロード先

- ホームページURL
  - http://www.toppers.jp/tecs.html

- パッケージ内容
  - TECS参照実装
  - TECS仕様書
  - ASP+TECS
  - skyeye 試験環境

参考文献

- 組込みプレス Vol.15
  - 「TECSを使ってみよう」
- TECS開発ブログ
  - http://tecs22022.blog85.fc2.com/
    - 「超簡易パッケージ」など
- CODE ZINE (少し古いところあり)
  - 「組込みコンポーネントシステムTECS 」
  - http://codezine.jp/article/corner/284