Code generation from State Machine diagram Feature Guide

by SparxSystems Japan

ステートマシン図からのコード生成 機能ガイド

(2016/10/07 最終更新)

1 はじめに

このドキュメントでは、Enterprise Architect Suite を利用してステートマシン図からソースコード生成を行うための手順を紹介しています。

Enterprise Architect Suite を利用してステートマシン図からのコード生成ができる言語は、以下の通りです。

- Enterprise Architect Suite ビジネスモデリング版
  →C,C++,Java,C#,VB.NET
- Enterprise Architect Suite システムエンジニアリング版・アルティメット版
  →C,C++,Java,C#,VB.NET および SystemC,VHDL,Verilog

なお、C,C++,Java,C#,VB.NET については、Enterprise Architect Suite であれば、すべてのエディションでアクティビティ図およびシーケンス図からのコード生成も可能です。これらの図からのコード生成についてはドキュメント「アクティビティ図・シーケンス図からのコード生成 機能ガイド」をご覧ください。

また、ドキュメント「ソースコードの読み込みと生成 機能ガイド」に掲載している内容については説明を省略しますので、このドキュメントとあわせて「ソースコードの読み込みと生成 機能ガイド」もご覧ください。

2 ステートマシン図からのソースコード生成における注意

Enterprise Architect のステートマシン図からのソースコード生成機能で、何もカスタマイズしない場合の生成結果は、特定の環境や OS 等を考慮していないものです。このステートマシン図からのソースコード生成では、それぞれの現場・環境などに応じて、テンプレートをカスタマイズして利用することを前提としています。実開発で適用する場合、どのようなステートマシン図を作成し、その結果としてどのようなソースコードになるのかをまず検討し、その結果にあうように、それぞれの現場ごとにカスタマイズすることになります。

C 言語および C++言語では、ステートマシン図からのソースコード生成のカスタマイズ例として、以下のページで情報を公開しています。

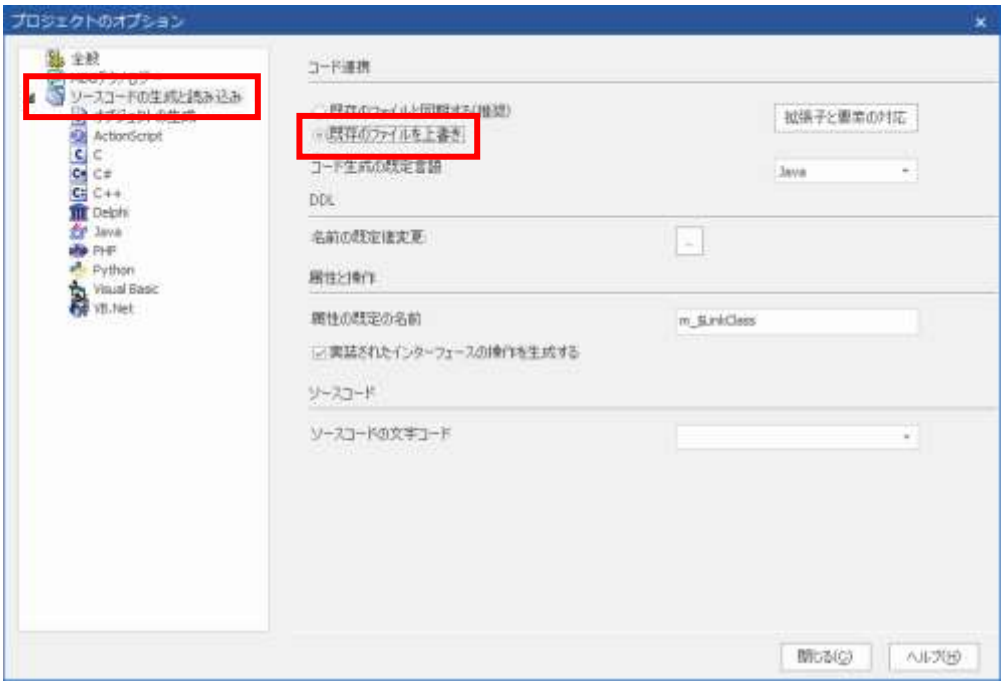
C 言語:
https://www.sparxsystems.jp/products/EA/tech/GenerateStateMachine.htm

C++言語:
https://www.sparxsystems.jp/products/EA/tech/GenerateStateMachineCPP.htm

このカスタマイズは、日本のお客様からの要望を元に、既定の出力内容とは異なる内容が出力されるよう、カスタマイズされています。
これらのテンプレートを利用している場合には、このドキュメントの内容と異なる出力結果になります。また、トリガについての扱いが、このドキュメントでの説明とは異なります。(外部からトリガ(イベント)を与える方式になります。)

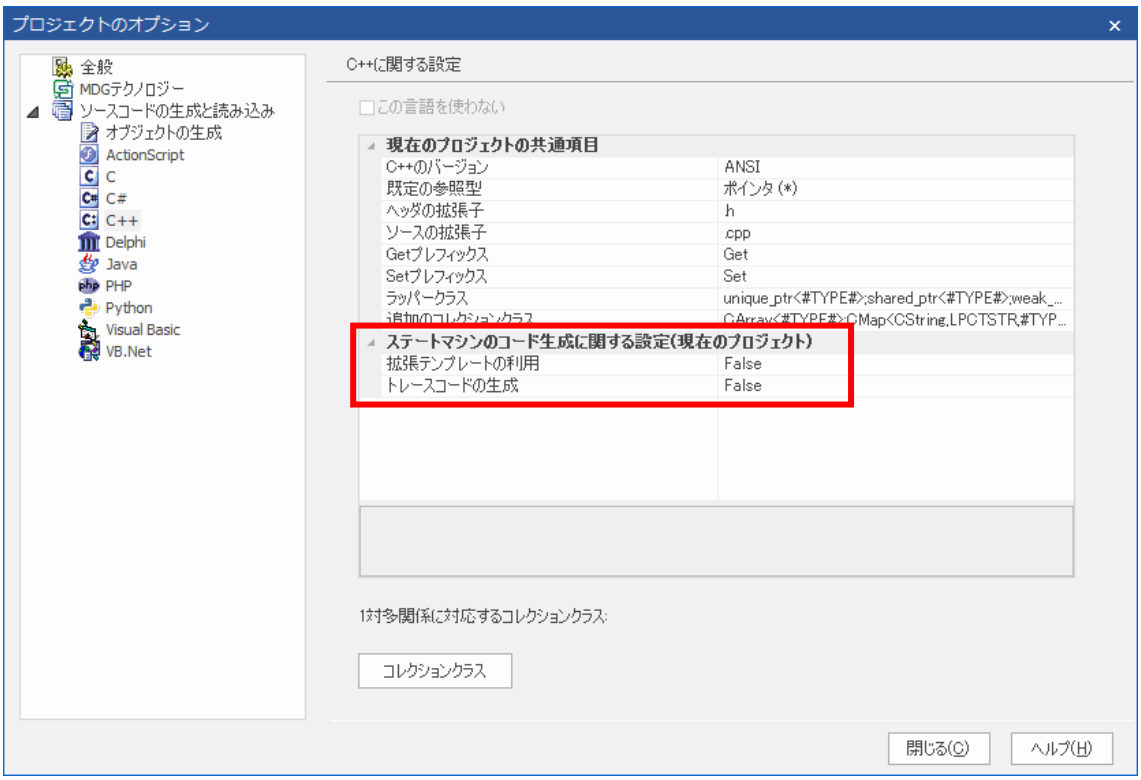
ただし、クラス要素と状態マシン要素との関係など、多くの内容はこのドキュメントと共通です。上記の URL で紹介しているテンプレートを利用する場合やカスタマイズする場合も、このドキュメントの情報が参考になります。

なお、ステートマシン図等の振る舞い図からのコード生成機能を利用する場合には、ソースコードの生成において「常に上書き」モードにする必要があります。「ホーム」リボン内の「オプション」パネルにある「プロジェクト」を実行してオプション画面を開き、「ソースコードの生成と読み込み」グループにある「既存のファイルを上書き」を設定して下さい。この設定が行われていない場合には、既存のソースファイルがある場合には、ステートマシン図の内容は更新(上書き)されません。



また、この結果、常にクラス要素からソースコード生成の片方向のみとなり、生成されたソースコードを編集することはできません。ソースコードを編集した場合には、次回クラス要素からソースコードを生成するときに、その内容は上書きされます。
(関係する内容として、このドキュメントの第 10 章もご覧下さい。)

さらに、バージョン 11.0 からは「exe 実行の状態マシン」機能が追加されています。この機能は、生成されるソースコードに特別な処理を追加し、実行時のプログラム内部の状態遷移をモデル内に表現することができます。この「exe 実行の状態マシン」機能に関するソースコード生成結果にならないよう、利用する言語ごとのオプション設定で「拡張テンプレートの利用」と「トレースコードの生成」を「False」に設定してください。

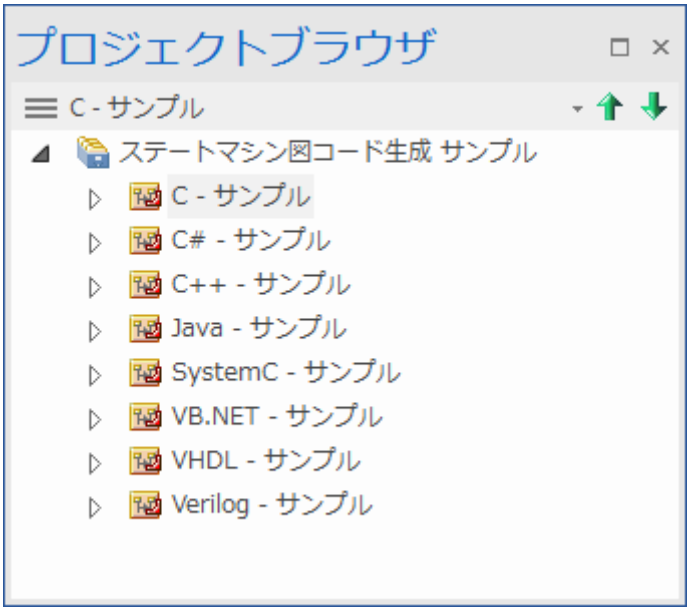


3　サンプルを利用してコード生成までの手順を確認する

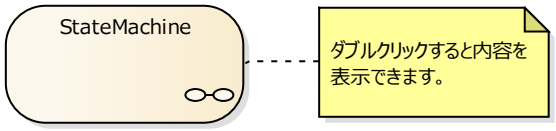
このソースコードの生成の作業の流れを理解するための一番良い方法は、サンプルを利用することです。この章では、サンプルからソースコードを生成するまでの手順を紹介します。

サンプルファイルは、Enterprise Architect のインストールディレクトリにある「StateMachine_Sample.eap」です。しかし、通常はインストールディレクトリは Windows の制限で内容の編集ができないため、このサンプルファイルを Windows のデスクトップなど、編集可能なディレクトリにコピーしてください。その後、サンプルファイルをダブルクリックして開きます。

プロジェクトブラウザには、以下の画像のようにいくつかの言語でのサンプルモデルが含まれていますので、希望するサンプルのパッケージを展開し、クラス図を開いてください。

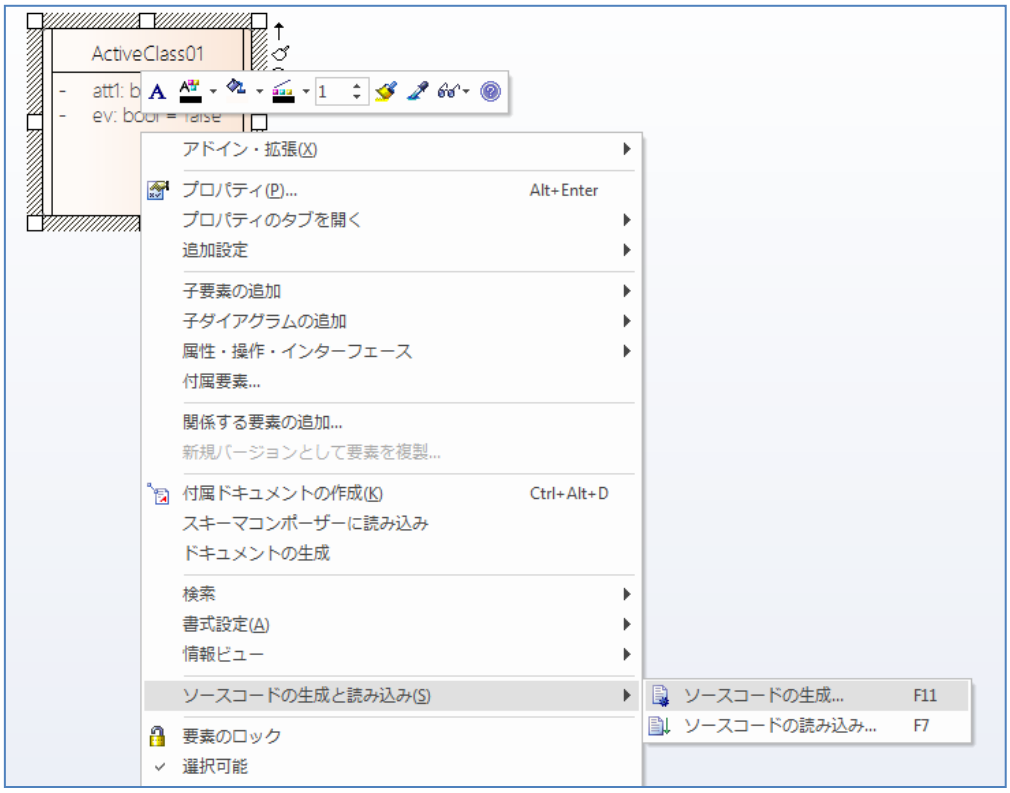


クラス図にはクラス要素が 1 つ配置されています。この要素をダブルクリックすると、このクラス要素が持つ状態マシン要素が配置された図が開きます。

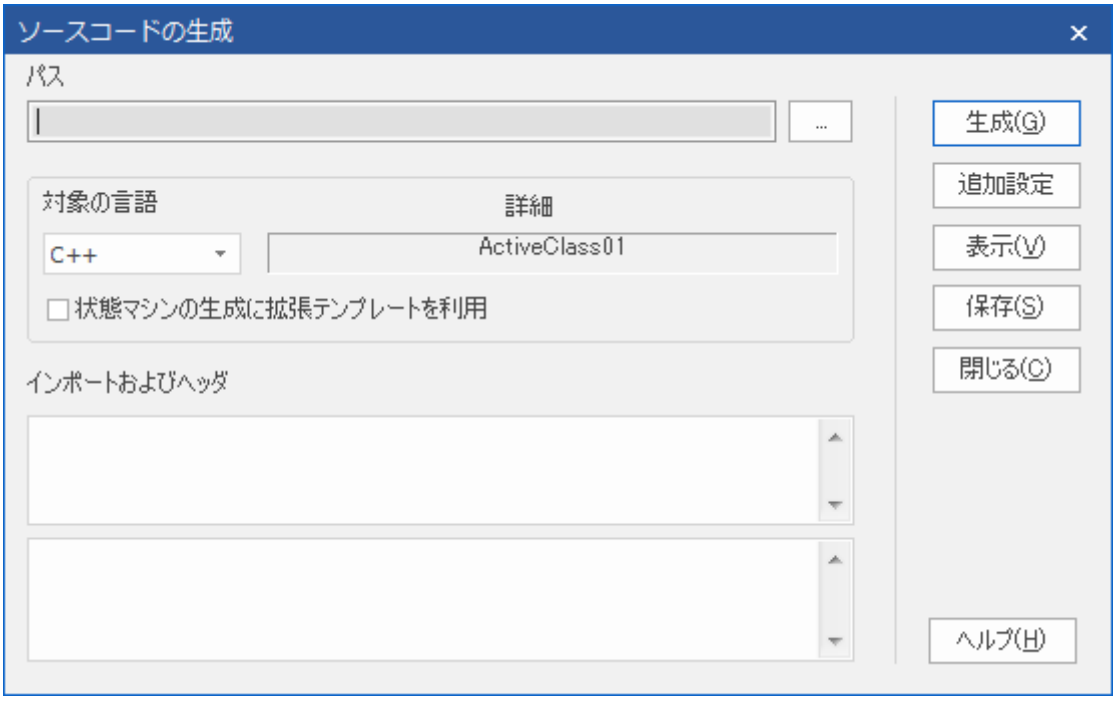


この状態マシン要素「StateMachine」をダブルクリックすると、振る舞いを記述したステートマシン図が開き、定義内容を確認できます。

ソースコードを生成するには、クラス図に配置されているクラス要素を右クリックしてください。コンテキストメニューから「ソースコードの生成と読み込み」→「ソースコードの生成」を選択してください。下の画像は、C++のサンプルのクラス図の例です。



次のような画面が表示されます。



この画面で、出力先となるパスおよびファイル名を指定します。また、「状態マシンの生成に拡張テンプレートを利用」にチェックが入っている場合には、チェックを外してください。

その後、「生成」ボタンを押すとソースコードが生成されます。生成したソースコードの内容を確認するには、クラスを右クリックして「ソースコードの生成と読み込み」→「ソースコードの表示」を実行してください。Enterprise Architect 内部のエディタで、生成したソースコードを確認することができます。

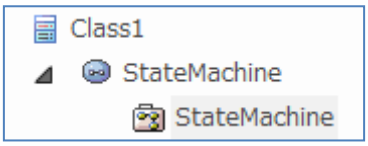
なお、ステートマシン図からソースコード生成を行う場合に既にソースファイルが存在する場合、必ず「上書き」にする必要があります。既存のソースコードに対して、内容を同期させることはできません。
(クラス要素ごとにこの設定を行う場合には、上記の画面で「拡張設定」ボタンを押し、表示される画面の中央上部にある「既存のファイルを上書き」を選択してください。サンプルデータでは設定の必要はありません。)

また、既存のソースコードからステートマシン図を生成(リバース)したり、ソースコードを編集した結果からモデルを同期更新したりすることもできません。

以上が、ソースコードを生成するまでの流れになります。

4 モデルの構成

ステートマシン図からソースコード生成を行うモデルは、必ず特定の構成になっていなければなりません。特定の構成とは、ソースコード生成を行うクラス要素の子要素として「状態マシン」要素が 1 つ存在し、その状態マシン要素の子ダイアグラムとしてステートマシン図が 1 つ存在することです。



既存のクラスに対して構成を新規に追加する場合には、ダイアグラム内で対象のクラスを右クリックし、「子ダイアグラムの追加」→「状態マシン」を実行します。あるいは、4 章で紹介したサンプルのパッケージをコピーしてから内容を編集することも可能です。

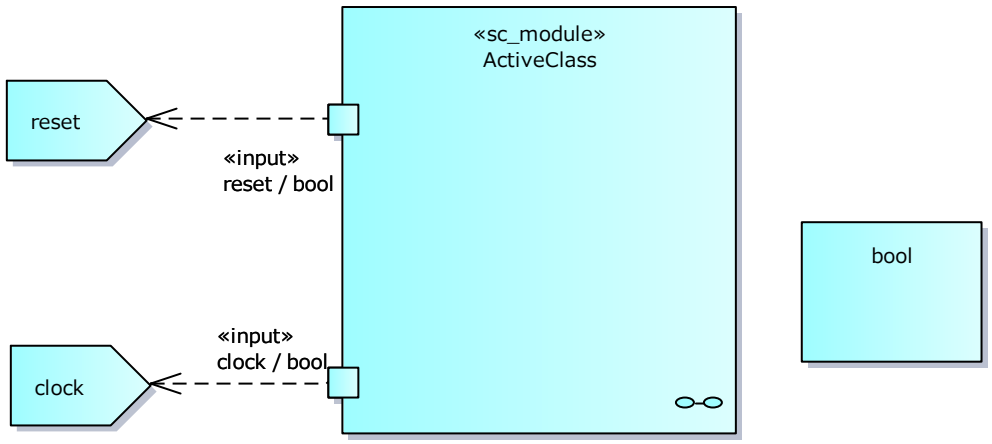
なお、1 つのクラスに対して複数の状態マシン要素は定義できません。また、1 つの状態

マシン要素に対して複数の開始要素を配置することはできません。ただし、ステートマシン図を階層構造にすることは可能です。状態要素を右クリックして「子ダイアグラムの追加」→「子ダイアグラムを作成」を選択することで、階層的なステートマシン図を作成することができます。

そのほか、必要に応じてイベントを示す「トリガ」を追加します。ハードウェア言語の場合には、アクティブクラスに受信するイベントを示す「ポート」も必要です。この詳細は後ほど説明します。

5 クラス図の構成

ステートマシン図からのコード生成を実現するクラス図は、以下の要素から構成されます。

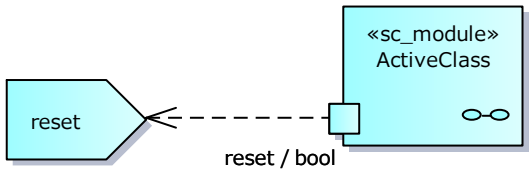


- アクティブクラス
- トリガ*
- ポート*
- ポートとトリガを結ぶ「依存」*
- ポートの型を定義するクラス**

このうち、*の項目は、ハードウェア系の言語(SystemC/VHDL/Verilog) でのみ必要です。C/C++などのソフトウェア系の言語は、定義は不要です。しかし、これらのソフトウェア系の言語では、状態の遷移を発生させるイベントの発生源がありませんので、Enterprise Architect で生成したソースコードだけでは、何も動作しません。外部から状態の変化を起

こす「トリガ」を投げるためのプログラムが別途必要になります。
**の項目は、必要に応じて追加して下さい。基本的には、利用する型はこのクラス図を含むパッケージ内で定義されていなければなりません。

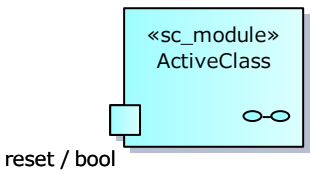
ハードウェア系言語の場合には、外部からのどのトリガに反応するかを、このクラス図で定義します。この定義は以下のような表現になります。



(オプション設定によっては、ポートの型(bool)は表示されません。ユーザーのオプション画面の「要素」グループにある「ポートとパートの型を常に表示」にチェックを入れるか、ダイアグラムのプロパティ画面の「要素」グループにある「ポートの分類子を表示」にチェックを入れると、このドキュメントと同じ表示になります。)

まず、イベントを示すトリガ要素を配置します。トリガ要素は、ステートマシン図で状態間の遷移にトリガを設定すると、プロジェクトブラウザに追加されます。この追加されたトリガ要素を、プロジェクトブラウザからドラッグ&ドロップで配置します。
(このクラス図の構成の前に、次の章で説明するステートマシン図の作成を行うことをおすすめします。)

次に、クラス要素にポート要素を追加します。これらの操作はそれぞれの言語用のツールボックスを利用すると便利です。ポート要素には名前と型を定義して下さい。上記の図および下の図の「reset：bool」と書かれている内容がポート要素の情報です(依存の接続の情報ではありません)。ポートの名前が「reset」で、型が「bool」です。ポート要素の型の指定は、右クリックして「追加設定」→「プロパティの種類の設定」で実行できます。



これらの要素を作成したら、「依存」の接続で両者を結んで下さい。「依存」の接続には名前などの情報は不要です。

そのほか、必要に応じてアクティブクラスに入出力のためのポートや属性などを追加し

て下さい。

6 ステートマシン図の構成

次に、ステートマシン図について説明します。

6.1 最低限の構成

最上位のステートマシン図では、「初期状態」「終了状態」が必要です。つまり、最低限の構成としては以下のようになります。

開始(初期状態)はソースコードに出力されません。開始状態から遷移した先の状態が最初の状態として利用されます。終了は全てのステートマシン図を通して 1 つのみ配置されます。開始状態からは 1 つの遷移のみが許され、その遷移にはガード条件やアクションなどは設定できません。

なお、状態マシン要素の直下にある最上位のステートマシン図は、ハードウェア系言語の場合には非同期処理を表現するために利用されます。サンプルを参考にして下さい。ソフトウェア系言語の場合には不要です。

6.2 トリガによる遷移

トリガによって遷移する場合には、遷移のプロパティの「トリガ」欄で指定します。「名前」の欄はコンボボックスになっていて、定義済みのトリガを指定することができます。

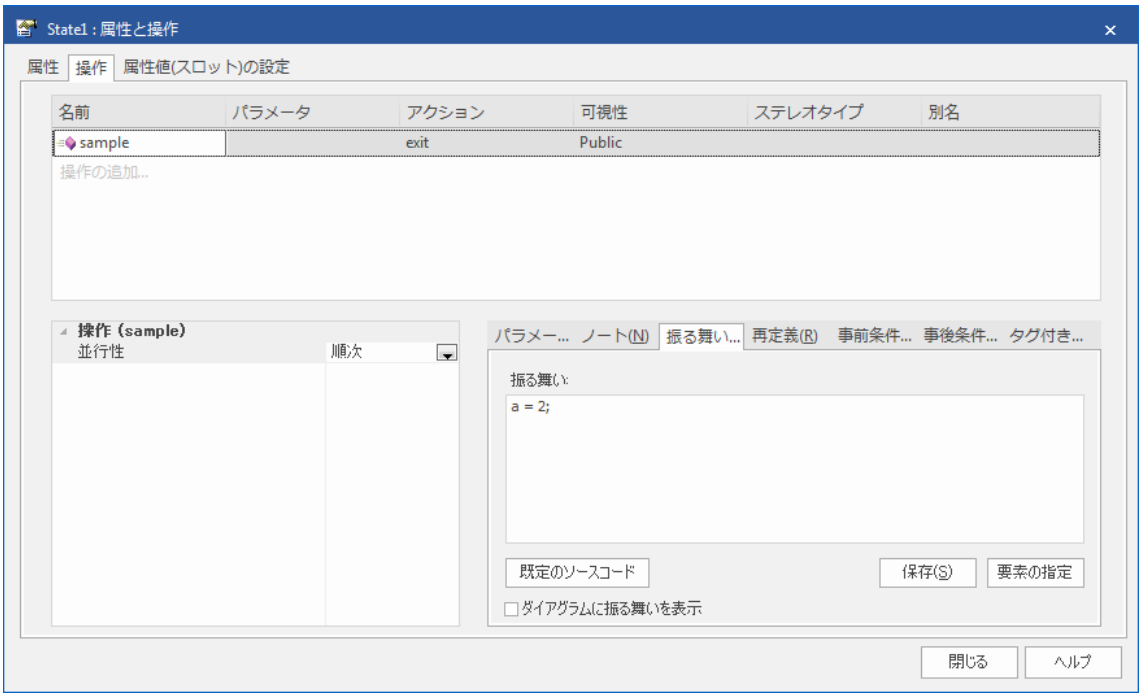
「仕様」の欄には、ハードウェア系言語の場合には、時間(Time)トリガの場合には positive あるいは negative を、変更(Change)トリガの場合には true あるいは false を指定します。ソフトウェア系の言語の場合には、種類には「変更」を指定し、仕様の欄に評価式を指定します。記入した内容は、「トリガ名 == 仕様」の形式で、if 文の判定に利用されます。

ハードウェア系言語の場合には、トリガによる遷移は最上位のステートマシン図でのみ

利用します。子ダイアグラム内では利用できません。

6.3 状態内のアクション

状態に定義する Do/Entry/Exit アクションは、それぞれソースコードに出力されます。これらのアクションを定義する場合には、状態要素を右クリックして「操作」を選択して下さい(状態要素を選択してショートカットキーの F10 キーも利用できます)。名前を指定した後、「振る舞い」タブの「振る舞い」の欄に、これらのアクションに関連する実際のソースコードを入力して下さい。以下の画面は入力の例です。その下のソースコードは C++での出力の例です。

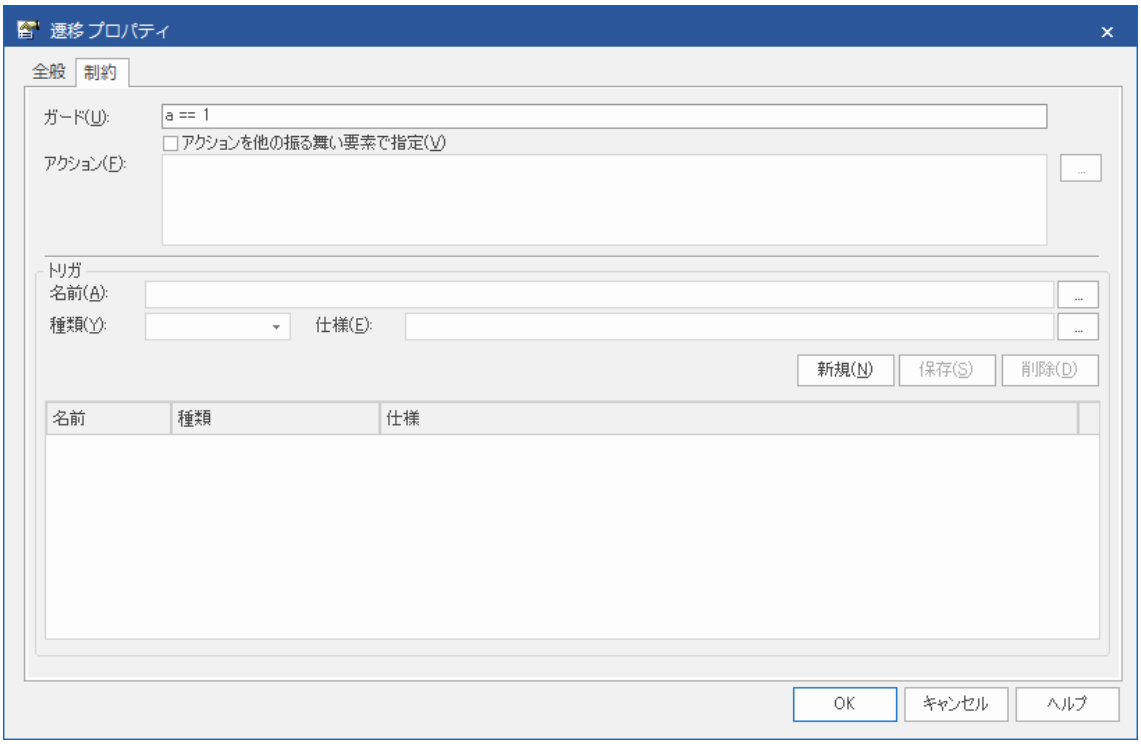



```
switch ( command )
{
		case Do:
		{
			nextState =  状態マシン_終了;
			break;
		}
		case Exit:
		{
			a = 2;
			break;
		}
		default:
		{
			break;
		}
}
```

なお、既定のテンプレートの内容では、最初の状態の entry アクションおよび終了直前の exit アクションは実行されません。ご注意ください。

6.4 遷移の条件

状態間の「遷移」は、何も指定しない場合には自動遷移となります。遷移する際の条件を指定する場合には、遷移のプロパティの「ガード」欄に入力します。入力した内容がそのまま条件判定の式になります。

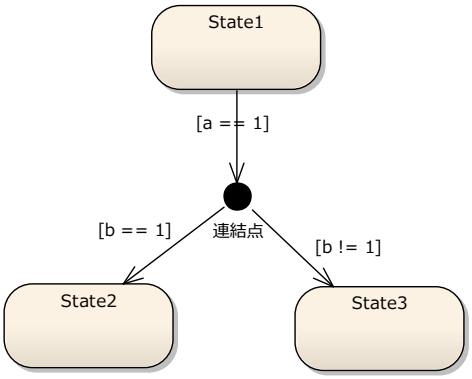


ガード条件を上記のように設定した場合には、C++言語では以下のように出力されます。if 文の条件になっていることを確認して下さい。

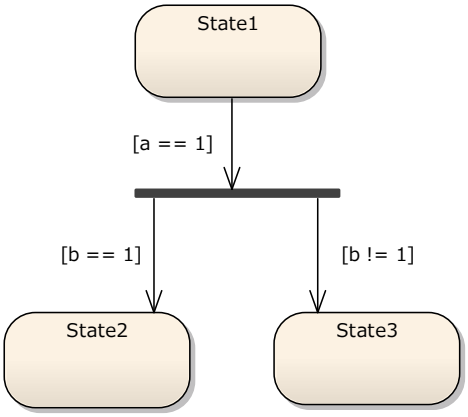
```
case Do:
{
        if ( a == 1 )
        {
                nextState =  状態マシン終了 ;
        }
        break;
}
```

6.5 条件分岐

条件分岐を表現するためには、「連結点」や「フォーク・ジョイン」要素を利用します。

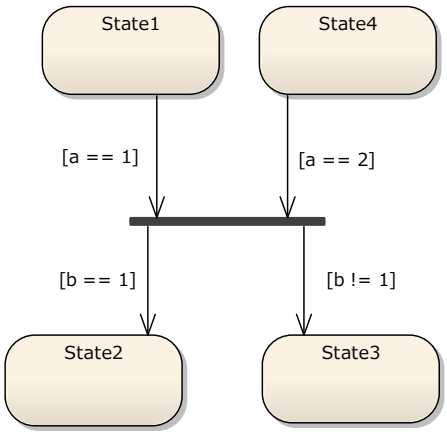


連結点要素を利用した例



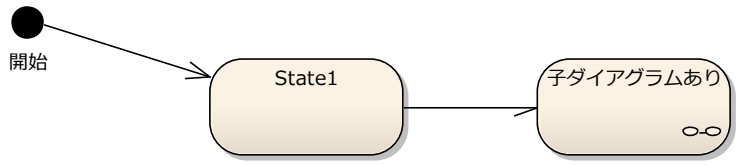
フォーク要素を利用した例

上記のいずれの場合でも、生成されるソースコードは同じです。これらの要素を利用することで、以下のような複雑な条件分岐も可能です。



6.6 子ステートマシン図から親ステートマシン図に戻る

下記のようなステートマシン図を作成することで、詳細な遷移を別のステートマシン図として表現できます。

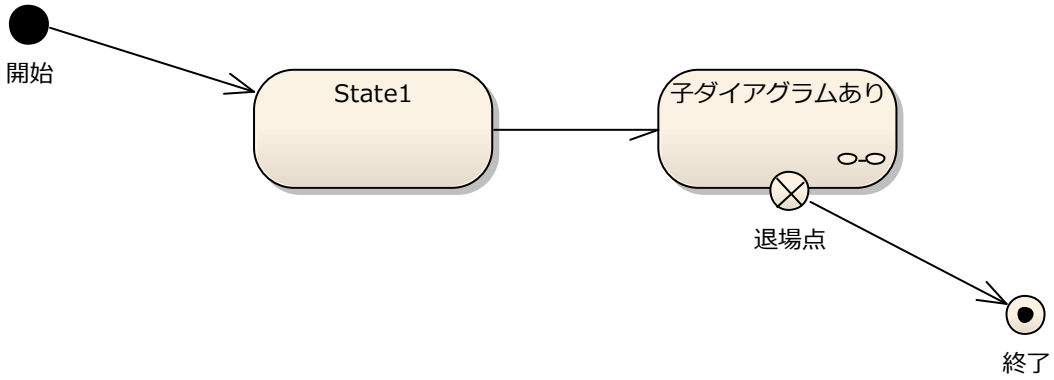


このようにすることで複雑な状態遷移を簡潔に表現できます。この子ダイアグラム(子ステートマシン図)から親への遷移を表現する場合には、「退場点」要素を利用します。

子ダイアグラムの例

このようにして作成した子ダイアグラムの退場点要素を、親のステートマシン図にコピーします。具体的には、子ダイアグラム要素の退場点要素を選択した状態で Ctrl+C キーを押し、親のステートマシン図で Ctrl+V を押します。

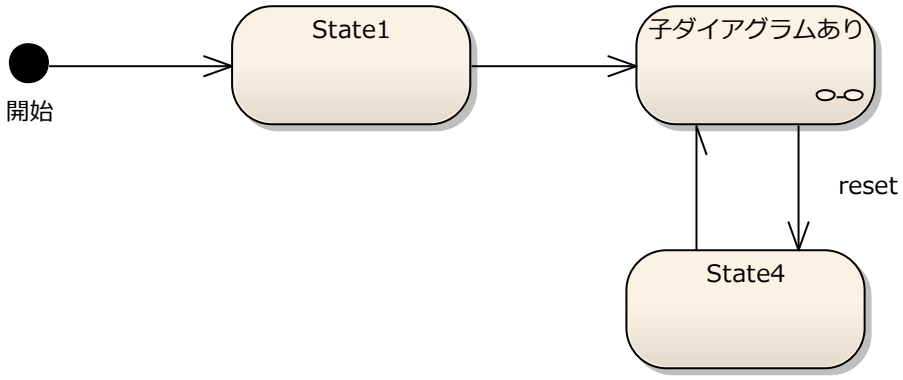
これで連結が可能になりますので、退場点から続きの状態を表現します。



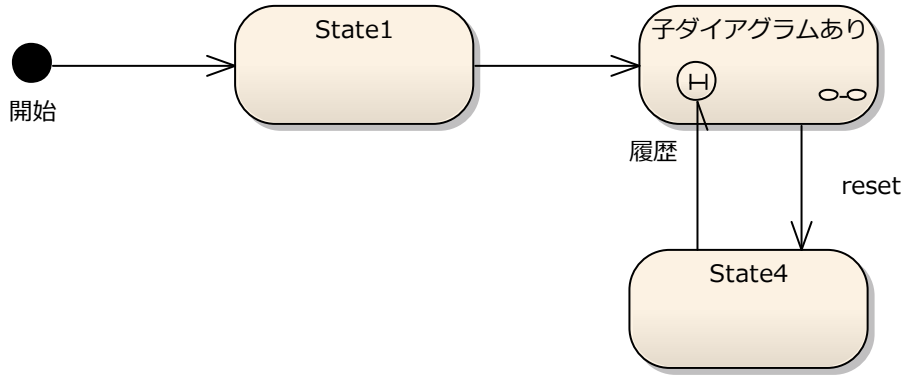
6.7 履歴

ある子ダイアグラムの遷移中に、親ダイアグラム側で定義されたトリガによって状態が変わる場合があります。このような場合に再度子ダイアグラムに戻るときには、以前の状態から継続するような表現が可能です。

親ダイアグラム:



このような場合には、履歴要素を利用します。



プロジェクトブラウザにおいて、履歴要素が対象状態の子として存在することを確認して下さい。ダイアグラム内で履歴要素を子ダイアグラム要素の上に移動すれば、自動的にこの関係が作成されます。

このような場合に生成されるコードの例です。

```
nextState = StateMachine_子ダイアグラムあり_History;
transcend = true;
StateMachine_History   = curr_State;
```

6.8 状態の入れ子

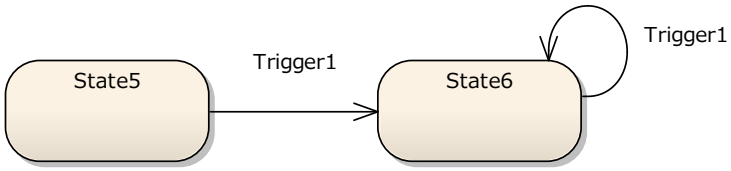
状態要素の中に別の状態が含まれる、いわゆる「入れ子」の状態に対応するためには、親となる状態要素に「子ダイアグラム」が存在する必要があります。親要素を右クリックして「子ダイアグラムの追加」→「子ダイアグラムを作成」を実行し、要素の右下に「∞」マークが表示されるようにしてください。

入れ子状態の内部の状態遷移については、この操作で作成された子ダイアグラムに記述することもできますし、親状態の上に状態要素を配置して、同じダイアグラム内に記述することもできます。

7 注意事項

以下、ステートマシン図からのソースコード生成機能を利用する上での注意点をいくつか明記します。

・ ハードウェア系言語のトリガでは、トリガによる全ての遷移先は同じ状態でなければなりません。また、遷移先の状態では、そのトリガによる自己遷移の定義が必要です。



・ C 言語を利用する場合には、プロジェクトのオプションの「オブジェクト指向のサポート」を「True」に設定しなければなりません。
・ アクティブクラスの名前や状態要素の名前などは、そのままソースコードの内容の一部として出力されます。日本語名も利用したい場合には、Enterprise Architect の「別名」機能を利用し、別名に日本語名を入力して下さい。
・ ポートの型がパッケージ内で定義されていない場合には、すべて String 型(具体的な値はオプションによって決まります)になります。
・ シミュレーション機能と併用する場合には、シミュレーション機能のために、属性名に sim.や this.などの接頭辞を追加する必要があります。この sim.などがコード生成時に不要になる場合には、コード生成テンプレートをカスタマイズし、ソースコードへの出力時に sim.を削除すると良いでしょう。
(第 2 章で紹介した C 言語および C++言語のサンプルは、この削除の処理が含まれています。)

8 生成されるソースコード

既定のコード生成テンプレートの内容を変更しない場合、生成されるソースコードにはいくつかのメソッド(関数)が含まれます。この概要を説明します。

- `StatesProc`
  それぞれの状態の `Do` イベントなどを処理するための分岐が含まれます。
- `TransitionsProc`

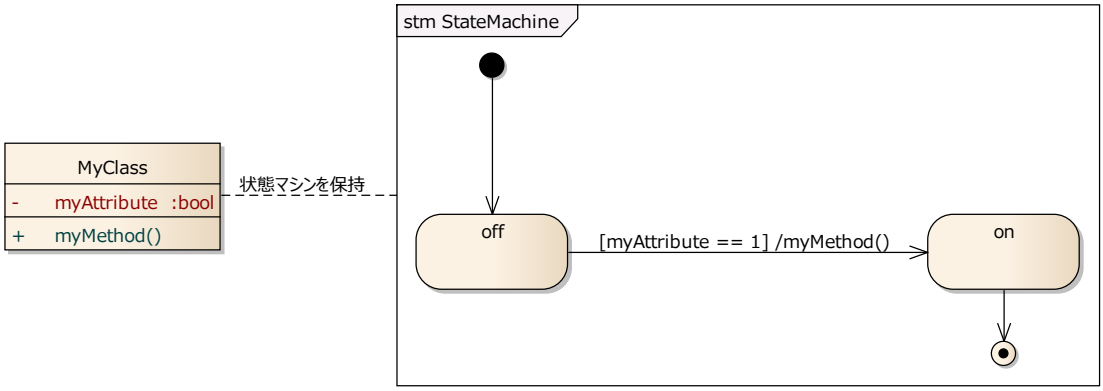
遷移の内容にアクションが含まれる場合、その処理をするための分岐が含まれます。遷移のアクションが全く定義されていない場合には、このメソッドの中身は空になります。

- `initializeStateMachine`
  状態マシンの初期化処理が含まれます。
- `runStateMachine`
  状態マシンの処理を行います。内部では、`StatesProc` を呼び出します。
  このドキュメントの説明にあるように、既定のコード生成テンプレートの場合には、外部からトリガを与える必要があります。ソフトウェア系言語の場合、トリガに対応する変数の値を変更するか、ガード条件の判定結果を変化させるように変数の値を変更することになりますが、これらの処理は `Enterprise Architect` の自動ソースコード生成の範囲には含まれておりません。

9 ラウンドトリップでの設計開発を行う場合

既に説明しましたように、ステートマシン図(振る舞い図)からのコード生成を利用する場合には、常に UML のクラス図から、ソースコードを生成する片方向での処理となり、生成されたソースコードを編集して、一部の操作(メソッド)の実装を記述したり、クラス図に反映させたりすることはできません。

ただ、実際には以下のような状況が考えられます。

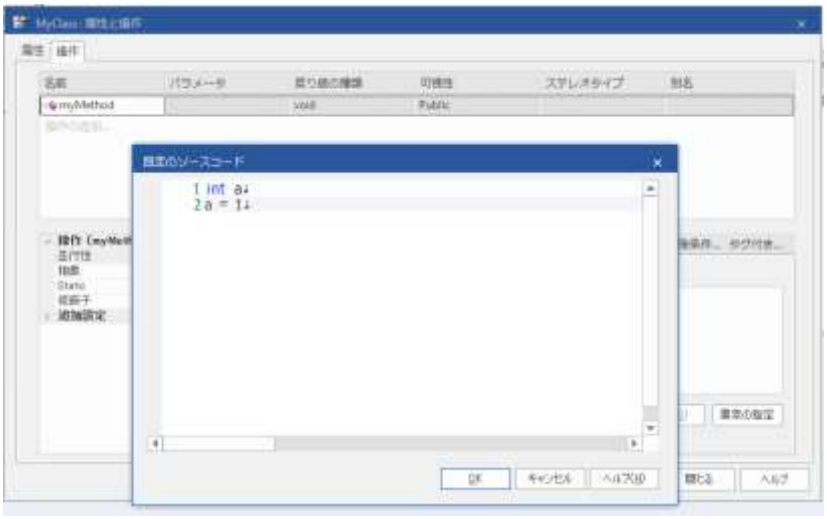


このように、ステートマシン図の中でクラスの操作を呼び出すような処理を記述する場合に、クラス図に定義した操作の中身(実装)は空になります。しかし、生成されたソースコードに処理を記述すると、ステートマシン図の内容を編集して再度ソースコード生成すると、その内容が上書きされ、失われます。

このような場合には、以下のいずれかの方法で対応します。

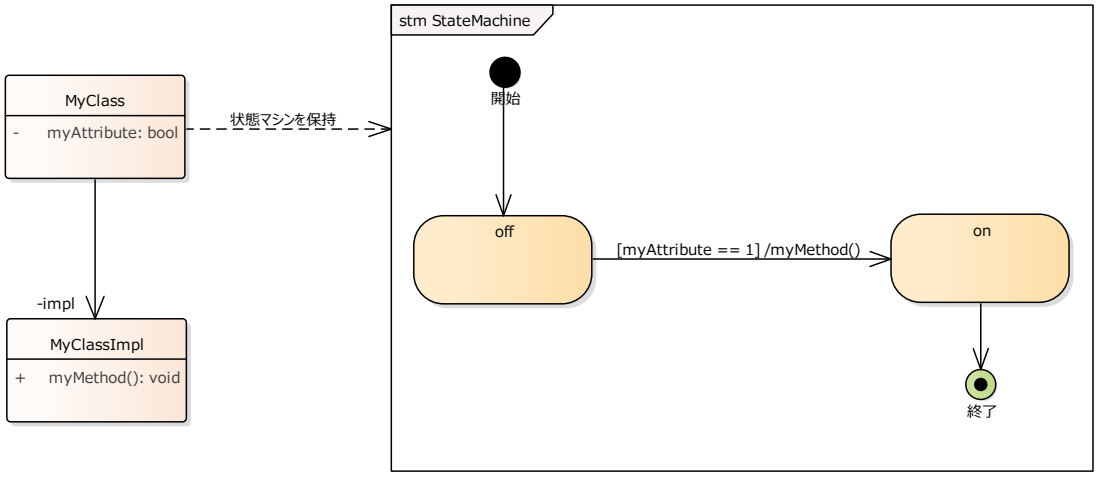
1. 操作の実装もモデル内に記述する

   対象の操作のプロパティ画面の「振る舞い」タブにある「既定のソースコード」ボタンを押すと表示されるエディタで処理内容を記述すると、その内容がソースコード生成時に自動的に含まれます。これにより、操作の処理自体もEnterprise Architectで記述し、変更する場合にはソースコードを修正するのではなく、生成元になるクラス要素の操作のプロパティ画面から修正し、再度ソースコードを生成し直します。



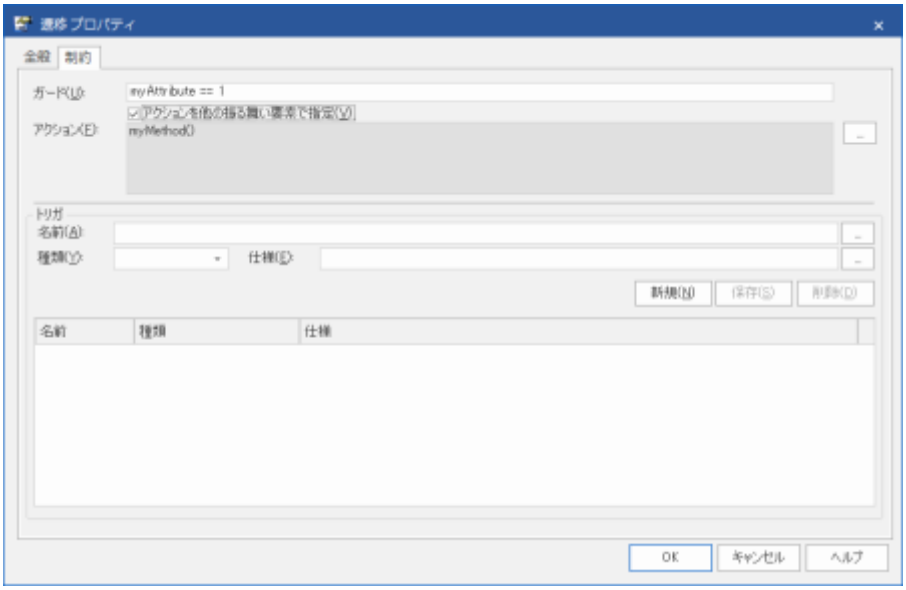
2. 状態遷移を制御するクラスと、処理を持つクラスを分ける

   2番目の方法として、クラスを以下のように分割する方法があります。



ここで、ステートマシン図内の遷移に関連づけられたmyMethodは、以下の画像のよ

うに、MyClassImplクラスの操作に関連づけています。



このようにすることで、第2章で紹介したC++言語のカスタマイズサンプルの場合には、以下のようなソースコードが生成されます。

```
case Exit:
{
	//Exit Behaviors..
	//Transition Actions
	if ((myAttribute == 1))
	{
		impl->myMethod();
	}
```

一方で、このmyMethodの操作を持つクラスは、状態マシンを持っていませんので、ラウンドトリップでの設計開発が可能です。つまり、クラス要素からソースコードを生成した後、ソースコードの内容を編集し、再度クラス要素に同期することができます。

この方法は、状態遷移設計とその実装部分の担当者が別れる場合に効果的です。状態遷移設計の担当者は、状態の遷移とそこから呼ばれる処理を列挙することに注力し、個々の処理については別クラスとして別の担当者が実装します。

なお、この方法をとる場合、第3章で説明した、クラス要素ごとに上書きコード生成する設定方法で、状態マシン要素を持つ要素のみを上書きコード生成にしておくと良いです。

改版履歴

2009/03/24　初版
(アドイン「MDGTechnology for RealTime UML」のためのドキュメントを改版して作成)
2009/08/31 ドキュメントのタイトルを変更。
2010/06/28　ソースコードの同期などについての補足情報を追加。
2010/12/20 別途提供している。C 言語および C++言語向けのカスタマイズしたテンプレートについての説明を追加。
2011/05/18 Enterprise Architect 9.0 のリリースに伴い、内容を更新。
2011/08/19 状態の入れ子についての説明を追記。
2011/09/05 モデルの構成について表現を改善。
2011/09/12 ソースコードの「上書き」でないとステートマシン図の内容が出力されない点を追記
2011/10/24 第 9 章を追加。その他、細かい表現の改善。
2011/12/08 Enterprise Architect 9.2 のリリースに伴い、内容を更新。
2013/03/01 第 10 章を追記。その他説明を追加。
2014/04/22 Enterprise Architect 11.0 のリリースに伴い、内容を更新。
2014/12/09 「exe 実行の状態マシン」についての説明を追記。
2015/12/01 Enterprise Architect 12.1 のリリースに伴い、内容を更新。
2016/10/07 Enterprise Architect 13.0 のリリースに伴い、内容を更新。