EV3RTの内部構造と進んだ使い方

（V1.0ベース）

２０２０年７月１８日

名古屋大学　大学院情報学研究科
附属組込みシステム研究センター
李　奕驍

発表内容

- EV3RTの概要と二つの実行形式
- ユーザアプリケーションとSDK
- ベースシステムの解説
- Platform Interface Layer
- 動的ローディング機能

発表内容

- EV3RTの概要と二つの実行形式
- ユーザアプリケーションとSDK
- ベースシステムの解説
- Platform Interface Layer
- 動的ローディング機能

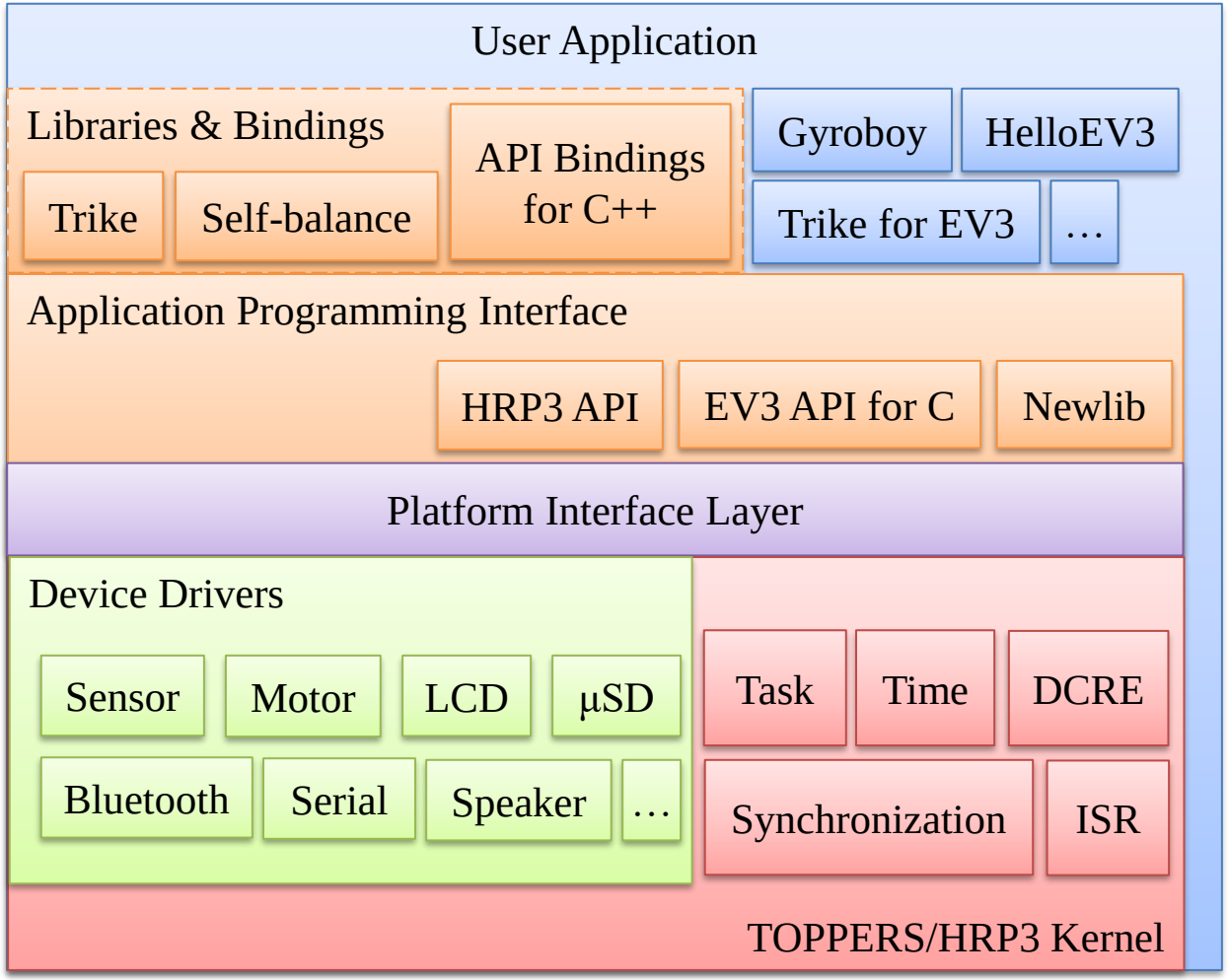
EV3RTの概要と二つの実行形式

- EV3本体
  - ROM・Flash
    - U-Boot（Universal Boot Loader）
      - ✓組込みシステムでよく使われている
    - lms2012
      - ✓LinuxベースのLEGO純正ファームウェア
  - 電源オンからの起動プロセス
    1. SDカードのルートフォルダにU-Boot用起動イメージuImageを確認
    2. uImageがあったら、uImageを実行する
    3. uImageがなければ、lms2012を起動する

- EV3RTでは、実行形式によって異なるuImageを使用
  - スタンドアローン形式と動的ローディング形式

スタンドアローン形式

- RTOSを用いた一般的な組込み開発に近い形式



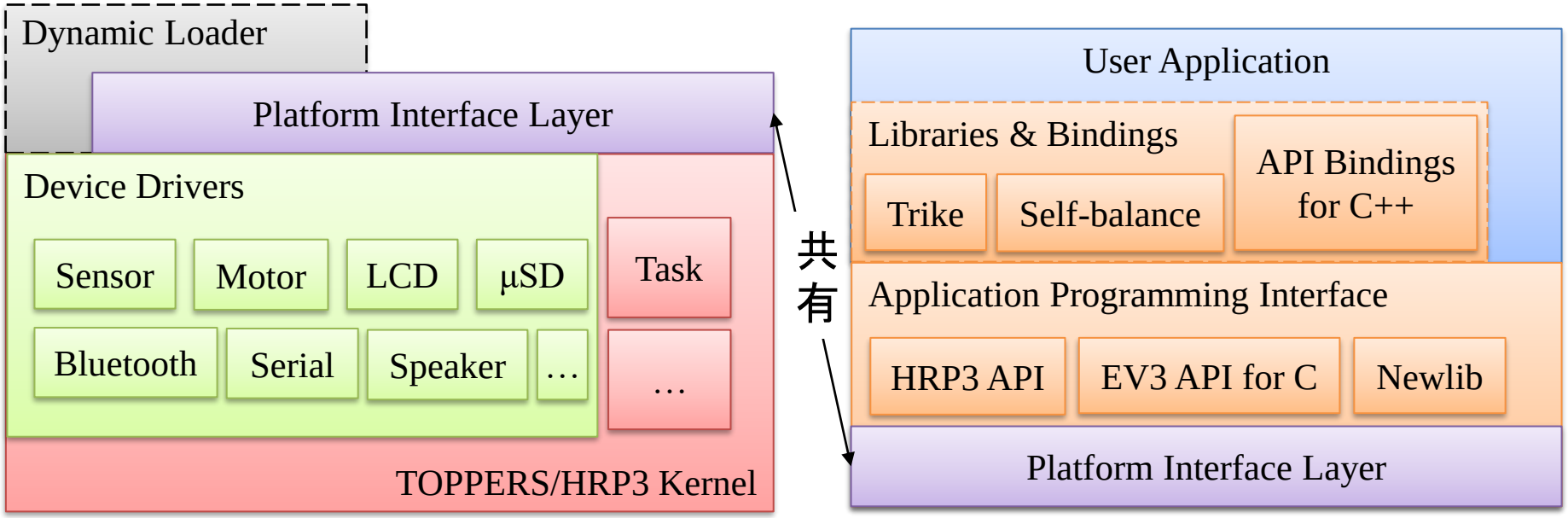
全てを一個のuImageに
- カーネル
- デバイスドライバ
- ミドルウェア
- ユーザアプリ

特徴
- EV3から直接起動
- 低レベル機能使える
- 静的API制限なし
- 特権処理が可能

デメリット
- アプリ選択できない
- 更新に手間かかる（再起動、再接続）

動的ローディング形式

- アーキテクチャは2つの部分に分けられる
  - Linux等汎用OSでの開発に近い形式
  - EV3RTのユーザアプリの開発に推奨



ベースシステム（+ローダ）
[uImage]
GPLを含む複数のライセンス

ユーザアプリ
[独立の実行ファイル]
Apache v2かTOPPERS

動的ローディング形式

- 特徴
  - SDカードに複数のアプリを格納し、選んで実行できる
  - アプリケーションを更新するたびに、再起動・再接続が不要
    - 特にパラメータ調整等で、更新が頻繁な場合、手間が省ける
  - 高レベルAPIで開発、非特権モードの専用ドメインで実行
  - スタンドアローン形式よりOSSライセンスの制限が緩い
  - 動的ローダ自体は一つのスタンドアローン形式アプリ

- デメリット：利用できる機能に一部制限ある
  - デバイスドライバ等、特権モードのものは直接使えない
  - ユーザアプリ用の静的APIはASP3カーネル程度に相当
    - 保護機能制御（SAC_YYY）等はベースシステム側専用

二つの実行形式のビルド方式

- makeコマンドの引数で指定

```
Dynamic:      make app=<folder name> → Loader用Module
Standalone:   make img=<folder name> → U-Boot用uImage
```

- ソースコードの中でビルド方式をマクロで判定できる
  - ユーザアプリの開発者は意識しなくても良い
  - プラットフォームを改造・拡張するときに有用
    - e.g. ev3api.h, app_common.cfg

```
#if defined(BUILD_MODULE)
#include "module_cfg.h"
#else
#include "kernel_cfg.h"
#endif
```

```
#if !defined(BUILD_MODULE)
INCLUDE("ev3.cfg");
#endif

INCLUDE("ev3api.cfg");
```

EV3RTのアーキテクチャ概要

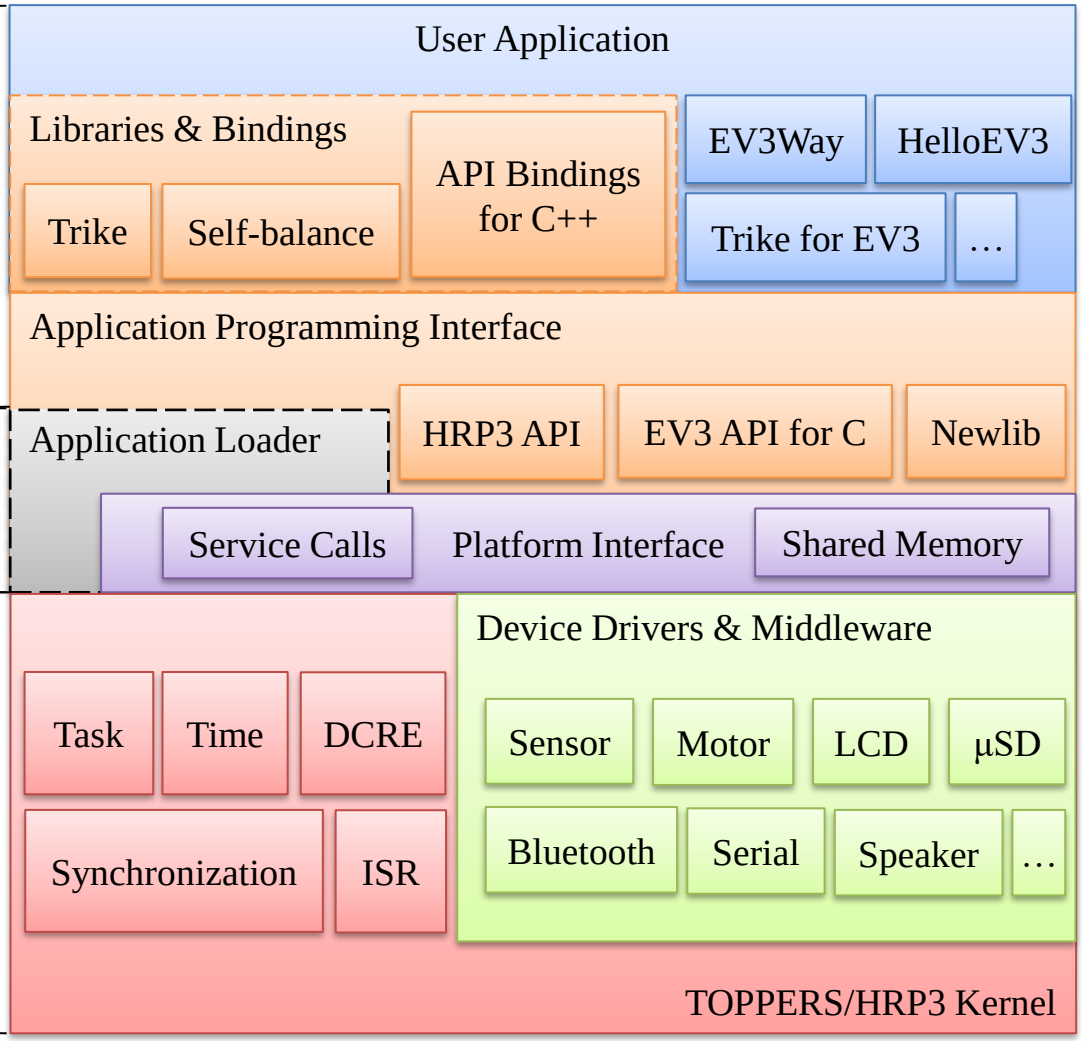
Application

User Application Layer：
非特権モードで実行、
アクセス違反等を検出、
プロジェクト管理機能、
使いやすいAPIとLib、
サンプル・プログラム

Platform Interface Layer：
UALとCSL通信、Loader

Base System

Core Services Layer：
HRP3高信頼RTOS、
アプリからSPFを保護、
PILの必要機能を提供、
各種デバイスドライバ、
主に特権モードで実行



EV3RTを活用しているプロジェクト

- mruby-ev3rt
  - mrubyでEV3RTのユーザアプリを開発
  - ベースシステム側は普通のEV3RTと同じ
  - ユーザアプリ側にmrubyの環境・ライブラリを追加

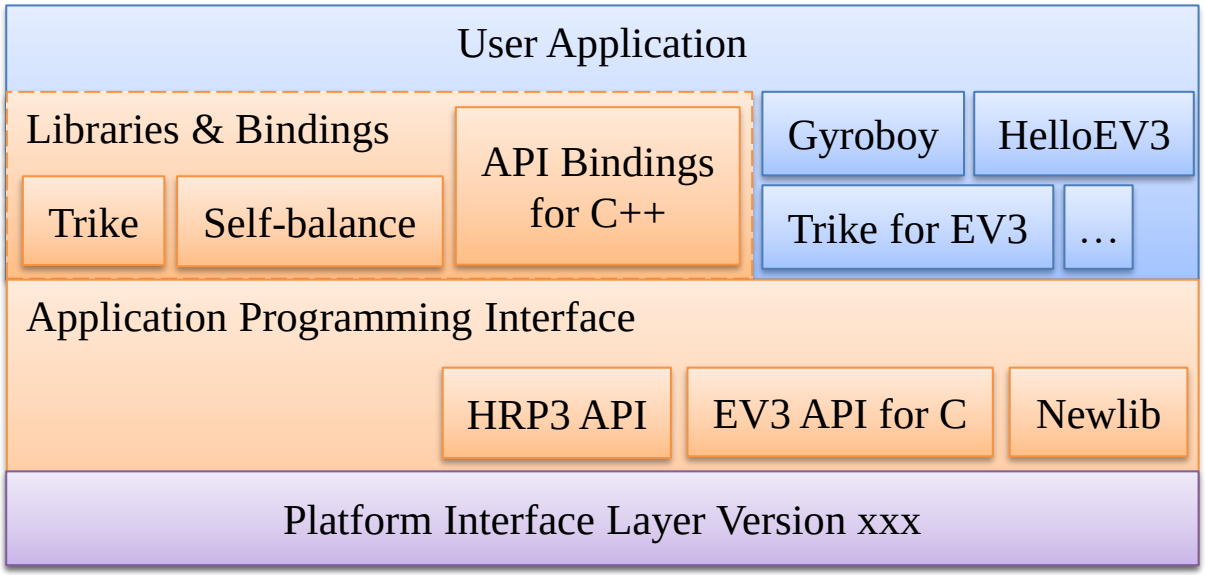
- TOPPERS/EV3RT for Athrill
  - 箱庭シミュレータ上でEV3RTのユーザアプリを開発
  - ユーザアプリ側は普通のEV3RTの実装を再利用している
    - シミュレータ向けのソースコードはほぼそのまま実機で使える
  - ベースシステム側はシミュレータ環境用になっている
    - ASP3カーネル、athrill CPU エミュレータ…

発表内容

- EV3RTの概要と二つの実行形式
- ユーザアプリケーションとSDK
- ベースシステムの解説
- Platform Interface Layer
- 動的ローディング機能

ユーザアプリケーションとSDK

- ユーザアプリ：Mindstorms EV3を制御するプログラム
  - EV3RTが提供するソフトウェア開発キット（SDK）で実装
  - 特定のユーザドメイン（TDOM_APP）で動作
  - 動的ローディング用のアプリモジュールとしてビルド可能
- ソフトウェア開発キット（SDK）の構成
  - API
  - 静的ライブラリ
  - サンプル（& Workspace）



Loadable Application Module

ユーザアプリケーションとSDK

- 主なフォルダ構成（hrp3/sdk）

```
common/ev3api/              APIのソースコード
common/library/             各種静的ライブラリ
workspace/                  ユーザアプリ用Workspace
workspace/helloev3/         サンプルプログラムHelloEV3
firmware/                   ファームウェア用Workspace
firmware/loader/            動的形式のローダ
```

  - アプリ・プロジェクトはWorkspaceの下でフォルダ単位で管理
    - Workspaceもフォルダ単位で管理され、複数作成できる
    - 既存のアプリ、Workspaceをコピーしたら、簡単に新規できる

Application Programming Interface

- TOPPERS/HRP3カーネルAPI
  - 静的APIとサービスコール
  - RTOS機能（タスク管理、同期・通信、タイマ等）
- 標準Cライブラリ（Newlib）
  - 動的メモリ管理、ファイル操作、入出力関数等
- EV3RT C言語 API
  - モータ、センサ、スピーカ、LCD等EV3RT独自の機能
  - Platform Interface Layerを用いて実装

Application Programming Interface

- 標準Cライブラリ（Newlib）
  - 方針：各種ミドルウェアのAPIの違いを吸収
    - メリット：開発者の学習コストを削減できる
    - デメリット：ユーザアプリはミドルウェアのフル機能を使えない
  - TLSFアロケータ → malloc等メモリ管理の標準関数
  - FatFS/SDカード → fopen/read等ファイルI/Oの標準関数
  - Bluetooth SPP → 通信用のファイル（SIO_BT_FILENO）
    - fprintf/fscanf等を使って簡単に仮想シリアルポートで通信できる
    - TOPPERSのシリアル関数でも通信可能（sio_opn_por等）

```
FILE *bt = ev3_serial_open_file(EV3_SERIAL_BT);
for (int i = 1;; i++) {
    fprintf(bt, "Bluetooth counter %d\n", i);
    tlsp_tsk(1000U * 1000U); // Sleep for 1 sec
}
```

静的ライブラリ

- アプリが共通して利用したい機能をライブラリで拡張
  - EV3RTはライブラリを簡単に開発・導入できる機能を持つ
    - ライブラリのプロジェクトはフォルダ単位で管理

```
sdk/common/library/libcpp-ev3   ETロボコン用C++ライブラリ
sdk/common/library/libcpp-test  サンプルライブラリ
```

    - アプリのMakefile（ev3way-cpp/Makefile.incの例）で導入できる

```
...
# Include libraries
include $(EV3RT_SDK_LIB_DIR)/libcpp-ev3/Makefile
...
```

静的ライブラリ

- 静的ライブラリの開発
  - ライブラリ・プロジェクト・フォルダの構成

```
Makefile                          ライブラリのMakefile
src/*                             ライブラリのソースコード
include/*                         アプリが使えるヘッダファイル
<libname>-[loadable|standalone].a 静的lib（オプション）
```

  - ライブラリのMakefile

```
THIS_LIB_NAME := <libname>          # ライブラリの名前
THIS_LIB_OBJS := c-1.o c-2.o        # C言語オブジェクト
THIS_LIB_CXXOBJS := cpp-1.o cpp-2.o # C++オブジェクト
include $(EV3RT_SDK_LIB_DIR)/../Makefile.slib
```

  - 静的ライブラリ（.a）が無い場合、アプリのビルドで自動生成

ユーザアプリケーション

- 初期化タスクでアプリの実行に必要な処理を行う

```
void _app_init_task(intptr_t unused) {
    brick_misc_command(CMD_BUSY_WAIT_INIT, 0);

    ER ercd = fetch_brick_info(&_global_ev3_brick_info);

    _initialize_ev3api_lcd();
    _initialize_ev3api_motor();
    _initialize_ev3api_newlib();
    _initialize_ev3api_sensor();

    // Call C++ global constructors
    extern func_ptr __init_array_start[0], __init_array_end[0];
    for (func_ptr* func = __init_array_start; func != __init_array_end; func++)
        (*func)();

    // EV3_CRE_CYC
    _initialize_ev3api_cyc();
}
```

  - SDK内部のタスク、ユーザで作成・定義する必要がない
  - 注意：C++ Global Constructorは待ちになるコードを使えない

ユーザアプリケーション

- 実行時の主なタスク構成

優先度	タスク	周期	実行時間
3	Bluetooth（高）	20ms	<1ms
4	LCDリフレッシュ	40ms	<1ms
TMIN_APP_TPRI - 1	ユーザ周期ハンドラ	User-defined	User-defined
>= TMIN_APP_TPRI	ユーザ アプリケーション		
TMAX_TPRI	Bluetooth（低）		

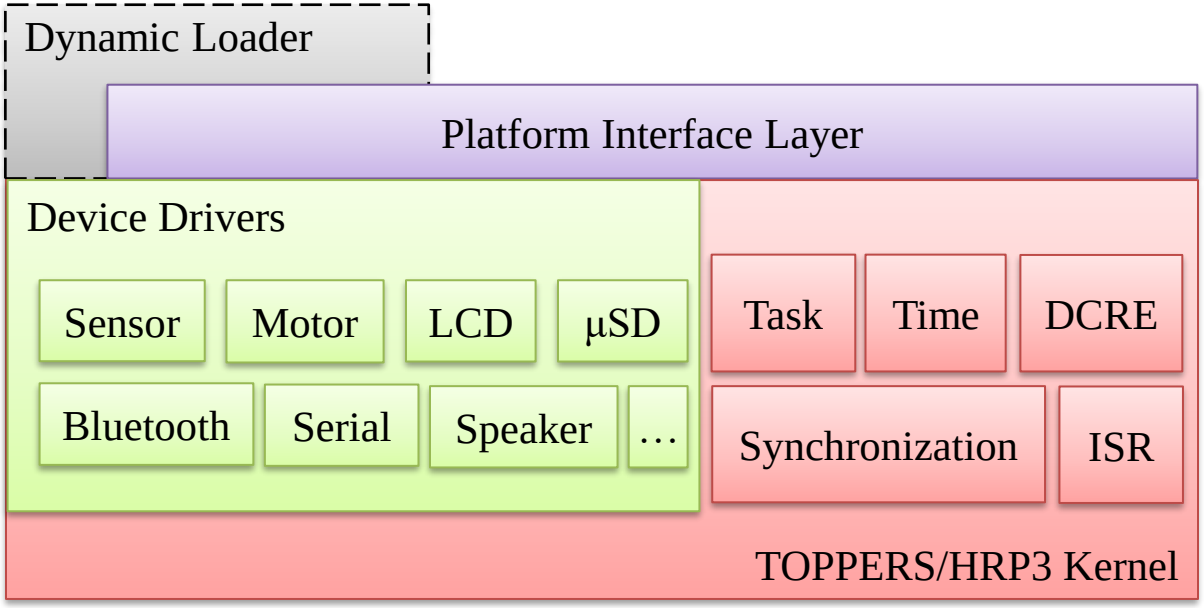
  – 汎用OSよりBackground Tasksが少なくリアルタイム性が良い

発表内容

- EV3RTの概要と二つの実行形式
- ユーザアプリケーションとSDK
- ベースシステムの解説
- Platform Interface Layer
- 動的ローディング機能

EV3RTのベースシステム

- ユーザアプリの実行をサポートするためのサービスを提供するシステムソフトウェア
- 構成
  - RTOSカーネル
  - 各種ドライバ
  - Platform I/F
  - ミドルウェア
    (e.g. Loader)



SPF構築コストの課題

- デバイスドライバ・ミドルウェアの膨大化

	Mindstorms NXT	Mindstorms EV3
Processor	ARM7@48MHz	ARM9@300MHz
Coprocessor	Atmel AVR@8MHz	n/a
ROM	256KB	16MB
RAM	64KB	64MB
Sensors	Analog, I2C	Analog, I2C, UART
Motors	Up to 3	Up to 4
Display	100×64 LCD	178×128 LCD
SD card	n/a	microSD (SDHC)
Bluetooth	v1.2	v2.1 + EDR
USB device	12Mbps	480Mbps
USB host	n/a	12Mbps
WiFi	n/a	USB dongle

性能・機能向上

Lines of Code

		nxtOSEK	lms2012
Mindstorms	LCD	157	489
	Speaker	389	597
	Motor	137	3,176
	Sensor	235	5,620
	Soft I2C	421	1,598
	Soft UART	n/a	16,889
SoC	SPI	135	919
	AVR	257	n/a
	SD	n/a	983
Generic	FAT	n/a	5,375
	USB	664	16,746~
	Bluetooth	330	14,186~
	WiFi	n/a	34,763~

ソフトウェア大規模化

– ゼロから実装するのは現実的でない
  - OSS活用等、コスト削減の手法を検討する必要がある

純正SPFのデバイスドライバ・ミドルウェア構成

- 汎用ミドルウェア（ネットワーク、ファイルシステム等）
  - （ユーザにとって）ターゲットに依存しない機能を提供
  - 組込み向けの軽量的・移植性良い実装も幾つかある
- SoC周辺デバイスのドライバ（シリアル、DMA等）
  - RTOS用サンプル・HAL等をメーカが提供していることが多い
- Mindstorms専用デバイスドライバ（センサ、モータ等）
  - Linux用ソースコードのみ、公開資料殆ど存在しない
  - 各デバイスを解析するのは困難で、既存ドライバを再利用
    - Linuxデバイスドライバの再利用手法を提案

Linuxデバイスドライバの再利用

Mindstorms専用Linuxデバイスドライバ

構成	Linux In-kernel API	ドライバ本体	アクセスI/F
分析	IRQ管理等の基本機能を使用、Linux独自機能ほぼ使用しない	Kernel Module形式、名前衝突等はある	特殊ファイル経由、オーバヘッド大きい
対応	RTOS APIでLinuxカーネル基本機能の互換レイヤを実装	マクロ定義等で衝突解決、静的リンク	オーバヘッド小さいサービスコール等に

```
// Include the compatibility layer
#include "driver_common.h"

// Hacks for this module
#define InitGpio PWM_InitGpio
static void SetGpioRisingIrq(...) {
    ...
}
...

// Include source file to be reused
#include "d_pwm.c"

// Interfaces
ER_UINT extsvc_motor_command(...) {
    ...
    Device1Write(...);
    ...
}
...
```

再利用の例

互換レイヤ
＋
新規コード
（1,033行）

再利用した
Linux
ドライバ
（32,183行）

互換レイヤ：
Memory、Locking、IRQ、HiRes Timer…

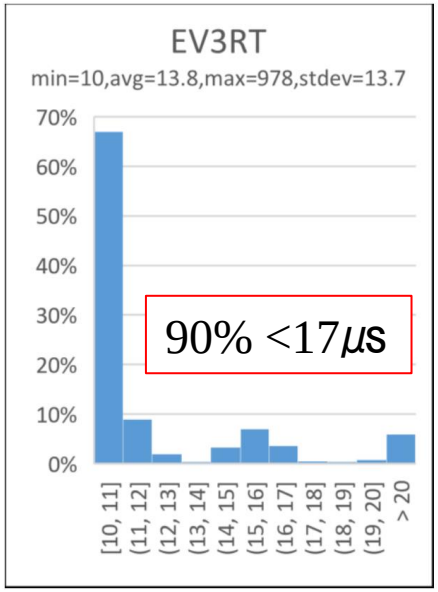
Linuxドライバ：
モータ、センサ、音声、LCD、I2C、Soft UART…

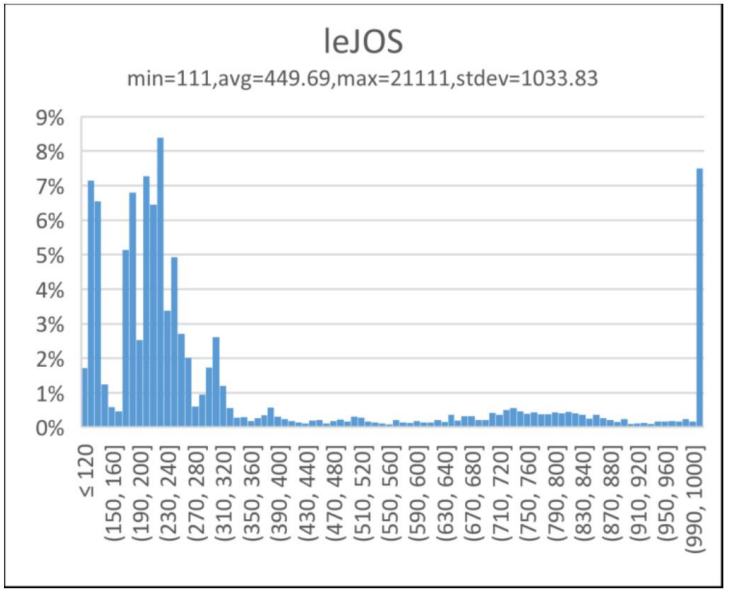
Linuxデバイスドライバの再利用

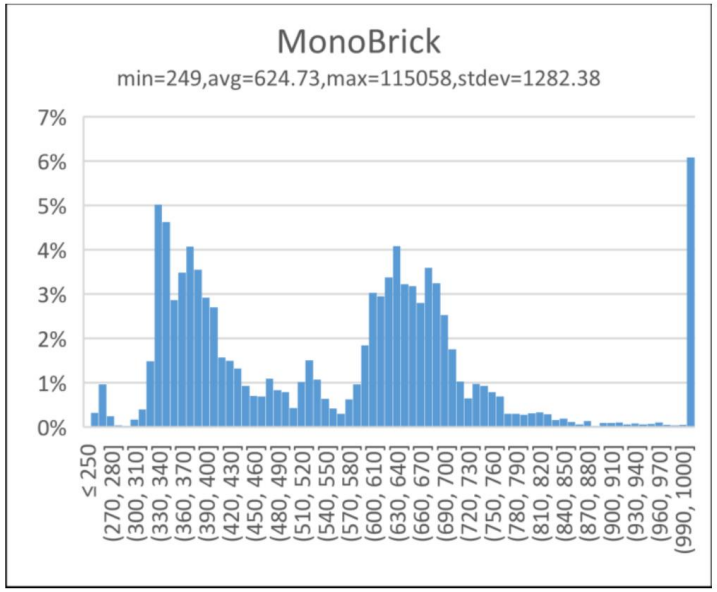
- オーバヘッド評価（モータドライバの場合）

Function	Average time	Maximum
ev3_motor_set_speed()	4.54 us	40 us
ev3_motor_set_speed()[Linux]	596 us	3424 us

- APIのオーバヘッドとばらつき（モータ制御の例）

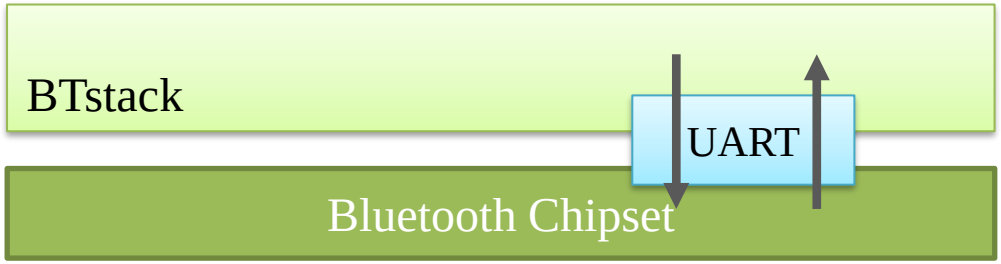






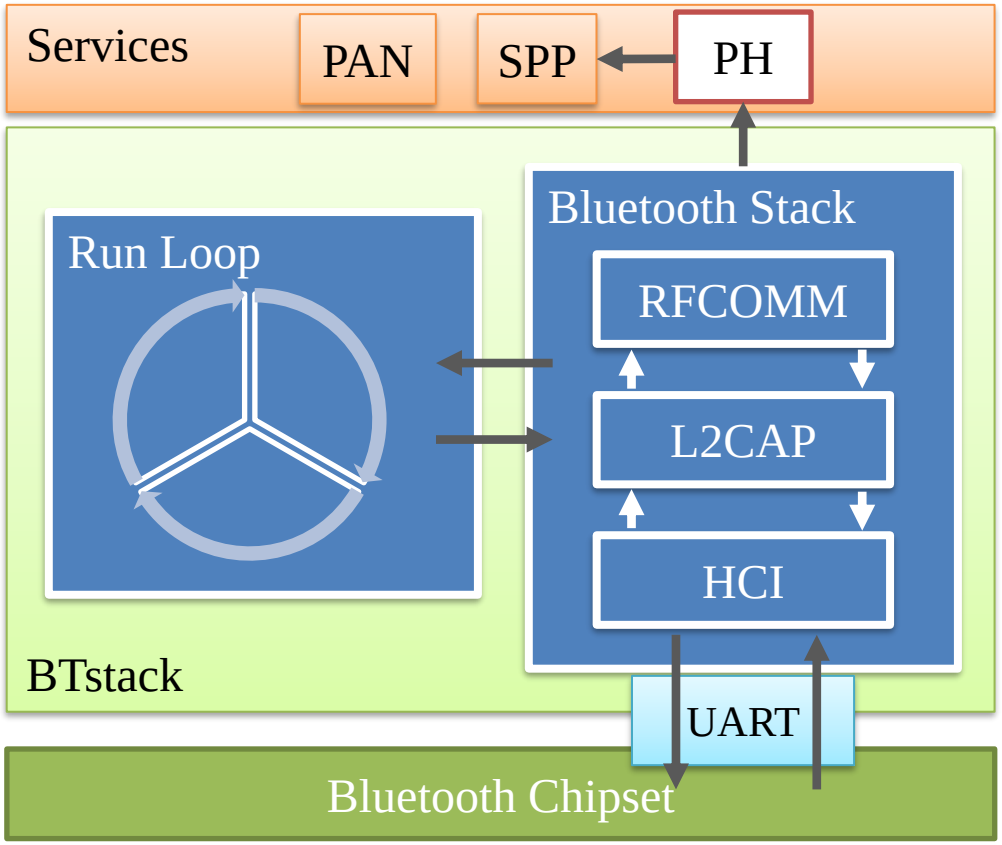
Bluetooth通信機能

- BTstackというオープンソースのスタックドライバを使用
  - 特徴
    - 軽量で組込み向け、OSなしでも動作できる
    - 機能が豊富、Bluetooth SIGにより認証済み
    - 対応するチップが多い、EV3のCC256x（v2.1 EDR）もサポートする
    - EV3RT用fork： https://github.com/ev3rt-git/btstack
  - CC256xとSoCはUARTで通信するので移植はとても簡単
    - UART操作関数でBTstackのHCIインタフェースを実現したら良い
    - UARTドライバの実装にDMAを使って性能向上、CPU負荷を軽減



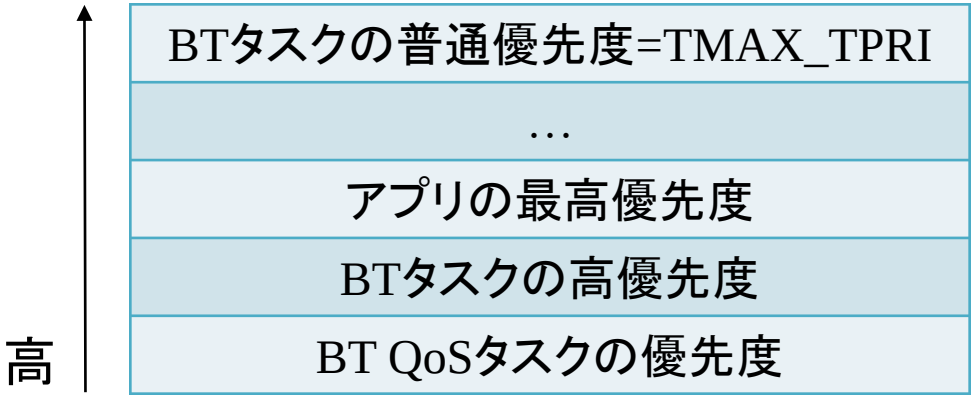
Bluetooth通信機能

- BTstackのAPIを使ってサービスを実装
  - SPP（Serial Port Profile）とPAN（Personal Area Network）対応
  - SPPの仮想シリアルはTOPPERS SIOドライバでも使えるようにした。接続された時にopen、切れたらcloseされる
  - 受送信データを処理するためにRun Loopが必要。Run Loop用のBTタスクを作成した



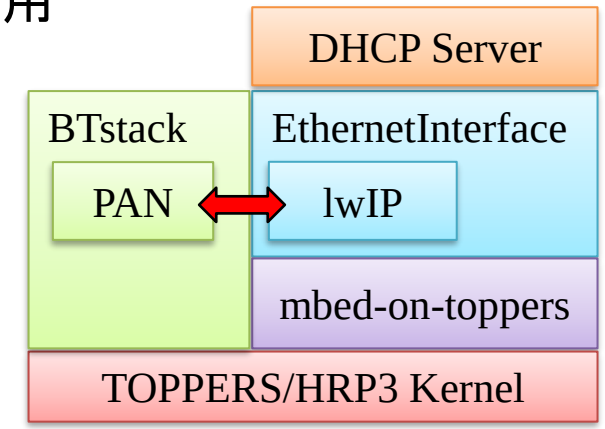
Bluetooth通信機能

- Bluetooth通信のQoS確保
  - 課題：BTタスクの優先度
    - アプリより高い優先度で動作すると
      - ✓高速で通信する場合、アプリのリアルタイム性に悪影響
    - アプリより低い優先度で動作すると
      - ✓スタベーション（例：無限ループ）で接続が切れてしまう可能性がある
  - 接続を保護するために、QoSタスクで周期的にBTタスクの優先度を変更
    - 現時点のポリシー　20ミリ秒ごとに1ミリ、BTタスクをアプリより高い優先度で実行させる
      - ✓参考：倒立制御処理　5ミリ秒周期で動作、実行時間は1ミリ秒以内



Bluetooth通信機能

- Bluetooth PANでネットワーク通信（TCP/IP）を対応
  - HTTP経由のアプリ更新に必要な機能
    - Beta 7以前はSPPのみ対応、アップロードの時、毎回Tera Term等からファイルやSPPの番号（環境依存）を指定する必要がある
    - Beta 7からPAN対応、「make upload」だけで簡単にアップロードできる
  - 課題：TOPPERS/HRP3用DHCPサーバのミドルウェアがない
    - EV3とPCのIPアドレスを自動設定するために必要
  - ARM mbed OS互換レイヤ（mbed-on-toppers）で対応
    - TCP/IPスタック（lwIP）とDHCPサーバを使用
    - BTstack PANとlwIPのbridgeを実装
  - 現時点、PANはスタンドアローン専用
    - 動的のユーザアプリから使用できない



その他の主なデバイスドライバ・ミドルウェア

- Console機能（drivers/console）
  - LCDでの操作メニューとログ表示

- ファイルシステムとSDカード（drivers/fatfs）
  - 汎用FATファイルシステムモジュール FatFS
    - Windows互換FATファイルシステムをサポート
    - 下位レイヤインターフェースを実装して、SDカードを操作できる

その他の主なデバイスドライバ・ミドルウェア

- USB Mass Storage Class（drivers/usbmsc）
  - SoCメーカが提供したUSBスタックTI StarterWare USBで開発
  - EV3のMini USBをPCに接続、SDカードリーダとして使える

- 設定ファイル機能（platform/src/ev3rt_config.c）
  - 一部の設定（例：Bluetooth PIN）をrc.conf.iniで変更できる
    - ベースシステムのリコンパイルが不要
  - INIファイルの解析にminIniというライブラリを使用

- バッテリ残量警告
  - 周期ハンドラEV3_BATTERY_MONITOR_CYCで電圧をチェック。低下の時、警告メッセージ出力＋LEDを赤に設定

その他の主なデバイスドライバ・ミドルウェア

- プラットフォーム初期化（platform/src/ev3main.c）
  - デバイスドライバ等を適切な順序で初期化する
  - EV3RTの起動バナーを表示

- TLSFメモリアロケータ（ev3_gcc/TLSF-2.4.6）
  - リアルタイムシステム向けの動的メモリアロケータ
    - O(1)の計算量でmalloc/free
  - 動的生成機能拡張と標準Cライブラリのために導入
  - 複数のメモリプールを対応
    - ベースシステム用メモリプールとアプリ用メモリプールを別々に管理できるのでHRP2のメモリ保護機能との相性が良い

ベースシステムのカスタマイズ

- 各処理優先度、ユーザアプリのリソース制限等の変更
  - リビルドが必要
    - 静的に生成されたリソースの情報（ページテーブル等）が変わった
    - 設定ファイルを書き換えるだけで対応できない

```
$ cd sdk/firmware; make img=loader
```

  - カスタマイズ可能な項目はev3_gcc/ev3.hにまとめてある

```
#define TMAX_APP_TSK_NUM     (32)
#define TMAX_APP_SEM_NUM     (16)
#define TMAX_APP_FLG_NUM     (16)
#define TMAX_APP_DTQ_NUM     (16)
…
#define TMAX_APP_TEXT_SIZE   (1024 * 1024)
#define TMAX_APP_DATA_SIZE   (1024 * 1024)
#define TMAX_APP_BINARY_SIZE (1024 * 1024)
```

e.g.
Application
Moduleの
リソース制限

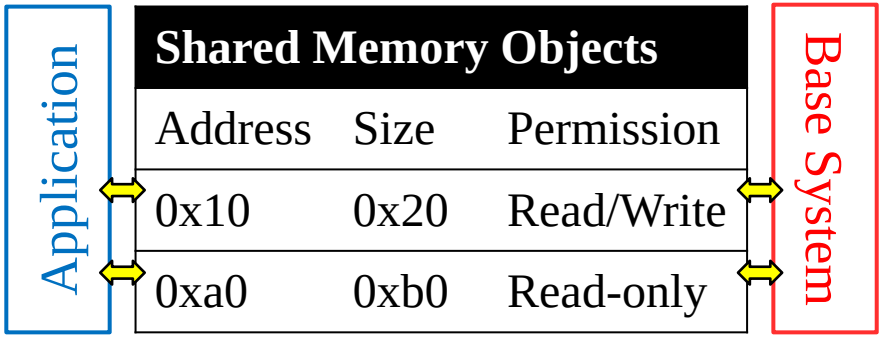
発表内容

- EV3RTの概要と二つの実行形式
- ユーザアプリケーションとSDK
- ベースシステムの解説
- Platform Interface Layer
- 動的ローディング機能

Platform Interface Layer

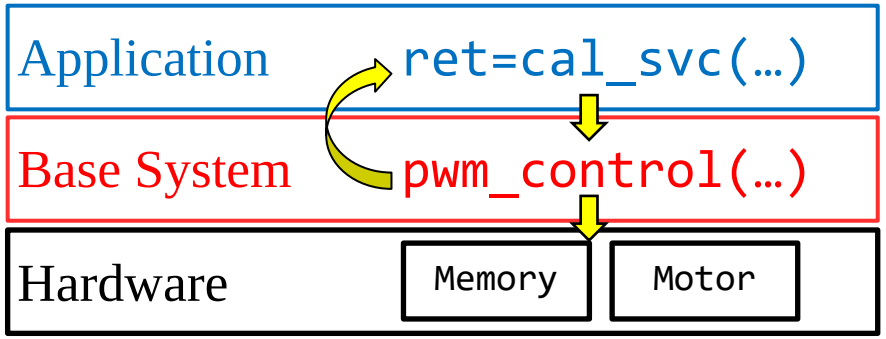
- アプリとSPF間通信の高信頼化

共有メモリ



- PILはメモリオブジェクトを適切に登録
- アプリから直接にアクセスできる
- オーバヘッドは極めて小さい
- 例：
  アプリ用LCD Frame Buffer → R/W
  センサ用データ構造 → Read-only

サービスコール（SVC）



- PILはSVCを登録し、I/Fを提供
- アプリからはトラップ命令で呼び出し
- オーバヘッドは比較的に大きい
- SPFは特権実行のSVC本体を実装：
  引数、デバイス状態等をチェック
  高度な処理を安全に実行できる

- アプリとSPFのソースコード依存を解消
  –別々に開発、ビルド可能で保守性、生産性向上

Platform Interface Layer

- 共有メモリ（brickinfo_t）のインタフェース
  - アプリから直接アクセス可能のメモリ領域ポインタの集合
  - ポインタ値はベースシステム
  - のドライバ初期化で設定
  - ユーザアプリは起動時、SVC _fetch_brick_info()でベースシステムから取得

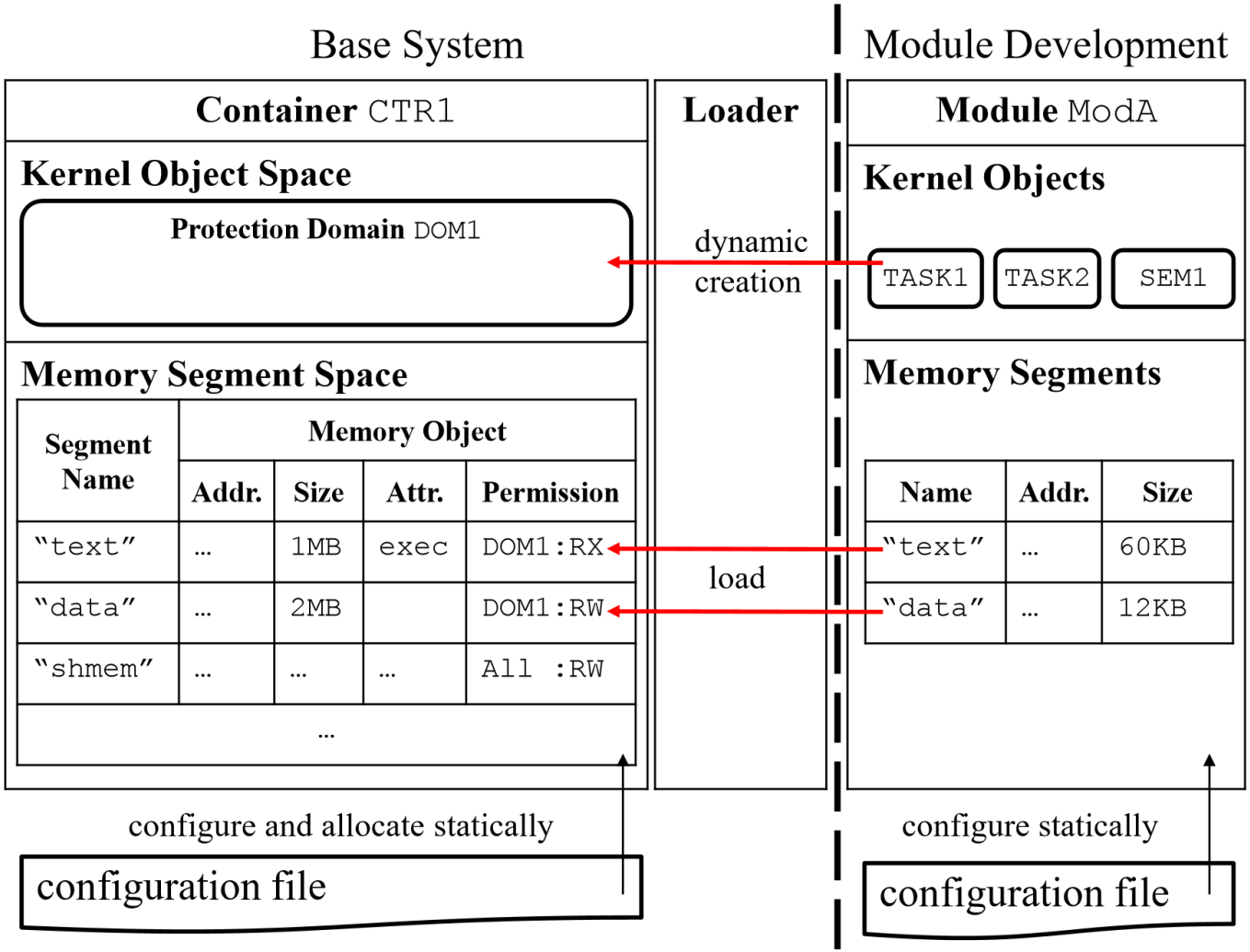
```
typedef struct {
    uart_data_t      *uart_sensors;
    analog_data_t    *analog_sensors;
    i2c_data_t       *i2c_sensors;
    motor_data_t     *motor_data;
    uint8_t          *motor_ready;
    volatile bool_t  *button_pressed;
    bitmap_t         *lcd_screen;
    font_t           *font_w6h8;
    font_t           *font_w10h16;
    void             *app_heap;
    int16_t          *motor_current;
    int16_t          *battery_current;
    int16_t          *battery_voltage;
    int16_t          *battery_temp;
} brickinfo_t;
```

発表内容

- EV3RTの概要と二つの実行形式
- ユーザアプリケーションとSDK
- ベースシステムの解説
- Platform Interface Layer
- 動的ローディング機能

動的ローディング機能

- 概要



Container：
BaseSysで定義。
最大リソース、
メモリ保護は
静的に指定。

Module：
独立のELF File。
Kernel Objectは
Cfgで静的定義。
Cfg内容を格納。

Loader：
Moduleの内容に
従って動的生成。

動的ローディング機能

- HRP3カーネルの動的生成機能拡張（DCRE）
  - Cfgで動的に生成できるリソースの最大数を静的指定・確保

```
AID_TSK(32); // 最大32個のタスクが生成できる
```

  - C言語でのカーネルオブジェクトの動的生成が可能に

```
typedef struct t_ctsk {
    ATR      tskatr;  /* タスク属性 */
    intptr_t exinf;          /* タスクの拡張情報 */
    TASK     task;           /* メインルーチン */
    ...
} T_CTSK;
ER_ID tskid = acre_tsk(pk_ctsk); // タスクを生成
```

  - EV3RTのローダはこの機能でアプリのオブジェクトを生成

ユーザアプリのモジュール

- 実行ファイルの形式
  - 独立のELFファイル
    - ベースシステムのコードに依存しない → 動的シンボル解決は不要
    - PIL、API、ライブラリは静的にリンク → 動的リンク機構は不要
    - 単一のドメイン（TDOM_APP）に属するため、コードとデータのメモリ保護設定は単純で、textセクションとdataセクションがあれば十分
  - Module Configuration TableをELFファイルに格納する
    - モジュールの静的API（cfgファイル）のデータ構造
    - ローダでモジュールのオブジェクトを動的に生成するために必要
  - 完全位置独立コード（-mno-pic-data-is-text-relative）でビルド
    - ローディング用メモリ領域はベースシステムの設定により変化する
  - 課題：実際のオブジェクトIDの取得
    - ユーザアプリに使うIDの実際の値はロードしないと分からない

Module Configuration Table

- Module Configuration Entryの配列

```
typedef struct {
    FN sfncd; const void *argument; void *retvalptr;
} MOD_CFG_ENTRY;
```

  - sfncd：静的APIの機能コード　e.g. TSFN_CRE_TSK/SEM/FLG/MTX
  - argument：引数へのポインタ、生成情報のパケットの場合が多い
  - retvalptr：　動的生成の関数の戻り値を格納する変数へのポインタ

–静的API　CRE_TSKの例：

```
CRE_TSK(APP_INIT_TASK,
        { TA_ACT, 0, _app_init_task, TPRI_APP_INIT_TASK, STACK_SIZE, NULL });
```

```
ID _module_id_APP_INIT_TASK __attribute__((section (".module.text")));
static const T_CTSK _module_ctsk_tab[] = {
    { TA_ACT, 0, _app_init_task, TPRI_APP_INIT_TASK, ROUND_STK_T(STACK_SIZE), ... },
};
const MOD_CFG_ENTRY _module_cfg_tab[] = {
    { TSFN_CRE_TSK, &_module_ctsk_tab[0], &_module_id_APP_INIT_TASK },
};
```

オブジェクトIDの生成と取得

- モジュール開発環境とローダ内部処理の流れ
  1. オブジェクトIDのマクロと格納するための変数を定義

```
extern ID _module_id_APP_INIT_TASK;
#define APP_INIT_TASK ((const ID)(_module_id_APP_INIT_TASK))
```

  2. ID用変数のポインタをMod Cfg Entryの戻り値として登録

```
const MOD_CFG_ENTRY _module_cfg_tab[] = {
    { TSFN_CRE_TSK, &_module_ctsk_tab[0], &_module_id_APP_INIT_TASK },
};
```

  3. ローダが動的生成関数（acre_tsk等）を呼び出してオブジェクトIDをMod Cfg Entryの戻り値として設定
  4. ユーザアプリは変数のマクロでIDを参照しているため、実際の値を使用することになる

アプリローダ

- ローディング操作の流れ
  0. コンテナがモジュールのセクションを格納する領域を用意

```
static STK_T app_text_mempool[COUNT_STK_T(TMAX_APP_TEXT_SIZE)]
        __attribute__((section(".app_text_mempool"),nocommon));
static STK_T app_data_mempool[COUNT_STK_T(TMAX_APP_DATA_SIZE)]
        __attribute__((section(".app_data_mempool"),nocommon));
```

  1. ELFファイルのセクションを上記の領域にコピー
  2. 完全位置独立コードのrelocationを行う
  3. Module Cfg Tableを参照して順次にオブジェクトを生成
  4. 生成したオブジェクトのIDをretvalptrに格納
  5. 全てのオブジェクトが生成されたら、ロード完了
  6. 生成に失敗するオブジェクトがあったら、各retvalptrを参考して今まで生成した全てのオブジェクトを削除

※Unloadは生成したタスクを強制終了してオブジェクトを削除

ご清聴ありがとうございました