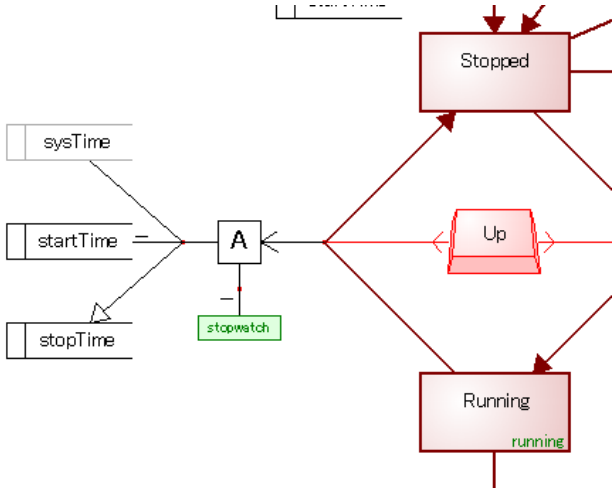
MetaEdit+　ジェネレータの動作について

MetaEdit+のソースコード（ドキュメント）ジェネレータは基本的に、独自のスクリプトを使ってモデル内を巡回し、見つかったメタモデルのデータ（プロパティー）に応じて、プログラム言語を構成するための文字列を出力する事で実現しています。これが、出力するプログラム言語を選ばず、要求仕様やデザイン仕様すら出力できる理由です。決して、メタモデルに適当なプログラムソースが書き込まれていて、メタモデルの順番に合わせてそれをつなげている訳ではありません。

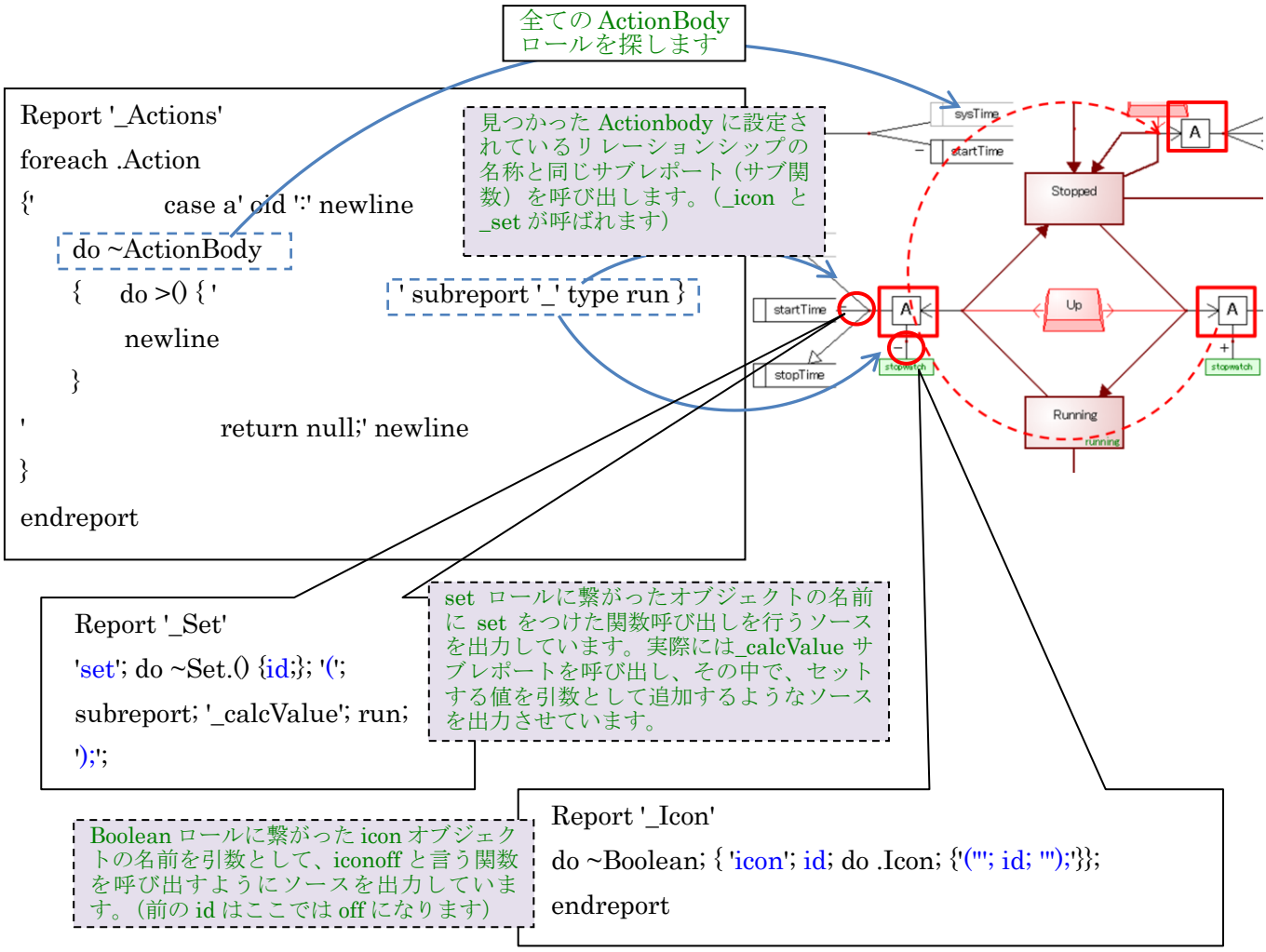
また、巡回方法は、“全てのメタモデルを順に見つける“、”特定のメタモデルに注目しそのメタモデルに接続されている全てのメタモデルに順に注目してゆく（これを繰り返す）“という方法があります。
IF を使ってメタモデルのタイプやプロパティーの値に応じて処理を変更する（出力するソースを変更する）事も可能です。また、サブ関数のようなものを使用して、それを呼び出すことも可能です。
文字列の出力先は、任意の名前のファイル（.c、.h、.cpp、.java 等）や MetaEdit+の標準出力が選択可能です。
以上の内容を MetaEdit+のコード生成の基本動作として、以下フレームワークの呼び出し方法と＃IFDEF に相当する部分の記述方法について説明します。

尚、この資料ではフレームワークの詳細な内容に関しては詳しく述べていません。別途用意した資料“ドメインフレームワークについて”に詳しく解説されています。併せてお読み頂く事でより深い理解が得られます。

フレームワークの呼び出しを記述するには、あるメタモデルがフレームワークに結びついていた場合に、そのプロパティーを引数にして、フレームワークへの呼び出し関数を記述する方法が一般的です。



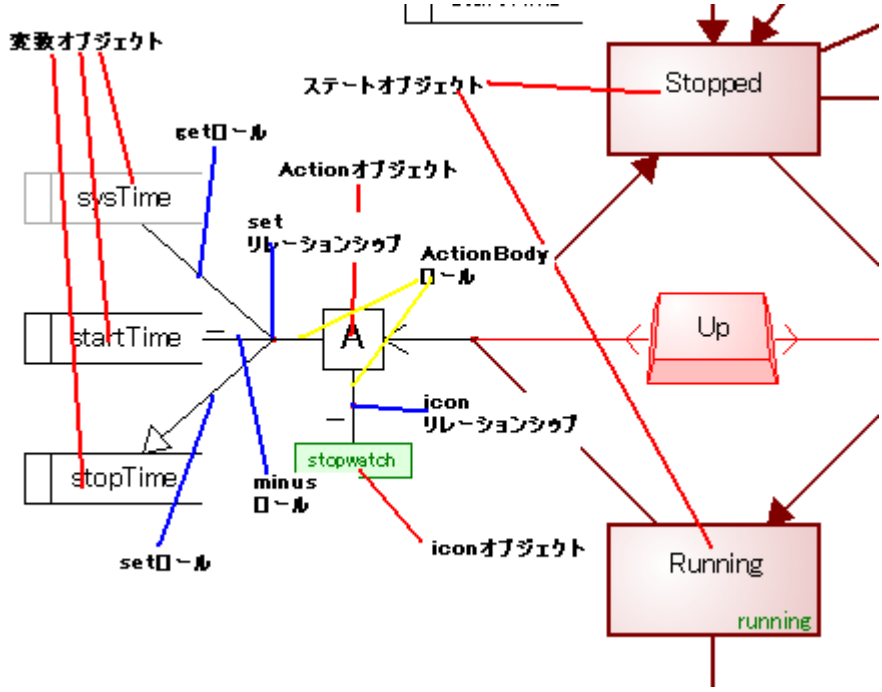
このモデルは、Up ボタンが押されたときに、Running ステートから Stopped ステートに移動し、同時に stopTime 変数に、sysTime 変数から startTime 変数を引き算した値を代入し、stopwatch アイコンを非表示にするという動作を表現したものです。（デジタル時計のストップウォッチ機能をモデル化しています）
このモデルに対応するジェネレータの定義は次ページの様になっています。
スクリプト中の青い文字が実際に出力される文字列（ソースコード）です。各オブジェクト/ロール/リレーションシップの名前に関しては、スクリプトの次のページに挿入した図に示します



_calcValue では上で作成した関数呼び出しを終了する為の処理を出力しています。

```
subreport; '_calcValue'; run;
');';
endreport

Report '_calcValue'
do ~Get.()
{       'get' id '()'
}
do ~(Minus | Plus)
{       '.me' type
        do .()
        {       '(get' id '())'
        }
}
endreport
```



生成されたソースコードは以下の様になります。

```
case a22_4857:
        setstopTime(getsysTime().meMinus(getstartTime()));
        iconOff("stopwatch");
        return null;
case d22_4302:
```

フレームワークが独立したプログラムとして動作しており、そのプログラムにメッセージを送る事で、フレームワークを使用するような場合には、メッセージを送る部分を記述させるようにジェネレータを作成することになります。

また、プログラム実行時に初期化処理などあらかじめフレームワークの実行が必要な場合は、特定のモデル図が使用しているメタモデルを全て検索するように設定し、それら必要な初期化コードを生成させることができます。以下に、その仕組みの具体例を示します。

具体例

```
public class '; id; ' extends AbstractWatchApplication {
';
subreport; '_Variables'; run;
'           public '; id; '(Master master) {
                    super(master, '"; oid; '");'; newline;
subreport; '_StateData'; run; newline;
subreport; '_TransitionData'; run; newline;
subreport; '_Decompositions'; run; newline;
subreport; '_StateDisplayData'; run;
'           };' newline;

newline; newline;

'           public Object perform(int methodId)
            {
                    switch (methodId) {
';
subreport; '_Actions'; run;
subreport; '_DisplayFns'; run;
'                   }
                    return null;
            }'; newline;

'}';
```

この部分で、フレームワークの初期化をさせている
全てのメタモデルオブジェクトを検索して処理を記述している

実際の動作に対する処理はこの部分で出力させている
メタモデルの接続応じて処理を行わせている
実際には
subreport '_Actions' run で上のソースの処理に移行

ちなみに、#IFDEF に関しては、#IFDEF を使ったバリエーションの影響を受けないようなメタモデル（全てのケースを包括するようなメタモデル環境）をつくり、#IFDEF のバリエーションを、モデル環境そのものに設定し、出力するソースを変更する方法で対応可能です。

具体的には、例えば#IFDEF を使って、複数のグレードや仕向地に合わせたプログラムを作っているような場合は、先ず全てのグレードと仕向地に対して適用できる様なメタモデルを作成します。モデル（図）そのものにプロパティーを設定することができるので、そこにグレードや仕向地等のプロパティーを設定しておき、そのプロパティーによって、出力するソースを変える（IF 文を使って出力する文字列を変更する）と言う方法で対処できます。

全てのグレードや仕向地に合わせたメタモデルが作成できず、バリエーションによって完全にモデルの書き方が変わるような場合は、そのバリエーションに応じたモデルを作成することになるでしょう。（なるべく、そうならない様にメタモデルを作成することが最良です）

FUJI SETSUBI
富士設備工業株式会社 電子機器事業部
〒591-8025 大阪府堺市北区長曽根町1928-1
Tel: 072-252-2128 www.fuji-setsu.co.jp