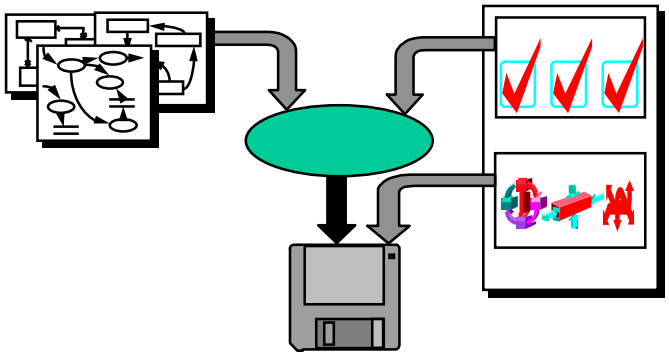
*ソフトウェア・アーキテクチャ*

*Project Technology 社　DesignPoint*

# MC-3020 の紹介

*オブジェクト指向を C 言語で実現した*
*ソフトウェア・アーキテクチャ*



2000 年 7 月 26 日

㈱東陽テクニカ

ソフトウェア・ソリューション
(http://www.toyo.co.jp/ss)



Copyright© 2000 ㈱東陽テクニカ

# 目次



## 図目次

# 1. MC3020 の概要

MC-3020 は、オブジェクト指向による分析モデルから、ANSI-C 言語に対応したソースコードを自動生成するアーキテクチャである。本書では、MC-3020 Ver.1.2 に対して(株)東陽テクニカが拡張をおこなった MC-3020.TOYO 1.9 について説明をする。

MC-3020 は二つの主要構成物 1)変換プログラム、2)ランタイム・ライブラリから構成される。変換プログラムは分析モデルをパースしてコードの変換を行う。一方、ランタイム・ライブラリは、選定された特別の実装環境（オペレーティング・システム、プログラム言語、プロセッサ等）でコンパイルされたコードを、正常に実行するために必要な各種機能を提供する。変換プログラムは、分析用 CASE ツールが作成した分析モデルを読み込み、パースしてソースコードを生成する。図 1-1 に、Project Technology 社の CASE ツールを使用した場合の構成要素間の相互関係を示す。

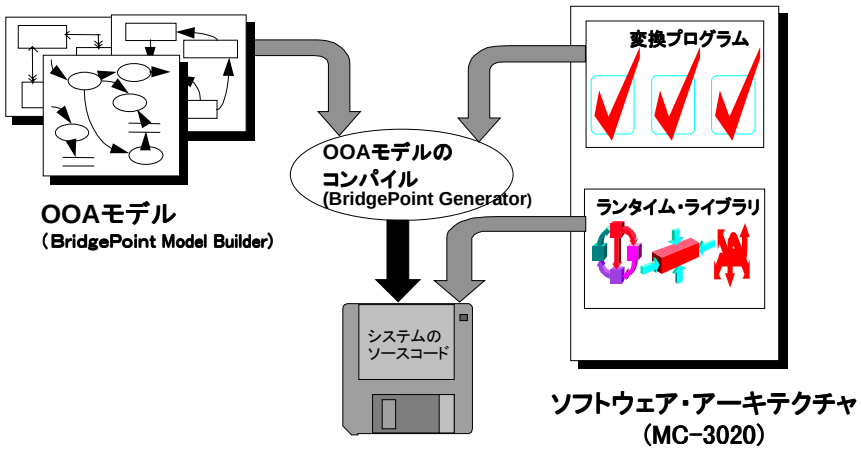



**図 1-1 MC-3020 の構成要素**

## 1.1. MC-3020 の構成物

MC3020 の構成物として、ソースコードの形態で提供されるものには、以下のものがある。

**●変換プログラム**

本構成物は、分析用 CASE ツールによって記述されたオブジェクト指向による分析モデルとブリッジの仕様定義を、変換エンジンの機能を使用して完全な C のソースコードに変換する。通常、変換エンジンが定義するアーキタイプ言語で記述された“構造アーキテイプ”と“アクション言語アーキタイプ”から構成される。

**●ランタイム・ライブラリ**

本構成物は、変換されたソースコードを実際に実行するために必要な付加的な機能を含む。オブジェクト指向分析の“メカニズム”とオペレーティング・システムに対するインターフェイスを含み、C 言語で記述されている。

**●ビルド・ツール**

本構成物は、モデルの変換、コードのコンパイル、システムの作成を自動化する機能を提供する。各プロジェクトで使用される開発環境や方針に適合するように変更することも可能である。

## 1.2.特徴

MC-3020 の特徴の概要を以下に示す。

### 1.2.1. C 言語によるコンパクトなソースコード生成

**●C 言語でのオブジェクト指向**

分析工程で識別されたクラスに関連した属性・振る舞い・処理を、C 言語の構造体とそれに関連した関数としてカプセル化する。このオブジェクト指向言語に相当する C 言語の特性を採用することで、変換の簡便さと設計時に発生するエラー要因を最小にすることを可能とする。

**●日本語分析モデルからの 100%コード生成**

BridgePoint Model Builder によって入力されたオブジェクト指向分析モデルは、完全なソースコードに変換される。分析モデル中で使用している日本語で、変数名

や関数名に相当する要素は、“日本語→英語変換テーブル”に定義されている英語名に変換しコード生成する。ユーザが英語名を記述していない場合は、任意のコンパイル可能な文字列に置き換える。

**●ANSI C 対応のソースコードの生成**

BridgePoint Model Builder で作成されたオブジェクト指向分析モデルは、ANSI C（C 言語）に対応したソースコードとして自動生成される。

**●8、16、32 ビット CPU への対応**

ターゲットシステムの RAM、ROM 容量が限定されたシステムへ対応可能なソースコードを生成する。

**●２パスコード生成による生成コードの最適化**

コード生成は２つのパスで構成されており、最初のパスでモデル構成とアクションを解析し、レポートを作成する。その解析結果を元に、次のパスで最適なコード生成を実施する。例えば、分析モデル上に最終的には全く使用されなかった不要な属性や関係が存在する場合は、それに関連するコードが生成されないように最適化される。また、使用するデータ領域のサイズ（イベントキューのサイズ等）も、できるだけ必要最低限を使用するようなソースコードを生成する。

### 1.2.2. 手書きコード（既存コード）との統合

オブジェクト指向分析モデルから自動生成したコードと手書きコード（既存コード、既存ライブラリ等）を容易に統合することができる。

**●分析モデルから手書きコードへのインタフェイス部分のスケルトンを自動性**

Bridge Point Model Builder で分析モデル中で使用する手書きコードへのインタフェイスを宣言することで、コンパイル可能なスケルトンを自動生成する。

**●ユーザ定義型のサポート**

手書きコードや、既存システムで定義しているユーザ定義型（C 言語の構造体）を自動生成コードと統合する機能を提供している。

**●ユーザ定義コードの実行**

ハードウェアやシステムに固有なユーザ定義処理（初期化処理等）や、障害回復

処理を容易にカスタマイズできるインターフェイス（コールアウト関数）を提供している。

### 1.2.3. コード生成オプションの指定

分析モデルに実装方法を記述するのではなく、通常のコンパイラオプションのようにコード生成時に“色付け”として、最適なコード生成オプションを指定することが可能である。

**●イベントの実装方法**

オブジェクト間のイベント送信（自分自身へのイベント送信も含む）を非同期イベントとしての実装するか同期イベントとしての実装するかをイベント単位で指定することが可能。非同期イベントは、送信元オブジェクトから受信先オブジェクトへのイベントとしてキューイングされる実装方法になる。同期イベントは、受信先オブジェクトのイベントを直接呼び出す実装方法となる。

**●イベントの優先順位付け**

イベントの実行時の優先順位を色付けとして指定することが可能である。

**●ROM/RAM への配置**

組み込みシステムのクラスの多くは単に参照されるだけで、動的に生成/削除されない。このようなリードオンリーのクラスを色付けで指定することで、色付けされたクラスのコードを ROM エリアに配置するコードを生成する。

**●オブジェクト・マッピング**

オブジェクト・マッピングが指定されたクラスは、クラス内の特定の属性を、既存のデータ領域の特定アドレスへ写像することが可能になる。

### 1.2.4. 組み込みシステムに最適なコード生成

**●メモリ管理**

動的なメモリ確保を実行するコードで問題となるようなヒープフラグメンテーションを防止するために、固定長のメモリ管理システムを使用するソースコードを自動生成する。メモリはオブジェクト単位に配列として確保され、この配列のサイズをユーザが指定することも可能である（指定しない場合は、システムのディフォルト使用する）。

**●割込みハンドラとタスクエントリ**

割込みハンドラとタスクエントリを非同期サービス用オブジェクト（非同期オブジェクトとして色付けをしたクラス）のアクションとして分析モデル内に記述して、自動生成することが可能である。但し、割込みハンドラやタスクエントリとして CPU や RTOS へ登録は、別途ユーザが実施する必要がある。

**●静的なインスタンス生成**

予め存在するインスタンスのデータを表形式で定義することで、静的な配列としてコードを生成する。

### 1.2.5.その他

**●自己診断**

デバッグ時に有効な情報として、ターゲットで実行中のシステムから現在実行しているアクション等の情報を、ログとして出力するための機構を提供する。

**●メトリックスとレポートの生成**

コード生成時に、分析モデルの構成要素（オブジェクト、関係、イベント等）に関するメトリックスとレポートをレポートディレクトリに生成する。

生成された情報は、最適なコードを生成したり、分析モデルに対する警告を発見するため等に使用される。

**●HTML 形式のドキュメントの生成**

分析モデル内に記述されているクラスの記述、関連記述、状態遷移表、アクション記述を HTML 形式のドキュメントとして自動生成する。

# 2. 生成コード概要

## 2.1.各クラスに対する生成コード

分析モデル中のクラスは、クラスのキー文字を接頭辞に持つ C 言語の構造体と関数群に変換される。インスタンスを保持する領域としては、クラス毎に予め用意されたプール領域（静的配列）に割り当てられる。プールの最大サイズはコード生成時に色付けとして指定できる。

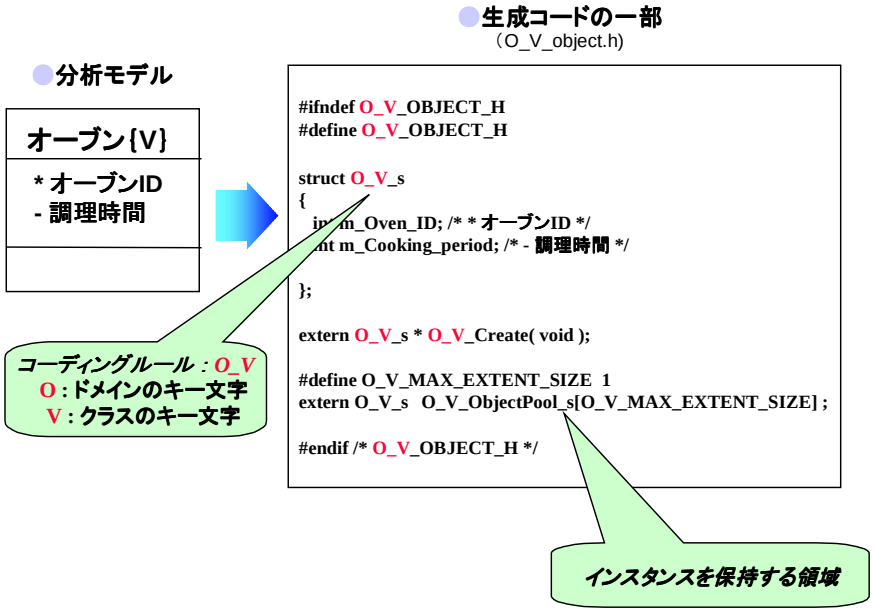



図 2-1 クラスに対する生成コード

クラスのインスタンスのデータを以下のように定義すると、配列の要素の値に初期値として設定される。


●生成コードの一部
(O_V_object.c )

```
O_V_s  O_V_ObjectPool_s [O_V_MAX_EXTENT_SIZE]  =
{
  /*  オーブンID, 調理時間  */
  { 1 , 5 } ,
  { 2 , 3 } ,
  { 3 , 10 }
} ;
```

**図 2-2 静的インスタンスに対するコード**

## 2.2.汎化（継承）に対する生成コード

オブジェクト指向の汎化（継承）は、スーパタイプ/サブタイプの構造体へのポインタとして実装される。

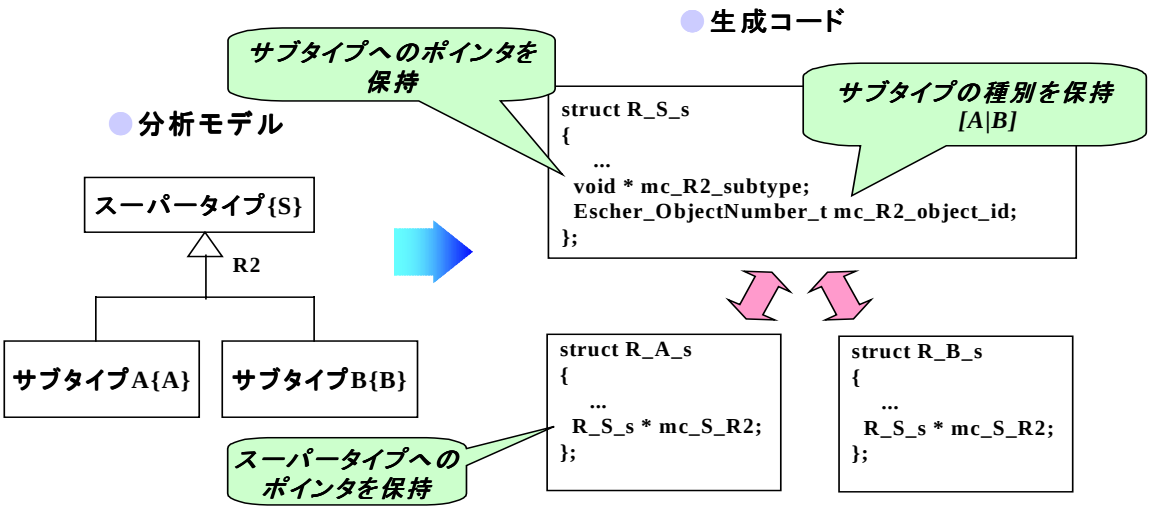



図 2-3 汎化（継承）に対するコード

## 2.3.状態図（ステートチャート）に対する生成コード

状態図を持つクラス（能動クラス）の場合は、関数ポインタを利用した状態遷移表と、各クラス単位のイベント・ディスパチャが生成される。各状態のアクションは、状態毎の関数として生成される。色付けによって、状態毎の関数にデバッグの際に有用なトレース情報を出力するソースコードを自動生成することも可能である。

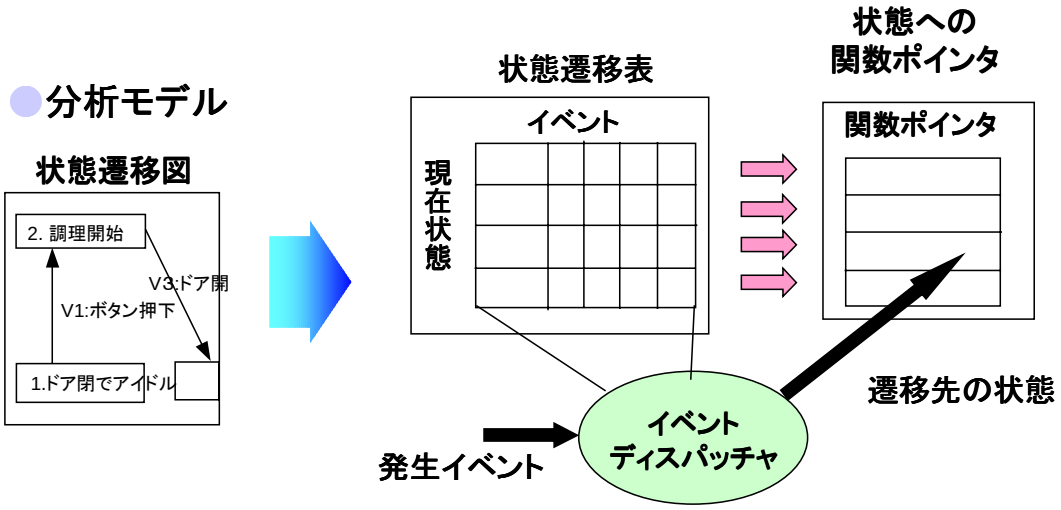



図 2-4 状態図からのコード生成のイメージ

### 2.3.1. 状態図から生成されるコードのサンプル

O_V_object.h

```
...

/*****************************************************************************
* Enumeration of state model states for object
*****************************************************************************/
#define O_V_STATE_1 1  /* State [1]: 'ドア閉でアイドル' */
#define O_V_STATE_2 2  /* State [2]: '調理開始' */
#define O_V_STATE_3 3  /* State [3]: '調理延長中' */
#define O_V_STATE_4 4  /* State [4]: '調理終了中' */
#define O_V_STATE_5 5  /* State [5]: 'ドア開でアイドル' */
#define O_V_STATE_6 6  /* State [6]: '調理中断中' */

/*****************************************************************************
 * Enumeration of state model event numbers
 *****************************************************************************/
#define O_V_EVENT_V1 0  /* 'ボタン押下' */
#define O_V_EVENT_V2 1  /* 'タイムアウト' */
#define O_V_EVENT_V3 2  /* 'ドア開' */
#define O_V_EVENT_V4 3  /* 'ドア閉' */
#define O_V_EVENT_V5 4  /* 'ブザー用タイマタイムアウト' */


/*****************************************************************************
 * State Action Methods
 *****************************************************************************/
extern void O_V_Action_1( O_V_s *, const OoaEvent_t * const );
extern void O_V_Action_2( O_V_s *, const OoaEvent_t * const );
extern void O_V_Action_3( O_V_s *, const OoaEvent_t * const );
extern void O_V_Action_4( O_V_s *, const OoaEvent_t * const );
extern void O_V_Action_5( O_V_s *, const OoaEvent_t * const );
extern void O_V_Action_6( O_V_s *, const OoaEvent_t * const );

extern void O_V_Dispatch( const OoaEvent_t * const );

...
```

O_V_object.c

状態遷移表
（現在状態とイベントの組み合わせで
次の状態を決定）

```
...

/*****************************************
 * State-Event Matrix (SEM)
 *****************************************/
static const Escher_StateNumber_t
O_V_StateEventMatrix[7][5] =
{
  /* Row 0: 'Unitialized' */
  { EVENT_CANT_HAPPEN, EVENT_CANT_HAPPEN, EVENT_CANT_HAPPEN, EVENT_CANT_HAPPEN, EVENT_CANT_HAPPEN },
  /* Row 1: O_V_STATE_1 'ドア閉でアイドル' */
  { O_V_STATE_2, EVENT_CANT_HAPPEN, O_V_STATE_5, EVENT_CANT_HAPPEN, EVENT_IS_IGNORED },
  /* Row 2: O_V_STATE_2 '調理開始' */
  { O_V_STATE_3, O_V_STATE_4, O_V_STATE_6, EVENT_CANT_HAPPEN, EVENT_IS_IGNORED },
  /* Row 3: O_V_STATE_3 '調理延長中' */
  { O_V_STATE_3, O_V_STATE_4, O_V_STATE_6, EVENT_CANT_HAPPEN, EVENT_IS_IGNORED },
  /* Row 4: O_V_STATE_4 '調理終了中' */
  { O_V_STATE_2, EVENT_CANT_HAPPEN, O_V_STATE_5, EVENT_CANT_HAPPEN, O_V_STATE_1 },
  /* Row 5: O_V_STATE_5 'ドア開でアイドル' */
  { EVENT_IS_IGNORED, EVENT_CANT_HAPPEN, EVENT_CANT_HAPPEN, O_V_STATE_1, EVENT_IS_IGNORED },
  /* Row 6: O_V_STATE_6 '調理中断中' */
  { EVENT_IS_IGNORED, EVENT_IS_IGNORED, EVENT_CANT_HAPPEN, O_V_STATE_1, EVENT_IS_IGNORED }
};

/*****************************************************************************
 * Array of pointers to the object's state action methods.
 *****************************************************************************/
static const StateAction_t
O_V_Actions[7] =
{
  0,
  O_V_Action_1,  /* 'ドア閉でアイドル' */
  O_V_Action_2,  /* '調理開始' */
  O_V_Action_3,  /* '調理延長中' */
  O_V_Action_4,  /* '調理終了中' */
  O_V_Action_5,  /* 'ドア開でアイドル' */
  O_V_Action_6   /* '調理中断中' */
};

/*****************************************************
 * O_V_Dispatch - State model level event dispatching.
 *****************************************************/
void
O_V_Dispatch( const OoaEvent_t * const event )
{
  O_V_s * instance = (O_V_s *)GetEventTargetInstance( event );
  Escher_EventNumber_t event_number = GetOoaEventNumber( event );
  Escher_StateNumber_t current_state;
  Escher_StateNumber_t next_state;
```

Vクラスの
イベントディスパチャ

```
  if ( instance )
  {
    current_state = instance->mc_current_state;
    if ( current_state > 6 )
    {
      /* Instance validation failure (deleted while event in flight) - TBD */
    }
    else
    {
      next_state = O_V_StateEventMatrix[ current_state ][ event_number ];
      if ( next_state <= 6 )
      {
        /* Execute the state action and update 'current state' */
        ( *O_V_Actions[ next_state ] )( instance, event );
        instance->mc_current_state = next_state;
      }
      else
      {
        /*  省略  */
      }
    }
  }
}
```

次の状態のアクションに対応
している関数の呼び出し

O_V_action.c

「調理開始」のアクションが生成される。

```
...

/*****************************************************************************
 * State [2]: '調理開始'
 *****************************************************************************/
void
O_V_Action_2( O_V_s * self, const OoaEvent_t * const event )
{
  O_V_Event2_s * v1;  /* イベント */
  O_L_s * v4;  /* ライト */
  O_T_s * v8;  /* チューブ */
  O_M_s * v12;  /* モニタ */
  ROX_STATE_TXN_START_TRACE( "V", 2, "調理開始" )

  /* CREATE EVENT INSTANCE イベント V2:'タイムアウト'() TO SELF */
  v1 = New_O_V_Event2_s( self );

  /* ASSIGN SELF.タイマ ID = TIM::timer_start(microseconds:SELF.調理時間, event_inst:イベント) */
  self->m_timer_id = TIM_timer_start( (OoaEvent_t *)v1, self->m_cooking_period );

  /* SELECT ONE ライト RELATED BY SELF->L[R2] */
  v4 = self->mc_L_R2;

  /* GENERATE L1:'ライト点灯'() TO ライト */
  {
    O_L_Event1_s * event5 = New_O_L_Event1_s( v4 );
    Escher_SendEvent( (OoaEvent_t *)event5 );
  }

  /* SELECT ONE チューブ RELATED BY SELF->T[R7] */
  v8 = self->mc_T_R7;

  /* GENERATE T1:'調理開始'() TO チューブ */
  {
    O_T_Event1_s * event9 = New_O_T_Event1_s( v8 );
    Escher_SendEvent( (OoaEvent_t *)event9 );
  }

  /* SELECT ONE モニタ RELATED BY SELF->M[R3] */
  v12 = self->mc_M_R3;

  /* GENERATE M1:'モニタ開始'() TO モニタ */
  {
    O_M_Event1_s * event13 = New_O_M_Event1_s( v12 );
    Escher_SendEvent( (OoaEvent_t *)event13 );
  }

  /* BRIDGE PIO::ブザーオフ() */
  PIO_buzzer_off();

  ROX_STATE_TXN_END_TRACE( "V", 2, "調理開始" )

}
```

トレース情報を出力する色付けをした場合に挿入される

分析モデルの中のアクション記述はコメントになる。

# 付録　A　サンプルモデルと自動生成コード

## 付録 A-1．サンプルモデル仕様

【簡易オーブン　要求仕様 】

小型で安価な電子レンジの制御をするものとする。この製品のコンセプトは次のようなものである。

1. レンジにはボタンが１つ付いており、ユーザがそのボタンを押すことができる。レンジの扉が閉じているときにボタンを押せば、レンジは１分間調理を行う。調理をするとは、すなわち、パワーチューブに電力を供給するということである。
2. 調理中にボタンを押すと、調理時間が１分間延長される。例えば、調理時間が４０秒残っている時に２度ボタンを押せば、２分４０秒の調理時間がセットされる。
3. 扉が開いている時にボタンを押しても何もおこならい。
4. レンジの中にライトがあり、調理中は常にライトをつけておく。調理中以外は、ライトを消しておくが、扉が開いている時はライトをつけておく。
5. 調理中に、扉が開かれれば調理を中断する。
6. 調理が終了したら、終了したことを知らせるために音を鳴らす。

（ボタンが一回押された時の調理時間は、現在は１分であるが、後に変更するかもしれない。）

## 付録 A-2．　分析モデルの紹介

付録として、上記サンプル仕様の分析モデルを Bridge Point Model Builder で入力した例を添付する（抜粋）。添付したモデルは以下の通り。

- クラス図
- オーブンの状態図
- ライトの状態図
- 割込みハンドラ

## 付録 A-3. 添付ファイル一覧

Bridge Point Model Builder で作成した分析モデルから MC-3020 を使用して生成したソースコードの一部を添付する。本ソースコードは、日立製 H8 で実行可能なものである。

- pt_transTable.c　　：　　日本語→英語変換テーブル
  分析モデルの中で使用している日本語をリストしたテーブルがコード生成時に作成される。最初に作成される時は、英語名としては任意の文字列(JP_1 等)が割り当てられる。任意の文字列を適当な英語名に直した後、コード生成をすると、英語名を使用したソースコードが生成される。

- O_V_object.h　　：　本文中で説明に用いたソースコード
- O_V_object.c　　：　本文中で説明に用いたソースコード
- O_V_actions.　　：　本文中で説明に用いたソースコード

- O_L_actions.c　　：　外部へのインタフェイスを呼び出している例

- PIO_bridge.h　　：　外部インタフェイスの実装例
- PIO_bridge.c　　：　外部インタフェイスの実装例

- O_dom_async　　：　割込みハンドラの自動生成例