TECS RPC

講師：大山　博司

TOPPERS　TECS　WG主査

オークマ株式会社FAシステム本部

目次

- TECS の概要
- RPC の概要
- TECS RPC を構成するもの
- TECS RPC をビルドし、動かしてみる
- TECS ジェネレータ V1.1 の新機能

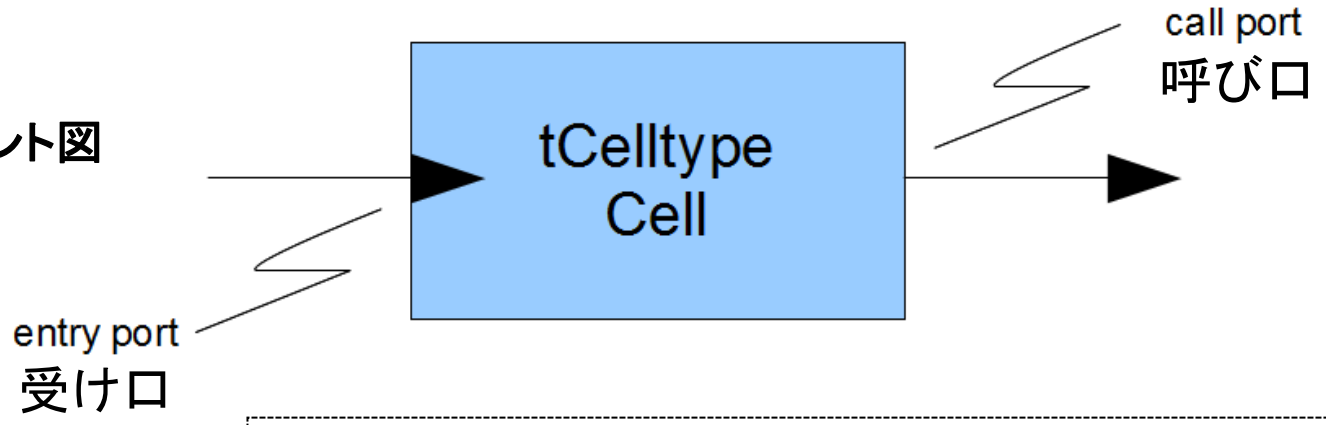
開発の動機

- 複雑な組込みシステムに適したコンポーネントシステムが必要である
  - コンポーネントシステムにより複数の問題の同時解決が期待できる
  - ソフトウェアの見える化
  - ソフトウェア部品の流通
  - RPCフレームワーク
- 汎用OS用のコンポーネントシステムは普及しているが、組込みシステムようのものは普及していない
  - しかし、汎用OS用の組込みシステムでは過剰である

基本コンポーネントモデル

- TECSコンポーネントは最小限の機能を持つ
  - 関数呼び出しを行うポート（口）のみを持つ

TECS コンポーネント図



TECS コンポーネントモデル (基本モデル)

cell **(セル)**はコンポーネントのインスタンス
cell は***attribute***.**(属性)**と呼ばれる定数を持つ
cell は ***var (internal variable)***(**変数**)と呼ばれる変数を持つ
cell は***entry port*** **(受け口)** を通して機能を提供する
cell は***call port***. **(呼び口)** を通して機能を利用する
　　関数の集合を ***signature*** **(シグニチャ)**と呼ぶ
cell は ***celltype***.**(セルタイプ)**に属する
　　celltype は属するcellの特性を決定する

鍵となる特性

- コンポーネント(cell) は静的に配置(生成)され、結合される
  - 静的だが柔軟である
    - リンケージの決定による（次のスライド）
- 小さなオーバーヘッド
  - ソースコードからビルドする
    - コンポーネントはソースコードでの流通となる（バイナリ形式は別途）
  - オーバーヘッドの最小化は最適化による
- 振舞いはC言語により記述
  - オブジェクト指向プログラミング（言語）は不要

リンケージの決定

- 通常のC言語の場合
  - リンケージはプログラムの記述時に決定される
    - 呼び元の関数は、どの関数を呼び出すかを決定する

- TECSコンポーネントの場合
  - リンケージは、コンポーネントを用いてソフトウェアを組上げる段階で決定される
    - TECS コンポーネント記述言語(TECS CDL)により組上げを記述する
      - リンケージはC言語のプログラム記述時には決定されない
      - TECSジェネレータの生成するマクロによりリンケージが変えられる

リンケージの決定 (2)

- もしコンポーネント間の関係が単純ならば、呼び口関数は、直接受け口関数に結合される
- もしコンポーネント間の関係が複雑ならば、vtable(関数テーブル)が使用され、実行時に結合が決定される

TECS 仕様

- 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 (TECS CDL)
- TECS コンポーネント実装モデル

- これらは TECS WG により策定作業が行われている

TECS の実装

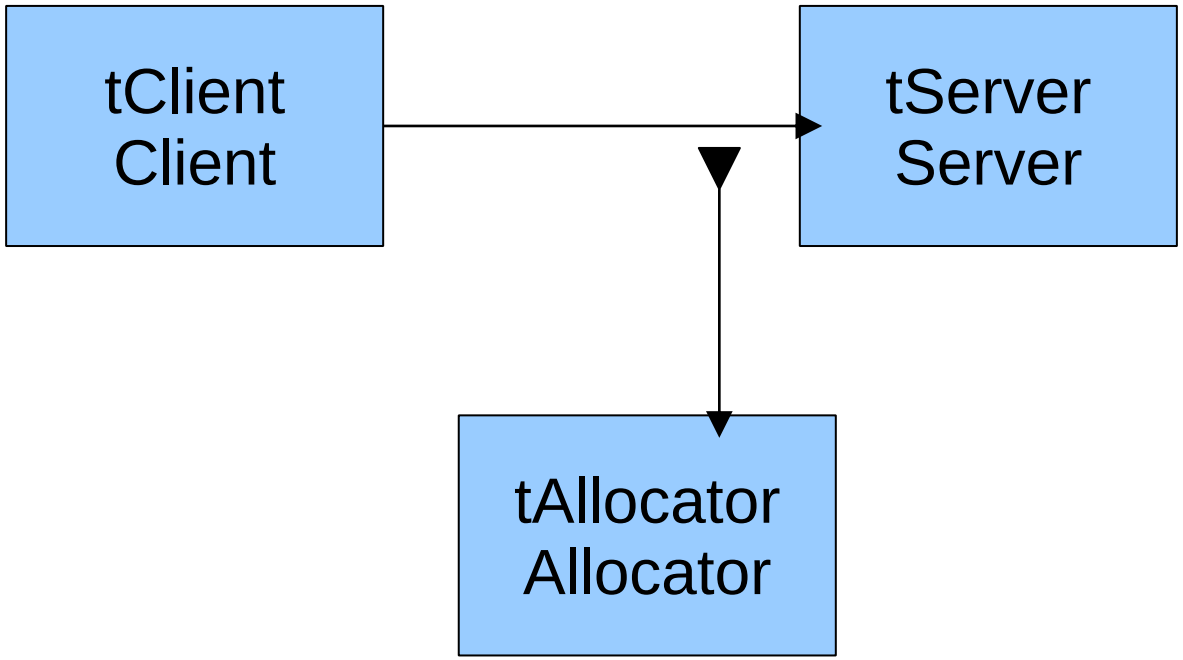
- TECS ジェネレータ
  - ***TECS CDL*** からインタフェースコードとcell を定義するコード(C言語)を生成する
  - プラグインを適用する
    - RPC プラグイン, Trace プラグイン, etc
- TOPPERS/ASP+TECS
  - RTOS のカーネルオブジェクトを TECS のセルタイプとして定義したもの
  - これらの実装は TECS WG により行われている

その他の鍵となる特性

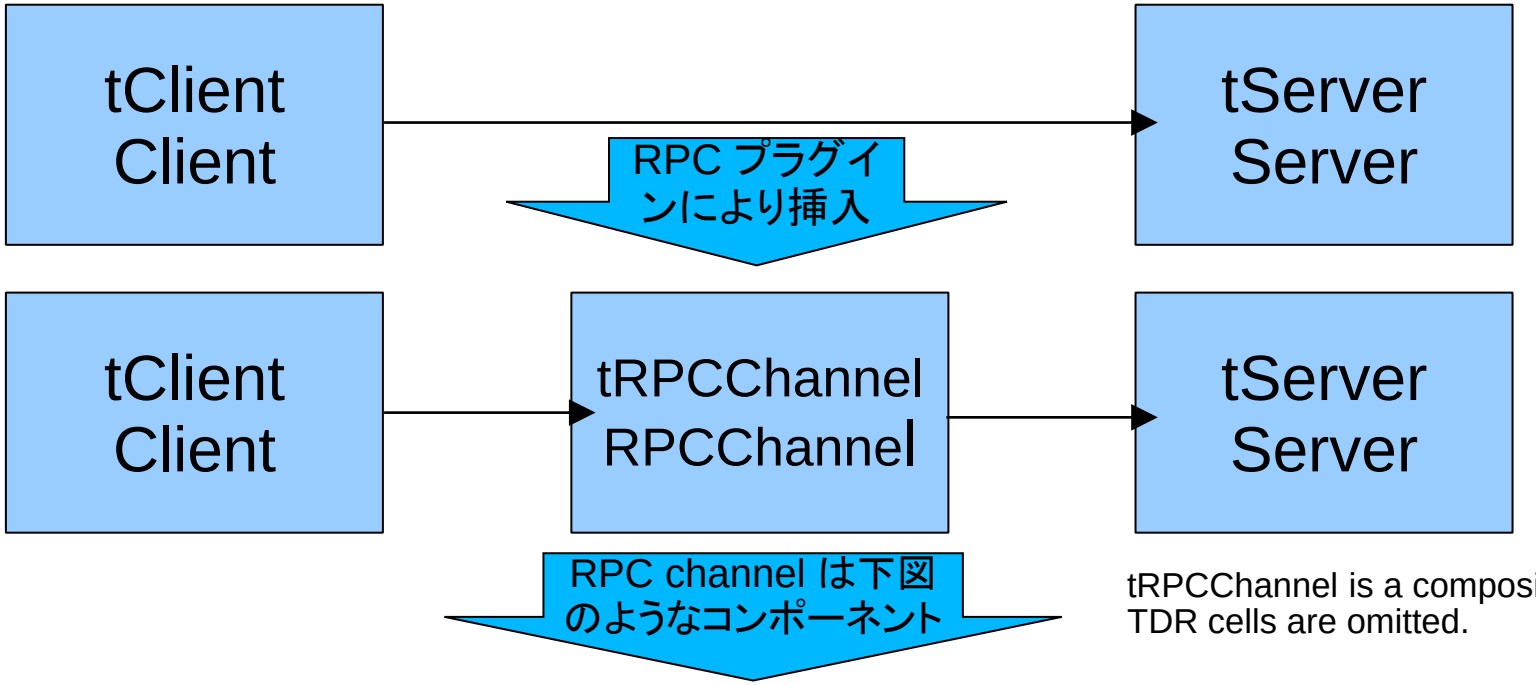
- コンポーネントシステムの下に隠さない
  - メモリアロケーター
  - RPC チャンネル
  - RTOS カーネルオブジェクト
- 小さな組込みシステムにも向く
  - Inline関数実装
  - ROM化コード

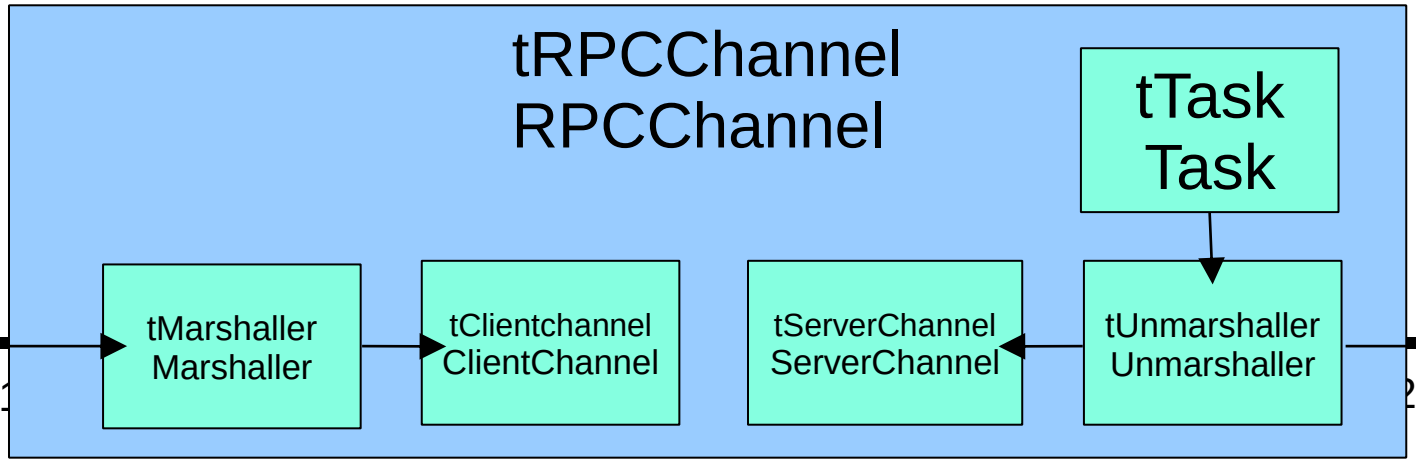
メモリアロケータ

- send：呼び元の cell がアロケートし 呼び先の cell が解放
- receive：呼び先の cell がアロケートし 呼び元の cell が解放



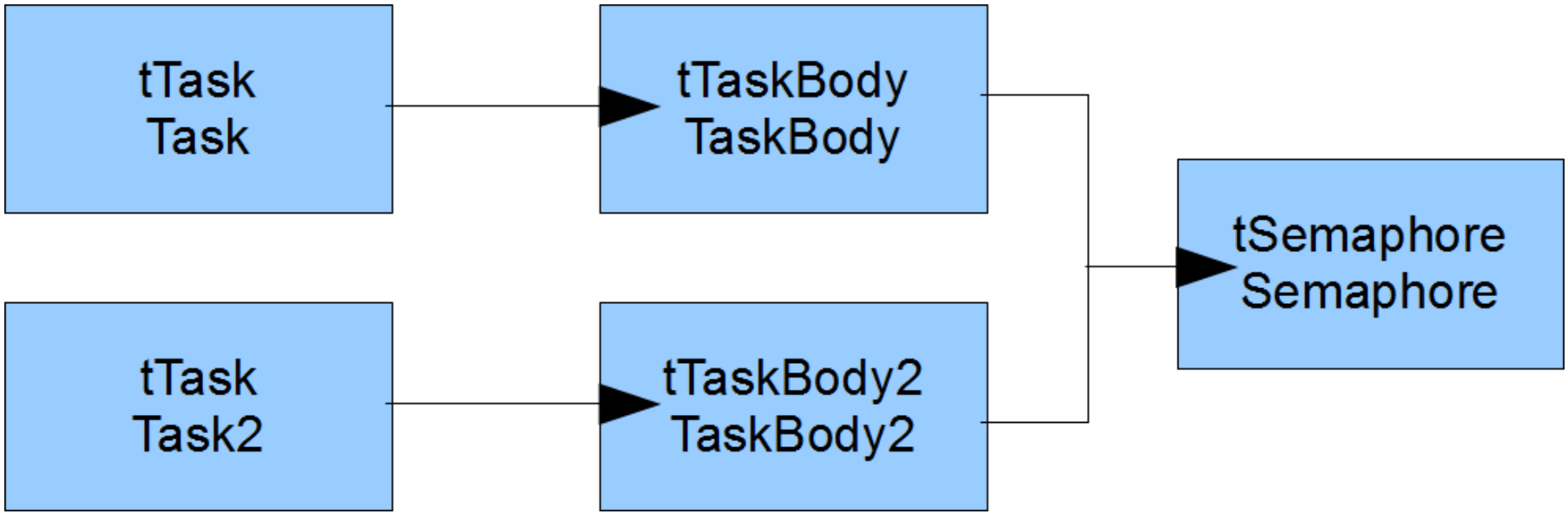
RPC: Remote Procedure Call





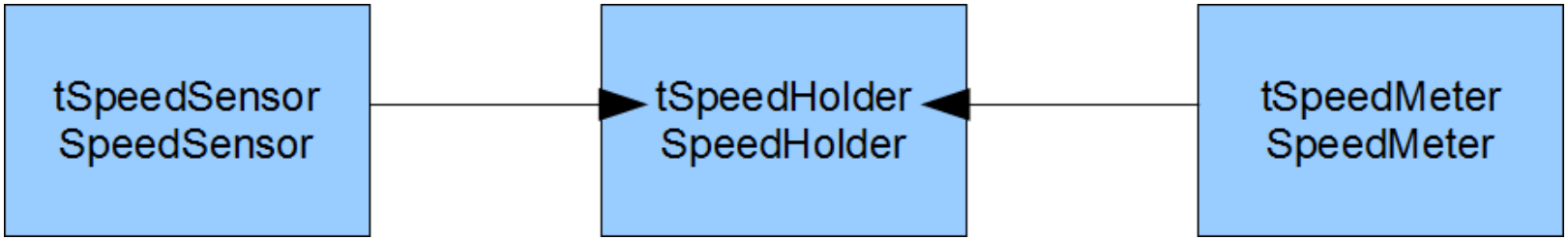
カーネルオブジェクト

- オーバーヘッドが小さいため、カーネルオブジェクトのような小さな粒度のものもコンポーネントとして扱える
  - セマフォがどのタスクに共有されているか可視化されている



メッセージ

- TECS コンポーネントにメッセージを扱うポートはない
- 以下のようにコンポーネントを割り込ませることで実現する
  - オーバーヘッドが小さいためコンポーネントとして扱える



目次

1. TECS の概要
2. RPC の概要
3. TECS RPC を構成するもの
4. TECS RPC をビルドし、動かしてみる
5. TECS ジェネレータ V1.1 の新機能

RPC とは

- Remote Procedure Call
  - 遠隔手続き呼出し
  - 分散処理を実現するための一つの手法
- 普及しているもの
  - ONC RPC (Sun RPC)
    - NFS の実装に使用
  - DCE RPC
    - Microsoft 社 DCOM (DCOM) のベースとなっている
- 類似のもの
  - XML RPC
  - SOAP … XML ベースの RPC の仕組み

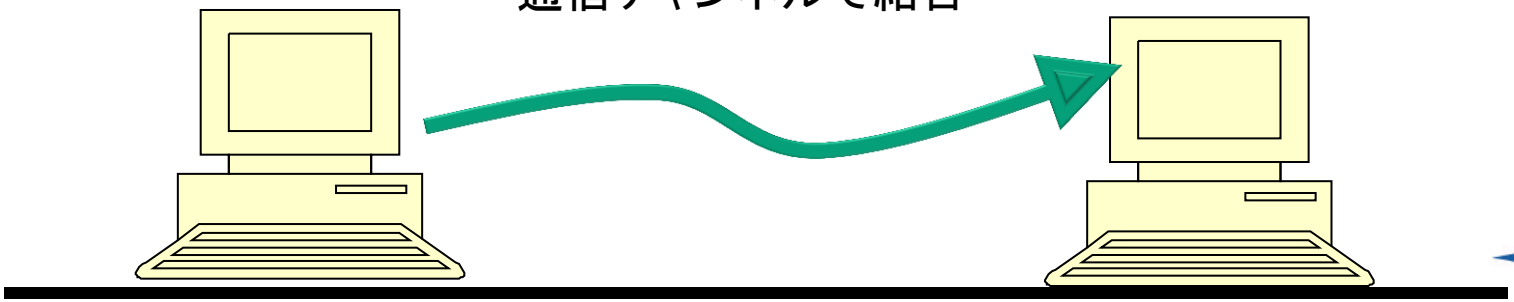
RPCのマジック

PE1で呼出されたfunc_1がPE2で呼出され実行される

PE1
```
main()
{
   ER er; short a; long b;
   a=0; b=100;
   er= func_1(a, b );
}
```

PE2
```
ER func_1( short a, long b )
{
   do_func(a,b);
   return E_OK;
}
```

通信チャンネルで結合



TOPPERS

RPCを使用しない通信(1)

- RPCを使用しない場合の設計手順
  - 以下の1)～3)を必要な機能の数だけ繰返す

１）パケットの設計

- 機能1を要求するパケット

```
offset
       +--------------------+
 0     |     ID_func_1      |
       +--------------------+
 2     |     short a        |
       +--------------------+
 4     |     long b         |
       +--------------------+
```

- 機能1の要求に対する応答パケットの構造

```
       +--------------------+
 0     |     ER             |
       +--------------------+
```

RPCを使用しない通信(2)

２）機能1の要求パケットを送出し、応答を受信するプログラムの設計

```
ER func_1(short a, long b )
{
    ER er;

    send_func_id( ID_func_1 );
    send_short( a );
    send_long( b );

    er = recv_ER();
    return er;
}
```

RPCを使用しない通信(3)

３）機能1の要求パケットを受信し、関数を呼出し、応答を返すプログラムの設計

```
void main_loop()
{
   ID func_id;
   while(){
      func_id = recv_func_id();
      switch( func_id ){
        case ID_func_1:
          func_1_driver();
          break;
       }
   }
}
/* 実際の func_1 を呼出すプログラム */
void func_1_driver()
{
   short a;   long b;   ER er;
  a = recv_short();
   b = recv_long();
   er = func_1( a, b );   /* 実際の func_1 を呼出す */
   send_er( er );
}
```

RPC を使用することのメリット

【従来】

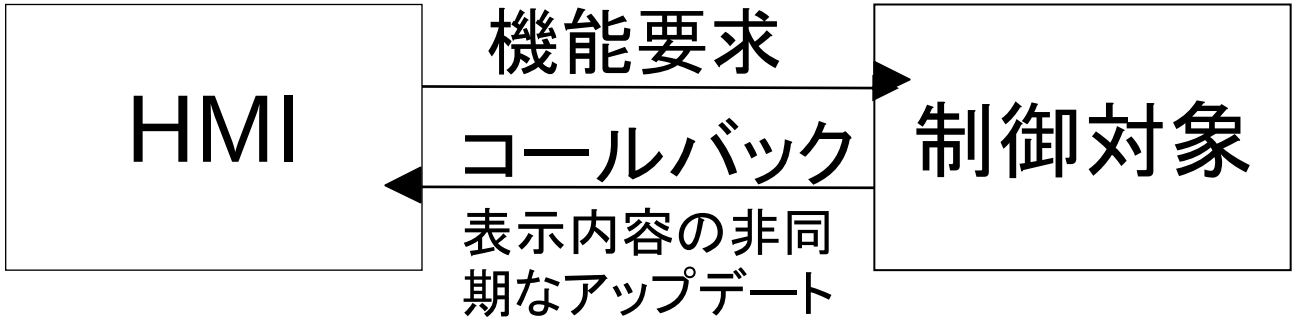
- 共有メモリを持つマルチプロセッサの場合
  - 変数結合になりがち
- 機械的に記述できる通信プロトコルの送受信プログラムを人手でメンテナンスする場合
  - 一つの関数で複数の機能から選択実行できるように設計しがち（スタンプ結合、制御結合）

【RPCを使うと】

- 関数結合になる
  - インタフェースが明確になる

使用例1 HMI と制御対象の分離

- HMI (ユーザインタフェース)と制御対象を分離
  - HMI を制御対象とは独立に更新できる
  - HMI を http サーバーで置き換えるなど、従来にはない拡張が容易になる

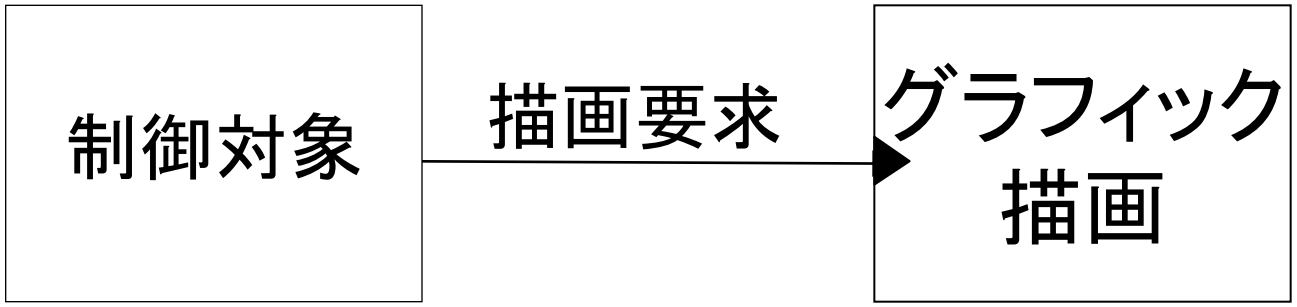



```
signature sCommand {
   ER  play( void );
   ER  stop( void );
   ER  forward( void );
   ER  rewind( void );
};
```

```
signature sNotify {
   ER  reachEnd( void );
   ER  mediaError( [in]int16_t errno );
};
```

使用例2 グラフィック描画の分離

- グラフィックを別プロセッサで処理
  - 非同期呼び出しで効率低下を防ぐ




```
signature sGraphic {
   [oneway]void  DrawLine( [in]int16_t sx, [in]int16_t sy, … );
   [oneway]void  DrawCircle( [in]int16_t cx, [in]int16_t cy, … );
    …
};
```

目次

1. TECS の概要
2. RPC の概要
3. TECS RPC を構成するもの
4. TECS RPC をビルドし、動かしてみる
5. TECS ジェネレータ V1.1 の新機能

マーシャラ、アンマーシャラ

- シリアライザとも呼ばれる
- マーシャラ … 関数呼出しをパケット化
- アンマーシャラ … パケットを解釈して関数呼出し
- 自動生成可能
  - 関数定義に曖昧さがなければ、マーシャラ、アンマーシャラへは機械的に変換できる
    - C言語の関数インタフェースは、まったく曖昧
    - TECS CDL のインタフェース（signature) は曖昧さがない
  - CDLからマーシャラ、アンマーシャラを自動生成

    ER func_1([in]short a, [in]long b);
    - 引数のデータ構造、データの流れる向き、データ型のサイズなどが明確でなくてはならない

RPCジェネレータ

- RPCチャンネルを自動生成するプログラムを RPCジェネレータと呼ぶ
- マーシャラ、アンマーシャラの設計工数不要
  - メンテナンスがとても楽になる
    - 機能要求の追加、引数の追加など
    - 通信に関するデバッグが不要になる
      - 本当に不要！
      - しいて言うとインタフェース定義がクライアントとサーバで相違している場合に問題が発生する
  - インタフェース定義により通信内容が明確化
    - 製品改良する際に重要
  - 設計者によらず均質化
    - インタフェース定義
    - パケット化ルール
    - エラー処理

Opaque RPC と Transparent RPC

- Transparent RPC
  - メモリ空間を共有するタスク間で使用する
    - ポインタは、そのまま引き渡される
    - 役割としては、呼び出し元のスタックを呼び先のタスクのスタックにコピーして呼び出すだけ
    - プラグイン名
      - **RPCPlugin, SharedRPCPlugin**
- Opaque RPC
  - メモリ空間を共有しないタスク間で使用する
    - ポインタをそのまま渡しても意味がない
    - ポインタの指している先のデータをコピーして渡す
      - そのためにはデータ構造の曖昧さがあってはならない
    - プラグイン名
      - **OpaqueRPCPlugin, SharedOpaqueRPCPlugin**

TDR

- TDR … TECS Data Representation
  - エンディアンの相違やサイズの相違などを調整する働き
- tTDR … ストレート TDR
  - Transparent RPC のデフォルト
  - エンディアン調整等、何も行わない
- tNBOTDR … Network Byte Order TDR
  - Opaque RPC のデフォルト
  - ネットワークバイトオーダー(Big Endian)で送受信
    - ヘテロ環境でも問題なく動作

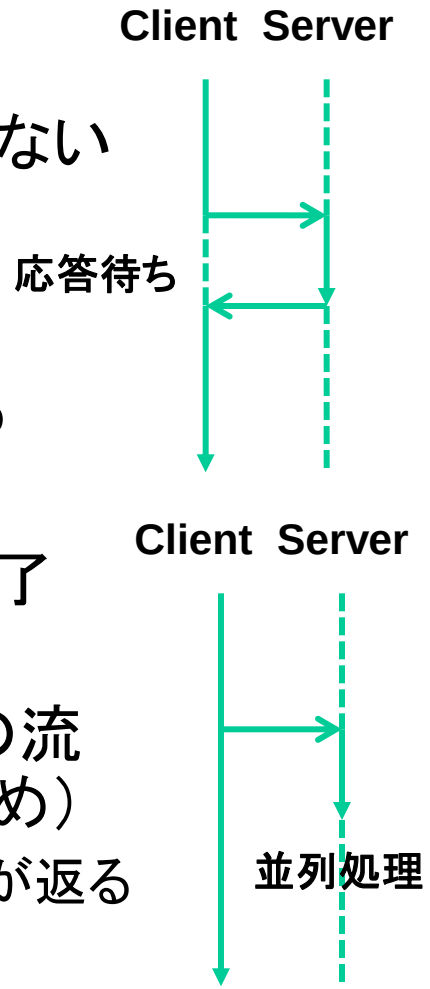
以下は未実装

- tSBOTDR … Sender Byte Order TDR
- tFBOTDR … Full Byte Order TDR

非同期呼出し

- 同期呼出し
  - 関数の処理が終わるまで呼出し元に制御が戻らない
  - 通常のC言語の関数呼出し
- 非同期呼出し
  - 関数の処理が終わる前に呼出し元に制御が戻る

    [oneway]ER func_1([in]short a, [in]long b);
  - 呼出し元では、パケット送出後、直ちに関数が終了
    - 並列処理となる
  - oneway は非同期を表すよりも一方向なデータの流れを表す（TECSでは直接呼出しも可能であるため）
    - 同期呼出し（＝往復呼出し）では、分散していると応答が返るまでに時間がかかる
  - ER はマーシャラで発生したエラーを返す



エラー回復モデル

- マーシャラ、アンマーシャラより下の層で発生するエラーへの対処
  - 簡単には、チャンネルをいったんクローズして、開きなおす
    - と同時に、決められた地点へ戻って再開する
      - もっとも単純には、最初に戻る
  - Cygwin 版のみテストコードを実装
    - 一方を途中で Ctrl-C で止めると、他方は待機状態に移る
- Transparent RPC は未対応
  - Transparent RPC では、通信チャンネルでエラーが発生するのは、ソフトまたはハード不具合と同等
  - PPAllocator あふれも含まれる

リージョン分割

- リージョンは、セルの配置を制御する働きがある
  - node, linkunit が指定されていると、独立したモジュールとしてビルドされることを表す
  - 異なるnode, linkunit 間の直接結合禁止

```
[linkunit]                    // 独立したリンク単位(node でもよい)
region rServer {
    cell tServerCelltype ServerCell{ };
}
[linkunit,
 to_through( rServer, OpaqueRPCPlugin, オプション)]
region rClient {
    cell tClientCelltype ClientCell{
       cCall = rServer::ServerCell.eService;
    };
}
```

PPAllocator

- サーバー（呼び先）側に配置する in, inout, out 引数のうち文字列や配列を記憶するための領域
  - send, receive 引数は、指定されたアロケータを使用
- プラグインオプション PPAllocatorSize で必要なサイズを指定する
  - size_is や string の最大値から割り出すことは可能だが、現在の実装では手動設定が必須
- 単純なアロケータ
  - アロケートした逆順にデアロケートする必要がある
  - 一気に全部をデアロケートする手段を持つ
  - 一回の関数呼び出しごとに、すべてデアロケート
- バッファオーバーラン攻撃を防ぐためサイズチェック

プラグインオプション (OpaqueRPCPlugin)

```
// チャンネルセルタイプ名
    "clientChannelCelltype= "¥tSocketClient"¥, “
    "serverChannelCelltype= "¥tSocketClient"¥, “
// チャンネル
    "clientChannelInitializer= !portNo=8931+$count$;
serverAddr=¥"127.0.0.1¥";!, "
    "serverChannelInitializer= !portNo=8931+$count$;!, “
// セマフォセルタイプ
    "clientSemaphoreInitializer= !count = 1; attribute =
C_EXP( ¥"TA_NULL¥" );! ,"
    "clientSemaphoreCelltype = tCelltype, “
    “noClientSemaphore = false, “
```

プラグインオプション (OpaqueRPCPlugin)

```
// TDR (TECS Data Representaion) セルタイプ
    "TDRCelltype=¥"tNBOTDR¥","
// サーバー側タスクセルタイプ
    "taskCelltype=tTask,"
    "taskPriority=11,“
    "StackSize=4096,“
// PPAllocator サイズ(引数の総和)
    "PPAllocatorSize=1024,“
// 代替アロケータ指定 (デフォルトは未指定)
"substituteAllocator=%Alloc.eAlloc=>CAlloc.eAlloc%, "
```

¥”, %, ! , ‘ で = 右辺の文字列を囲むことができる

プラグインオプション (OpaqueRPCPlugin)

```
// クライアント側チャンネルセル名
   "clientChannelCell='ClientChannel_$destination$', “
// サーバー側チャンネルセル名
   "serverChannelCell= !ServerChannel_$destination$!, “
// エラーハンドラー(セル名.呼び口; デフォルトは未指定)
    "clientErrorHandler =
'ClientRPCErrorHandler_$destination$.eHandler', “
    "serverErrorHandler =
'ServerRPCErrorHandler_$destination$.eHandler', “
```

$...$ は以下の置換ルールがある　(throughPlugin.rb 参照)

- $count$ は適用数 (>=0)
- $source$ … 呼び元のセル名
- $destination$ … 呼び先のセル名

TOPPERS

プラグインオプション (その他)

- Transparent RPC (RPCPlugin) のオプション
  “noClientSemaphore = true または false, “
  "semaphoreCelltype = tSemaphore, “
  "taskPriority = 17, ”
  "channelCelltype = tDataqueueOWChannel, "
  "TDRCelltype = tTDR, "
  “channelCellName = (tecsgen が指定), "
  “PPAllocatorSize = N, “ N: 整数
  （Dataqueueのサイズを調整する手段が不足）
- SharedRPCPlugin, SharedOpaqueRPCPlugin の引数は整理途中

目次

1. TECS の概要
2. RPC の概要
3. TECS RPC を構成するもの
4. TECS RPC をビルドし、動かしてみる
5. TECS ジェネレータ V1.1 の新機能

skyeye 版の試験実行

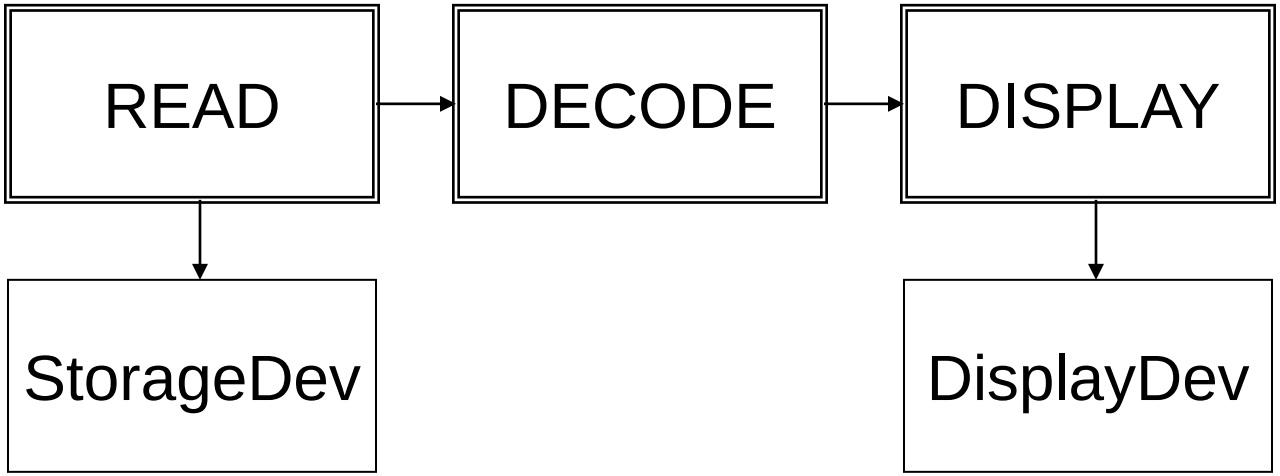
- skyeye はパッケージ同梱のものを使用すること
  - 複数の TAP に対応する拡張がなされている
- README-skyeyeWithTINET.pdf を参照して、正確に設定を行うこと
  - トラブル解決には TCP/IP に関する知識も必要
    - まずは ping
    - 次は ipconfig など
- 非常に時間がかかるので、じっくり待つこと
  - TransparentRPC の実行に比べて、極端に遅い
    - TCP/IP 実効に非常に時間がかかる

cygwin 版の試験実行

- 非常に動きが速い
- ループバックアドレス 127.1.0.1 で接続
  - 同じPC同志なら特別な設定不要
  - ファイアウォールのメッセージが出る(ことがある)
- 出力されるメッセージは、以下の3つ
  - アプリケーション tSimpleClient.c, tSimpleServer.c の出力するメッセージ
  - RPC_DEBUGで出力されるデバッグメッセージ
  - TracePlugin により出力されるメッセージ
- Ctrl-C で中断することで、エラー回復を試すことができる

画像処理への応用

読出し、デコード、表示を並列処理



- コンポーネント化によりハードウェア実現したものとソフトウェア実現したものの容易な入替えが期待できる
  - TECSのハードウェアコンポーネントとの統合は、未解決の課題

目次

1. TECS の概要
2. RPC の概要
3. TECS RPC を構成するもの
4. TECS RPC をビルドし、動かしてみる
5. TECS ジェネレータ V1.1 の新機能

TECS ジェネレータ V1.1の新機能

- エラーメッセージの日本語表示
  - 多言語対応（現時点では、日本語と英語のみ）
- セルの前方参照
  - セルのプロトタイプ宣言が不要
- 固定結合
  - 初期化のための結合の記述が不要
- リージョンによるモジュールの分割
- namespace による名前衝突の回避
- RPCプラグイン
  - エラー回復モデル、メモリリークしないコードの生成など製品レベル
- プラグインの拡充 (従来 throughプラグインのみ)
  - シグニチャプラグイン … RubyTECSBridgePlugin
  - セルタイププラグイン … ATK*Plugin
  - セルプラグイン … RepeatCellPlugin, RepeatJoinPlugin

TECS ジェネレータ V1.1の新機能

- セルの前方参照
  - TECS CDL は、前方参照できない（一部例外を除く）
    - typedef, const, struct
    - signature, celltype, composite, cell
  - cell の前方参照を可能とした
    - cell のプロトタイプ宣言が不要となった
    - プラグインで生成されるセルは、評価順によって、プロトタイプ宣言が必要

TECS ジェネレータ V1.1の新機能

- エラーメッセージの日本語表示(多言語対応)
  - 現時点では、日本語と英語のみ
  - 言語の決定順序
    - Codepage (exerb 版のみ, chcp コマンドで変更)
    - LANG 環境変数 (非exerb版のみ)
    - TECSGEN_LANG 環境変数
    - TECSGEN_FILE_LANG 環境変数 (ファイルのみ)
    - -k オプション (ファイルのみ)

    - 上から順に評価し、一番最後に指定されたものが有効となる
    - Cygwin コンソールでexerb版を動作させると、codepage に従う

TECS ジェネレータ V1.1の新機能

- 固定結合
  - 初期化のための結合の記述が不要

```
[singleton,active]
celltype tInitializer {
    call sInitialize cInit[];        // 添数なしの呼び口配列
};
cell tInitializer Initializer{ };    // ← Cell1, Cell2 へ結合される

celltype tCelltype {
     entry sInitialize eEntry <= tInitializer . cInit;  // ←固定結合
};
cell tCelltype Cell1 { };       // Initializer から呼び出される
cell tCelltype Cell2 { };       // Initializer から呼び出される
```

TECS ジェネレータ V1.1の新機能

- リージョンによるモジュールの分割
  - region の指定子 node, linkunit は、独立したリンク単位であることを表す
    - 直接結合ができなくなった
      - OpaqueRPC プラグインなどで分離すればよい
    - 他の指定子 domain, class も追加になった
      - TOPPERSの仕様に合わせたもの
      - 現時点では記述しても、効果はない
  - region は namespace の機能を含む
    - 名前衝突の回避
      - region 間の結合には、namespace名が必要になった
      - 子region から親region へは namespace 名は不要

TECS ジェネレータ V1.1の新機能

- namespace による名前衝突の回避
  - signature, celltype を namespace の下に置くことができる
    - 名前衝突回避
    - typedef, const, struct を namespace の下に置くことはできない
      - define による名前衝突回避を行っているため
    - cell は namespace の下ではなく region の下に置く
    - namespace … 名前衝突の回避
    - region … cell の配置を制御、名前衝突の回避

TECS ジェネレータ V1.1の新機能

- プラグインの拡充
  - シグニチャプラグイン
    - singnature に対して適用
    - 例) RubyTECSBridgePlugin
  - セルタイププラグイン
    - セルタイプに適用
    - 例) ATK*Plugin
  - セルプラグイン
    - セルに適用
    - 例) RepeatCellPlugin, RepeatJoinPlugin
- セルの id 指定子
  - セルの定義順とは異なる順序で id を付けたい
  - ATK プラグインで使用