TECSをサポートするコンポーネント設計ツール ZIPC Toy! と活用法

キャッツ株式会社
ソフトウェア事業部
今井 良和

目次

- TECSについて
- コンポーネント記述言語 CDL
- ツールチェーン
- ZIPC Toy! ： コンポーネント設計
- ZIPC ® ： コンポーネントの振る舞い設計

- 実演： LEGO® Mindstorms NXT

TECSについて：概要

TECS ： Toppers Embedded Component System
組込み向けにインタフェースを具体化したコンポーネントシステム

- 目的
  - ソフトウェアをコンポーネント構造にすることで見通しを向上
  - コンポーネント記述を標準化して部品の再利用，部品流通を促進
- 特徴
  - コンポーネントの静的な生成と結合
  - ソフトウェアの構造を **コンポーネント記述言語（CDL）** で表現
  - C言語の曖昧さを排除したインタフェース定義、データ構造定義

TECSについて：ソースベースの部品化の課題

従来： ソースの部品化

製品A
・ソナーセンサー（障害物検知）
・モーター2個　（右/左車輪）
・障害物があれば右回転
　なければ前進

Controller.c

```
…
Task(Driver)
{
    uint8_t level = SonarSensor_get();
    uint8_t output = 0;
    if (level > 100) {
       Driver_set(10, 0); //前進
    } else {
       Driver_set(10, 10);    // 右回転
    }

    TerminateTask();
}
…
```

SonarSensor.c

```
…
uint8_t SonarSensor_get(void)
{
  return sensor_abc(PORT_LS);
}
…
```

Driver.c

```
…
void Driver_set(uint8_t speed,
                uint8_t turn)
{
  nxt_motor_set_speed(
        PORT_MTR, speed+turn, 0);
  nxt_motor_set_speed(
        PORT_MTL, speed(-turn,0);
}
…
```

他の製品にソースを活かしたいが…

製品B
・**光センサー**（障害物検知）
・モーター2個　（右/左車輪）
・障害物があれば右回転
　なければ前進

→ Controller.c を変更
LightSensor.cを追加

製品C
・ソナーセンサー（障害物検知）
・**モーター2個（後輪, ステアリング）**
・障害物があれば右回転
　なければ前進

→ Driver.cを変更

製品D
・ソナーセンサー（障害物検知）
・**光センサー**（ライン検知）
・モーター2個　（右・左回転）
・**障害物があれば後退**
　ライン上なら左回転
　ライン上でなければ右回転

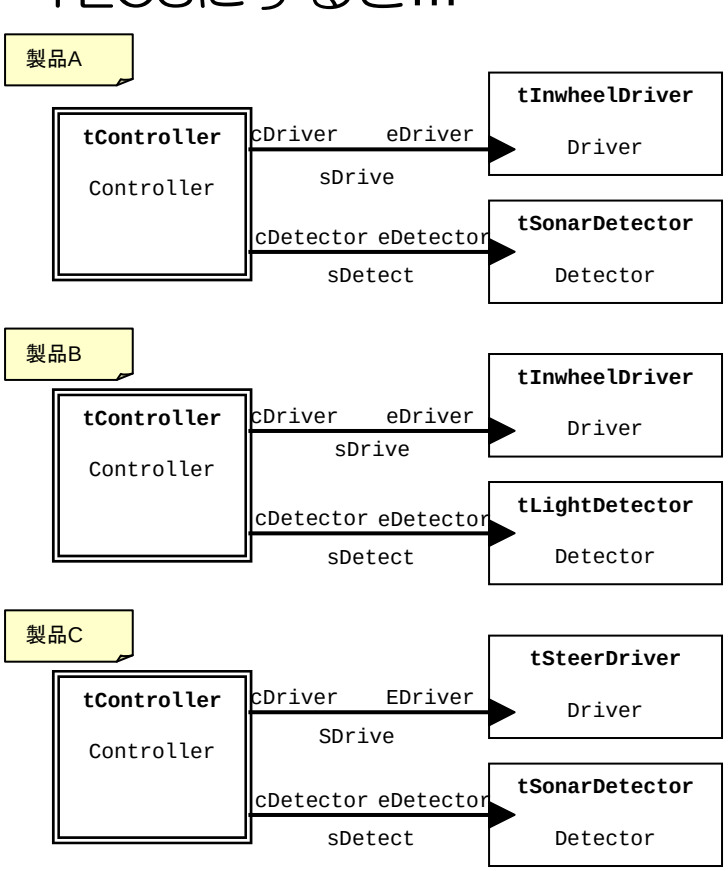
→ Controller.c を変更
LightSensor.cを追加
Driver.cを変更

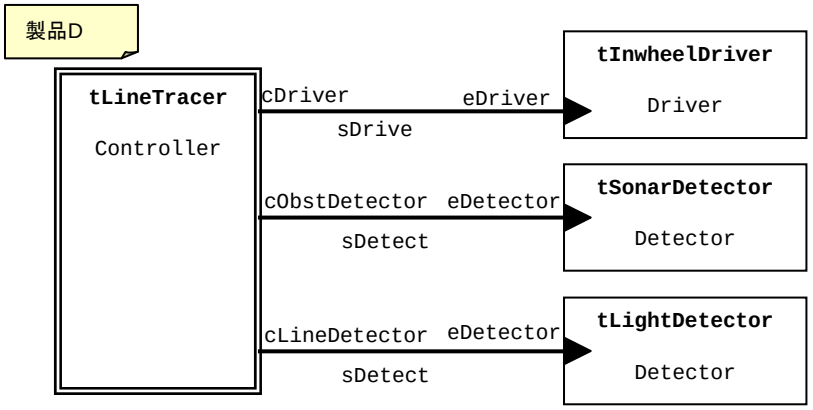
再利用できないソースコードが多い

写真：http://lejos-osek.sourceforge.net/jp

TECSについて :コンポーネントシステムの利点

TECSにすると…

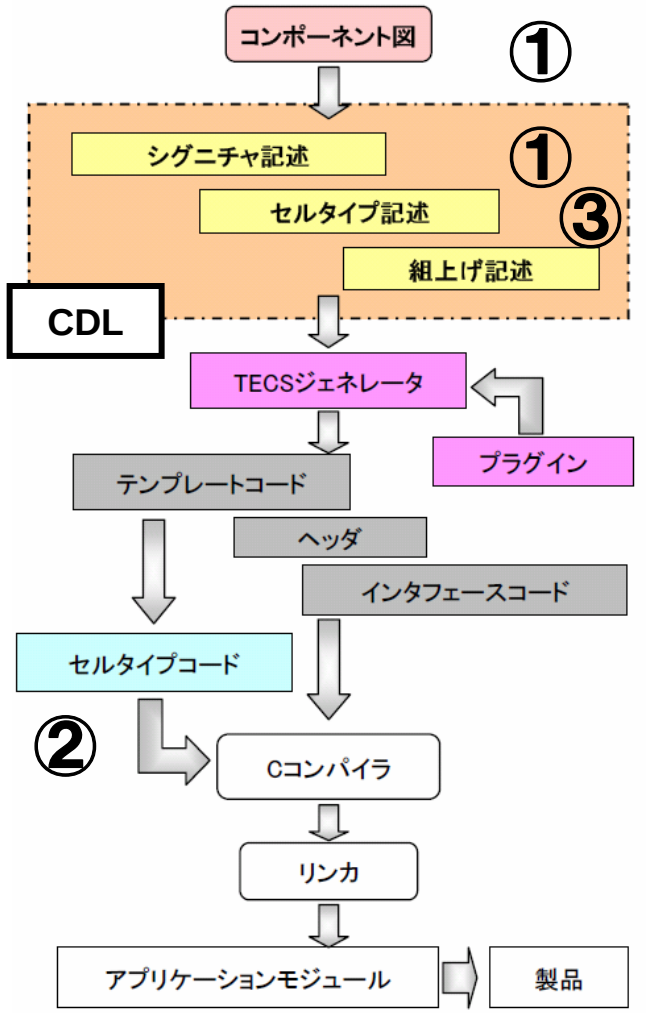




部品 ＼ 製品	製品A	製品B	製品C	製品D
tController	○	○	○	
tLineTracer				○
tInwheelDriver	○	○		○
tSteerDriver			○	
tSonarDetector	○		○	○
tLightDetector		○		○

再利用性が向上

TECSについて：開発の流れと開発者の役割



設計

実装

① アーキテクチャ設計者

・どのような部品が必要とされるか
・どのように振る舞うべきか
・どのようなインターフェースか
を設計する

② コンポーネント開発者

・コンポーネントの振る舞いを実装

③ アプリケーション開発者

・コンポーネントを配置
・インタフェースを接続
・パラメータを入力

コンポーネント記述言語 CDL：構成要素

TECS CDL は主に３つの記述から構成

シグニチャ記述
コンポーネントのインターフェース関数を定義する

セルタイプ記述
コンポーネントが持つ受け口、呼び口を定義する

組み上げ記述
コンポーネント間の呼び出し関係を定義して
システムを組み上げる

CDLによって
コンポーネントのインタフェースを明確に定義
ソースコードは組み上げ（他のコンポーネント）に依存しない

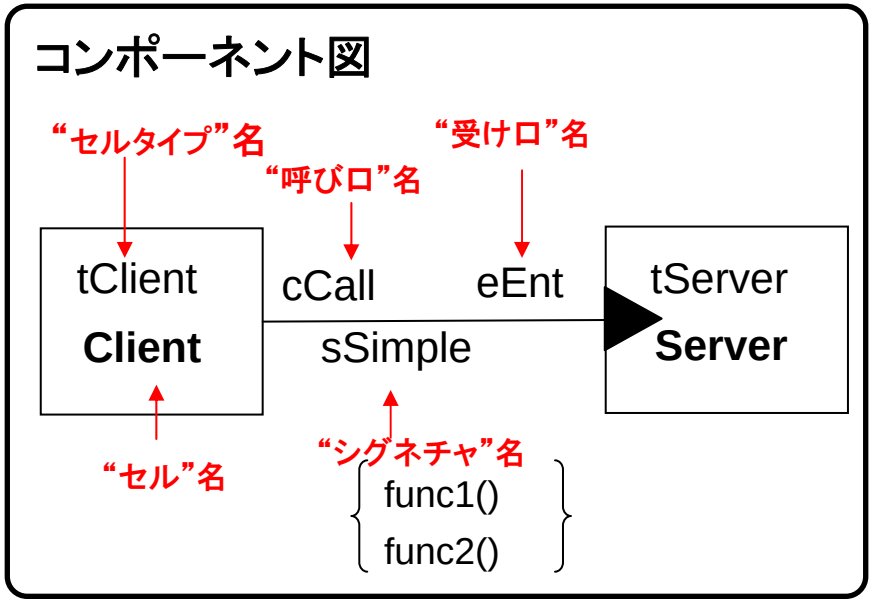
→ コンポーネントの再利用性が向上
ソースをソフト部品として流通できる

```
import_C( "tecs.h" );

typedef int32_t ER;
                                シグニチャの定義
signature sSimple {
  ER  func1( [in]int32_t inval );
  ER  func2( [out,string]char_t
*str );
};
                                セルタイプの定義
celltype tServer {
  entry sSimple eEnt;
};

celltype tClient {
  call sSimple cCall;
};
                                組み上げ記述
cell tServer Server {
};

cell tClient Client {
  cCall = Server.eEnt;
};
```

コンポーネント記述言語 CDL：定義

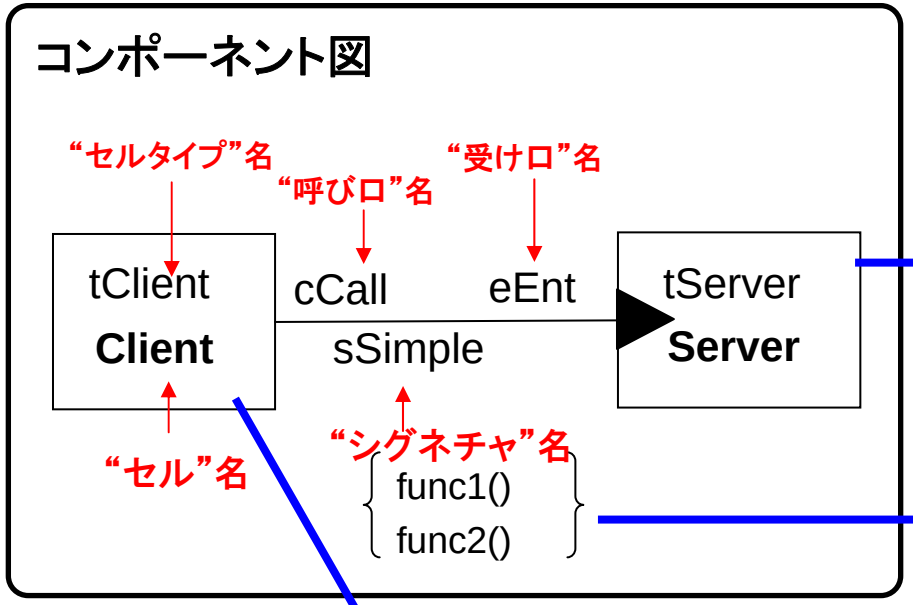
- TECSで使用するコンポーネント記述言語
  (TECS Component Description Language)

- TECS CDLの用語
  セル
  セルタイプ
  シグニチャ
  呼び口, 受け口
  セル属性
  セル変数



言語で表現
TECS CDL

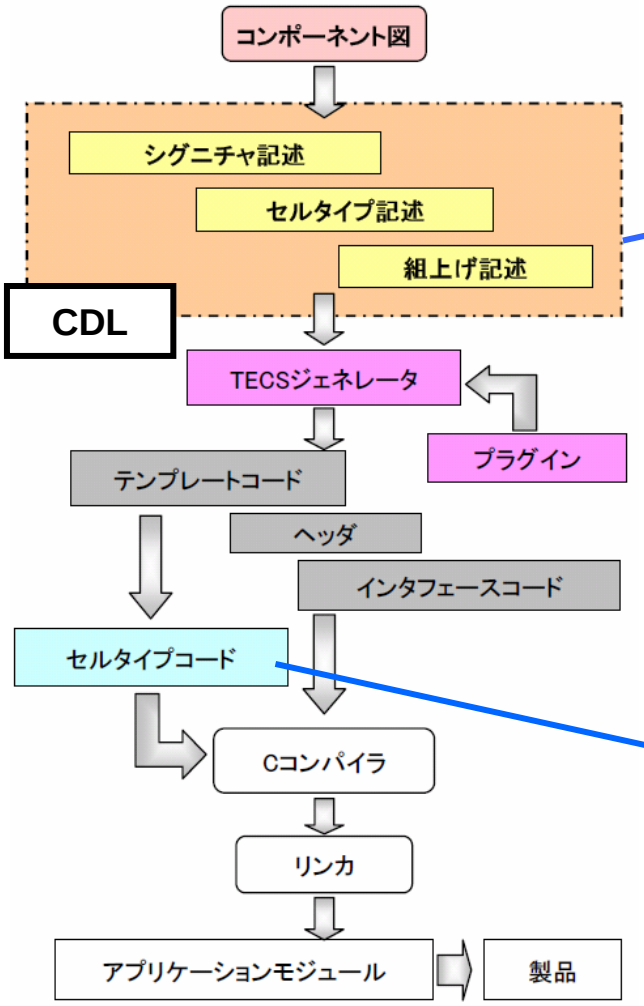
コンポーネント記述言語 CDL：図との対応



CDL

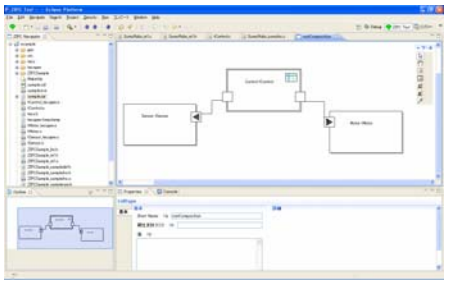
```
import_C( "tecs.h" );

typedef int32_t ER;
                                  シグネチャの定義
signature sSimple {
  ER  func1( [in]int32_t inval );
  ER  func2( [out,string]char_t *str );
};
                                  セルタイプの定義
celltype tServer {
  entry sSimple eEnt;
  attr {  const uint16_t  port };
  val { int8_t clients; }          セル属性
};                                 セル変数

celltype tClient {
  call sSimple cCall;
  attr { const uint16_t port};
};
                                  組み上げ記述
cell tServer Server {
 port = 1000;
};

cell tClient Client {
  cCall = Server.eEnt;
  port = 21000;
};
```

ツールチェーン



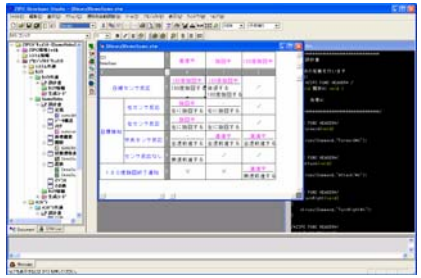
アーキテクチャ設計
（コンポーネント設計）
組み上げ記述

ZIPC Toy!
Technology of youth



振る舞い設計
実装 （自動コード生成）

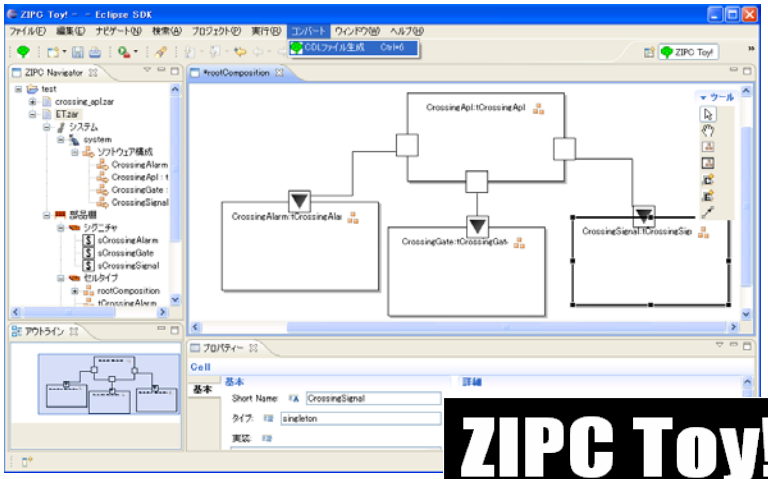
ZIPC
CASE Tool for Real Time System



ZIPC Toy! ：コンポーネント設計

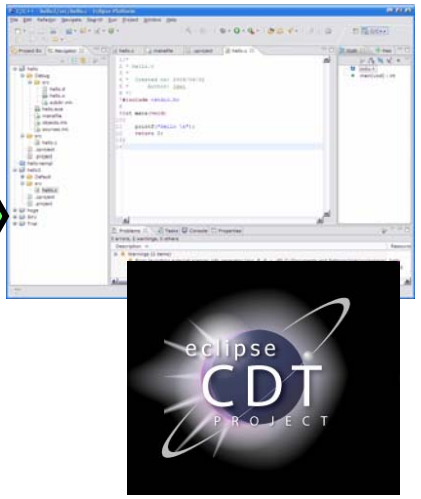
■構造設計/コンポーネント設計
　GUIで TECSコンポーネントモデルを作成
　プロパティ入力

■ 実装コードの開発環境を作成
　CDLからソースコードのひな形を作成（TECSGEN）
　コンパイル・ビルド環境を作成 （Eclipse CDT）

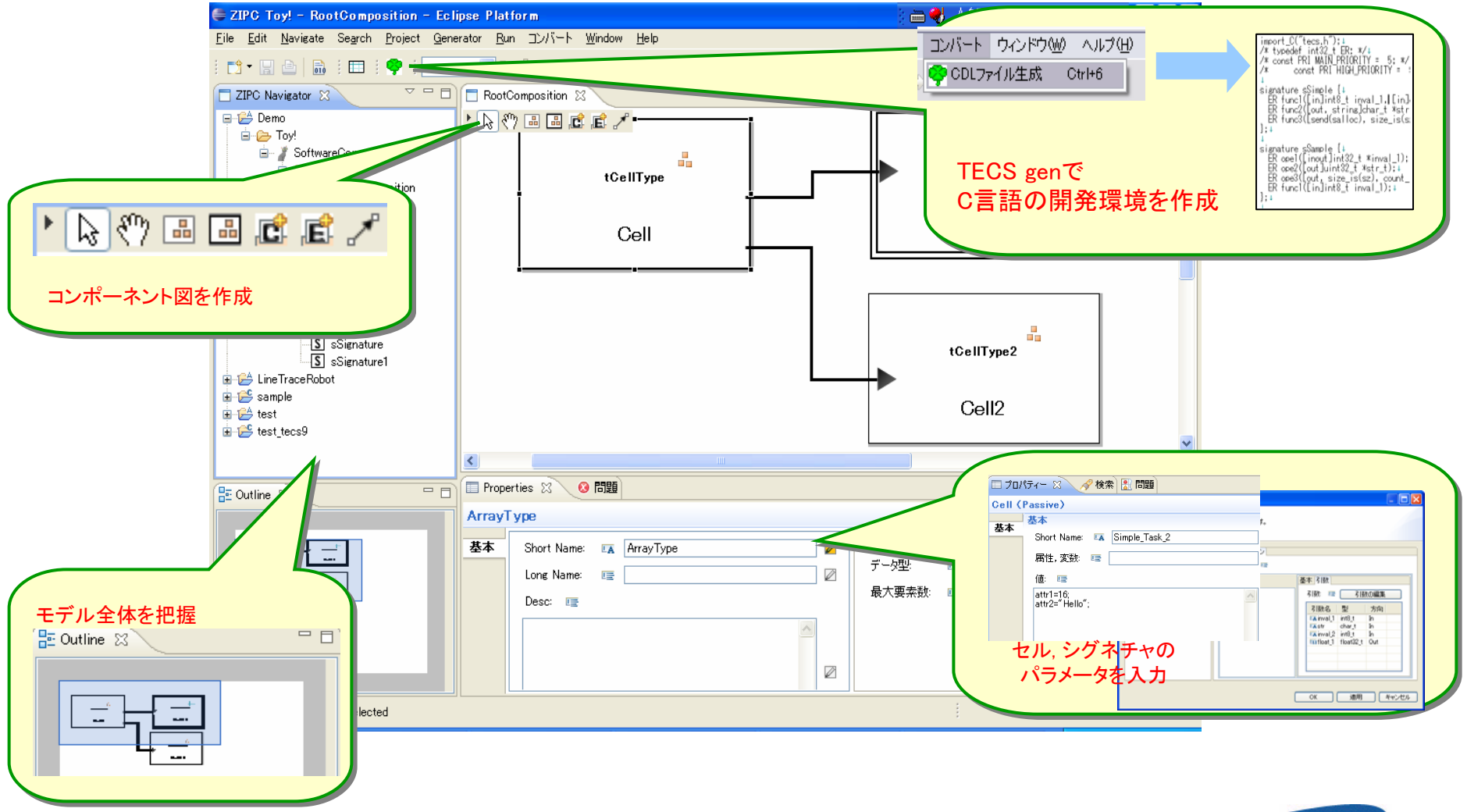


ZIPC Toy!
Technology of youth

```
[singleton]
celltype tCrossingSingnal {
  entry sCrossingSignal eCrossingSignal;
};

[singleton]
celltype tCrossingGate {
  entry sCrossingGate eCrossingGate;
};


cell tCrossingGate CrossingGate {

};

cell tCrossingSingnal CrossingSingnal {

};

cell tCrossingAlarm CrossingAlarm {

};

cell tCrossingApl CrossingApl {
  cCrossingAlarm = CrossingAlarm.eCrossingAlarm;
  cCrossingSignal = CrossingSingnal.eCrossingSignal;
  cCrossingGate = CrossingGate.eCrossingGate;

  priority_sysini = 6;
  stack_size_sysini = 1024;
```



ZIPC Toy! ：コンポーネント設計



ZIPC ® ：振る舞い設計

ZIPC ® ：状態遷移表をベースとした組み込み向けCASEツール

- 状態遷移表によるモレヌケのない設計
- シミュレーション機能
- Cソースコードの自動生成

状態

事象（イベント）

driver	初期状態	ジャイロ初期化	左モータ初期化	右モータ初期化	運転

TaskMain()	ジャイロ初期化
	cGyro_init();
		／	／	／	int8_t pwm_l, pwm_r;
					
					calc(VAR_speed,
					VAR_turn,
					cGyro_get(),
					VAR_gyro_offset,
					cMotorL_get_position(),
					cMotorR_get_position(),
					cBattery_get(),
					&pwm_l, &pwm_r);
					cMotorL_set_speed(pwm_l);
					cMotorR_set_speed(pwm_r);

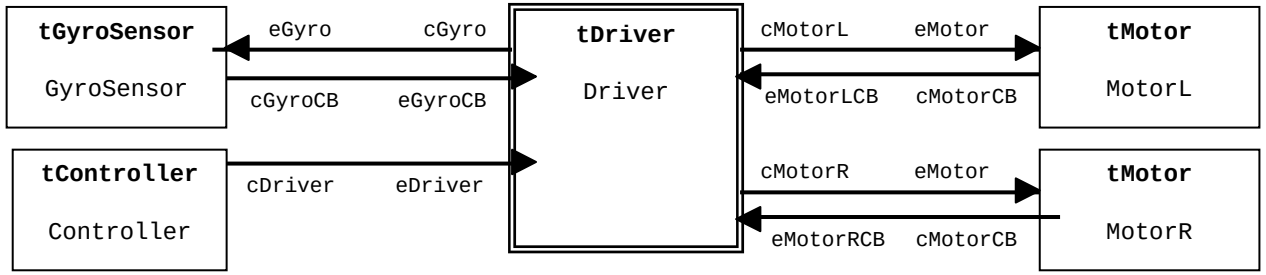
eGyro_init_fin(uint32_t offset)	×	左モータ初期化 VAR_gyro_offset = offset; cMotorL_init();	×	×	×
eMotorL_init_fin()	×	×	右モータ初期化 cMotorR_init();	×	×
eMotorR_init_fin()	×	×	×	運転 VAR_speed = 0; VAR_turn = 0;	×
eDriver_drive(float32_t speed, float32_t turn)	／	／	／	／	VAR_speed = speed; VAR_turn = turn;

アクション

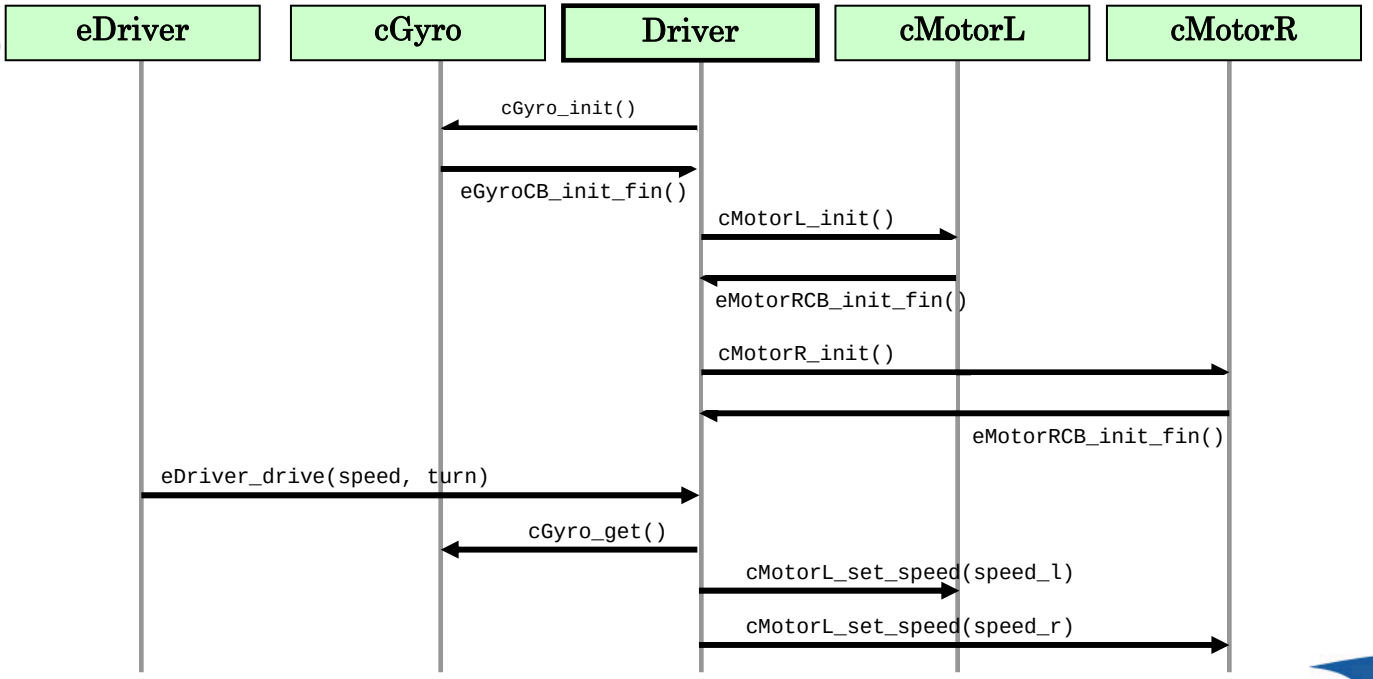
ZIPC CASE Tool for Real Time System

ZIPC Toy! + ZIPC ®

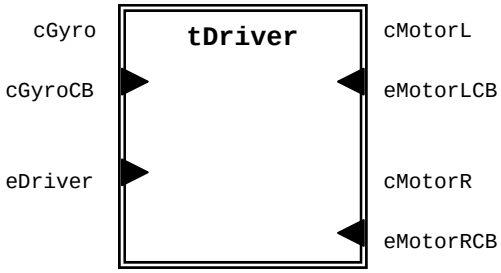
- TECSコンポーネント図



- 振る舞い（シーケンス図）



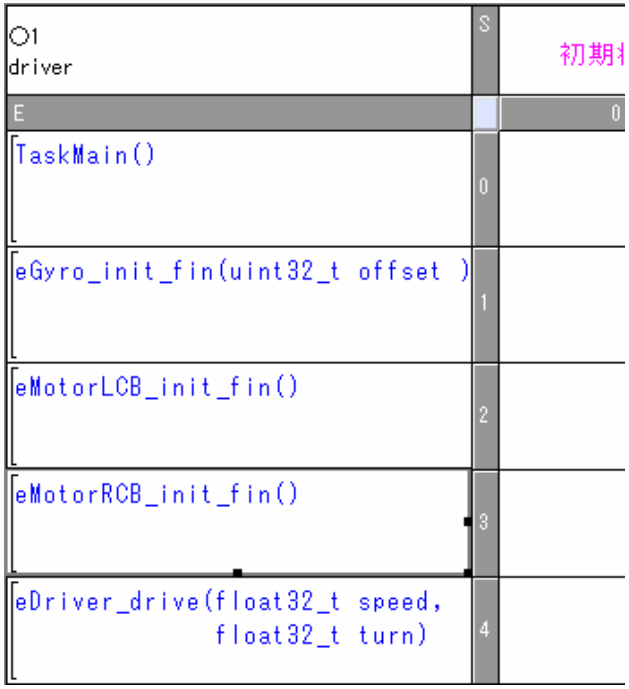
ZIPC Toy! + ZIPC ：（1）STMスケルトン作成




```
signature sGyroSensor {
  void init(void);
};
signature sGyroSensorCB {
  void init_fin([in]int_32_t offset);
};

signature sMotor {
  void init(void);
  int32_t get_position();
  void set_speed(int8_t pwm);
};

Signature sMotorCB {
  void init_fin(void);
};

signature sDriver {
  void drive([in]float32_t speed,
             [in]float32_t turn);
};
```

```
[singleton,active]celltype tDriver {
call  sGyroSensor cGyro;
  entry sGyroSensorCB eGyroCB;
  call  sMotor      cMotorL;
  entry sMotorCB    eMotorLCB;
  call  sMotor      cMotorR;
  entry sMotorCB    eMotorRCB;
  entry sDriver     eDriver;
  var {
    float32_t  speed;
    float32_t  turn;
    int32_t    gyro_offset;
    int8_t     state;
  };
};
```

受け口の関数



ZIPC Toy! + ZIPC ： (2) アクションの記述

```
cGyro_init()
cGyro_get()
cMotorL_init()
cMotorL_get_position()
cMotorL_set_speed(int8_t pwm)
cMotorR_init()
cMotorR_get_position();
cMotorR_set_speed(int8_t pwm)

VAR_speed
VAR_turn
VAR_gyro_offset
VAR_state
```

呼び口の関数

セル変数

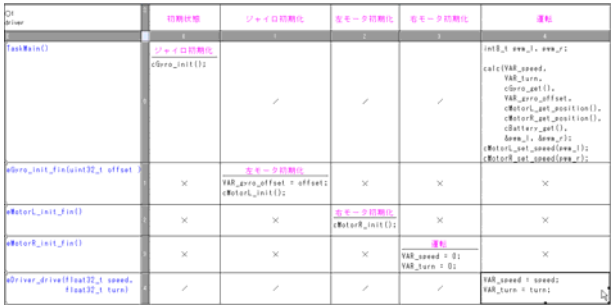
○1 driver	S	初期状態	ジャイロ初期化	左モータ初期化	右モータ初期化	運転
E		0	1	2	3	4

TaskMain()	0	ジャイロ初期化
		cGyro_init();
			／	／	／	int8_t pwm_l, pwm_r;
						
						calc(VAR_speed,
						VAR_turn,
						cGyro_get(),
						VAR_gyro_offset,
						cMotorL_get_position(),
						cMotorR_get_position(),
						cBattery_get(),
						&pwm_l, &pwm_r);
						cMotorL_set_speed(pwm_l);
						cMotorR_set_speed(pwm_r);

eGyro_init_fin(uint32_t offset)	1	×	左モータ初期化 VAR_gyro_offset = offset; cMotorL_init();	×	×	×
eMotorL_init_fin()	2	×	×	右モータ初期化 cMotorR_init();	×	×
eMotorR_init_fin()	3	×	×	×	運転 VAR_speed = 0; VAR_turn = 0;	×
eDriver_drive(float32_t speed, float32_t turn)	4	／	／	／	／	VAR_speed = speed; VAR_turn = turn;

コンポーネントのインタフェースに沿った実装が簡単に
状態遷移表によるヌケモレのない設計

ZIPC Toy! + ZIPC ® ：セルタイプコードの生成



STM 設計書

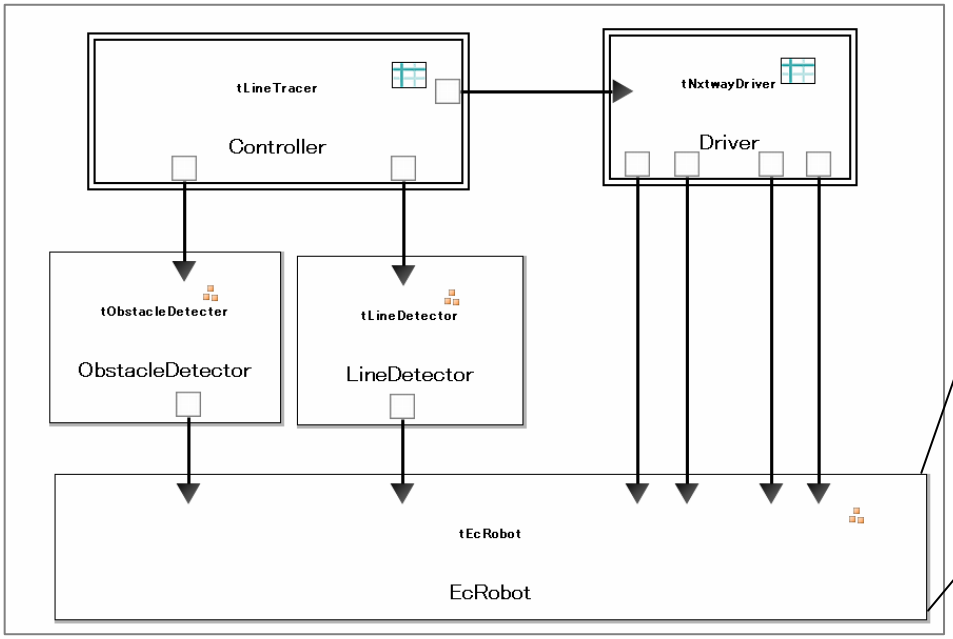
ZIPC
CASE Tool for Real Time System

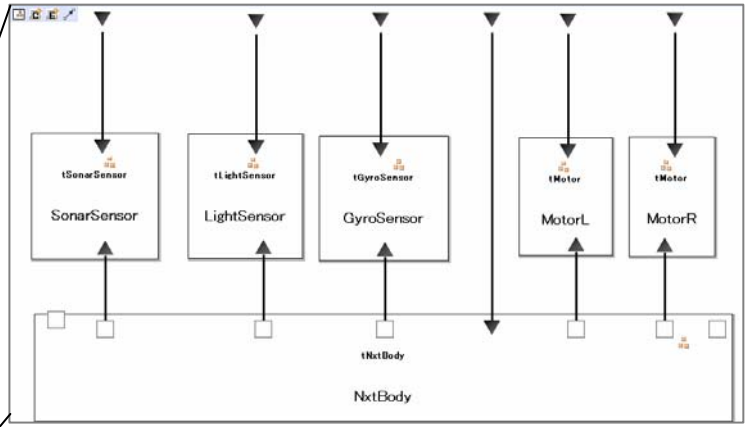
ジェネレータ

tDriver.c

```
void TaskMain( void )
{
    switch( ZComTsk_m1State[ZComTsk_M1] )
    {
        case ZComTsk_M1S0:
            ZComTsk_m1e0s0();
            break;
        case ZComTsk_M1S4:
            ZComTsk_m1e0s4();
            break;
        default:
            break;
    }
}

void eGyro_init_fin( uint32_t offset )
{
    switch( ZComTsk_m1State[ZComTsk_M1] )
    {
        case ZComTsk_M1S1:
            ZComTsk_m1e1s1( offset );
            break;
        default:
            break;
    }
}

static void ZComTsk_m1e0s4( void )
{
    int8_t pwm_l, pwm_r;
    calc(VAR_speed,
         VAR_turn,
         cGyro_get(),
         VAR_gyro_offset,
         cMotorL_get_position(),
         cMotorR_get_position(),
                  cBattery_get(),
                  &pwm_l, &pwm_r);
    cMotorL_set_speed(pwm_l);
    cMotorR_set_speed(pwm_r);
}
...
```

例： LEGO® Mindstorms NXT



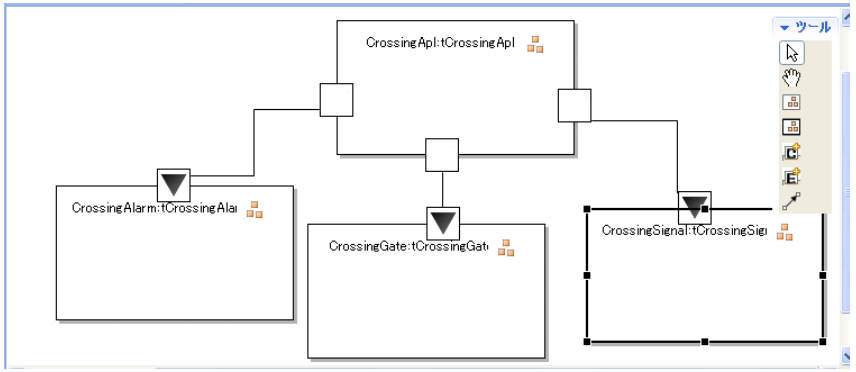


ZIPC Toy！：今後の拡張

■ CDLのインポート

```
[singleton]
celltype tCrossingSingnal {
  entry sCrossingSignal eCrossingSignal;
};

[singleton]
celltype tCrossingGate {
  entry sCrossingGate eCrossingGate;
};


cell tCrossingGate CrossingGate {

};

cell tCrossingSingnal CrossingSingnal {

};

cell tCrossingAlarm CrossingAlarm {

};

cell tCrossingApl CrossingApl {
  cCrossingAlarm = CrossingAlarm.eCrossingAlarm;
  cCrossingSignal = CrossingSingnal.eCrossingSignal;
  cCrossingGate = CrossingGate.eCrossingGate;

  priority_sysini = 6;
  stack_size_sysini = 1024;
```

ZIPC Toy!
Technology of youth



ZIPC Toy！：今後の拡張

■ 既存のCコードのコンポーネント化

```
#include “ecrobot.h”
#include “nxtlogo.h”

const int8_t MOTOR_L_PORT = PORT_B;
const int8_t MOTOR_R_PORT = PORT_C;
const int8_t GYRO_PORT = PORT_S1;

float32_t forward;
float32_t turn;
int32_t   gyro_offset;

void drive(float32_t forward, float32_t turn)
{
  int8_t pwm_l, pwm_r;
  balance_control(
    forward,
    turn,
    nxt_gyro_get(GYRO_PORT),
    gyro_offset,
    nxt_motor_get_position(MOTOR_L_PORT),
    nxt_motor_get_position(MOTOR_R_PORT),
    ecrobot_get_battery_voltage(),
    &pwm_l, &pwm_r
  );

  nxt_motor_set_speed(MOTOR_L_PORT, pwm_l, 0);
  nxt_motor_set_speed(MOTOR_R_PORT, pwm_r, 0);
}

void init(void)
{
  balance_init();
  nxt_motor_set_position(MOTOR_L_PORT, 0);
  nxt_motor_set_position(MOTOR_R_PORT, 0);

  gyro_offset=0;
}
```

自動抽出

関数定義

```
void drive(float32_t forward,
           float32_t turn);
void init();
```

関数参照

```
void balance_control(float32_t, float32_t,
…);
int32_t nxt_gyro_get(int8_t),
int32_t nxt_motor_get_position(int8_t);
int32_t ecrobot_get_battery_voltage(void);

void  nxt_motor_set_speed(int8_t, int8_t*,
int8_t)
void balance_init(void);
void nxt_motor_set_position(int8_t,int32_t);
```

ユーザが選択

- シグネチャ
- 受け口
- 呼び口

定数定義

```
float32_t forward;
float32_t turn;
int32_t   gyro_offset;
```

→ 属性

変数定義

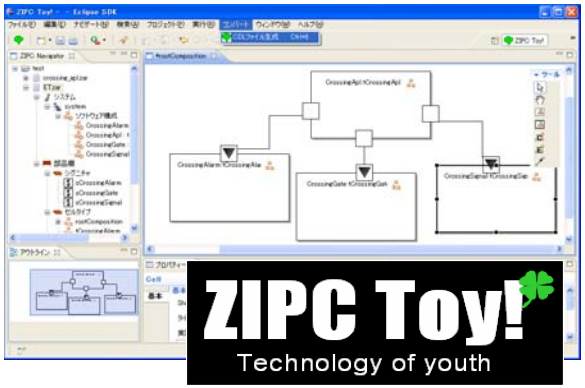
```
const int8_t MOTOR_L_PORT;
const int8_t MOTOR_R_PORT;
const int8_t GYRO_PORT;
```

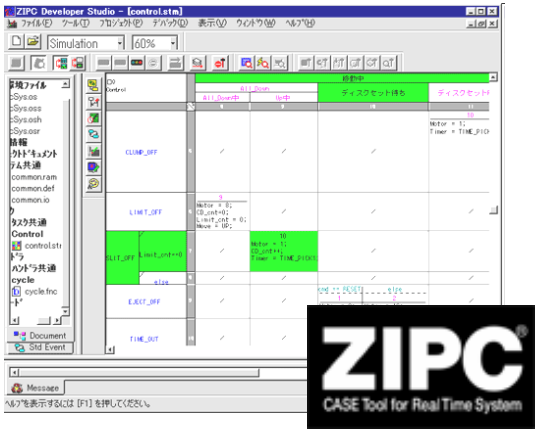
→ 変数

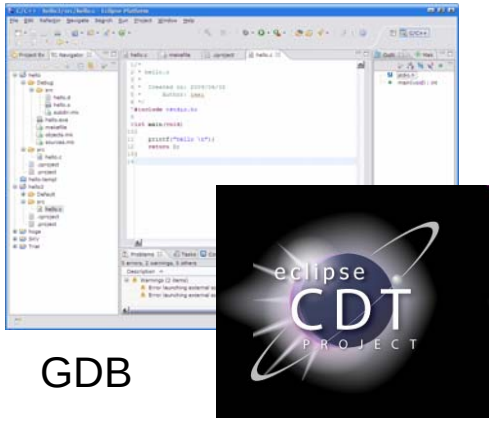
ZIPC Toy！: 今後の拡張

- デバッグ
  - ブレークポイントの設定
  - 変数の参照
  - 実行トレース

  …







GDB