Code Template Framework Guide

by SparxSystems Japan

Enterprise Architect 日本語版

コードテンプレートフレームワーク 機能ガイド

発展編

(2022/04/27 最終更新)

目次

1.はじめに

Enterprise Architect には、コードテンプレートフレームワーク(以下 CTF と表記します)と呼ばれる新しい概念・機能が搭載されています。このドキュメントでは、この CTF を利用した発展的な機能について説明します。

この CTF に関する説明は、以下の 4 つに分割して行います。

- 基礎編
  CTF の概念の説明・サンプルを通した機能の確認
- 応用編
  既存のテンプレートの修正
  (ステレオタイプを指定したテンプレートの追加)
- 発展編(本ドキュメント)
  Enterprise Architect の対応していない独自のプログラム言語のソースコード生成
- 振る舞い図からのコード生成編
  ステートマシン図など、振る舞い図からのソースコード生成時に役に立つ情報

2.独自言語のソースコード生成

本ドキュメントでは、架空のプログラム言語 SSJ を例にして、UML の簡単なクラス図から SSJ のソースコードを生成するまでを例として、CTF のカスタマイズ機能について説明したいと思います。

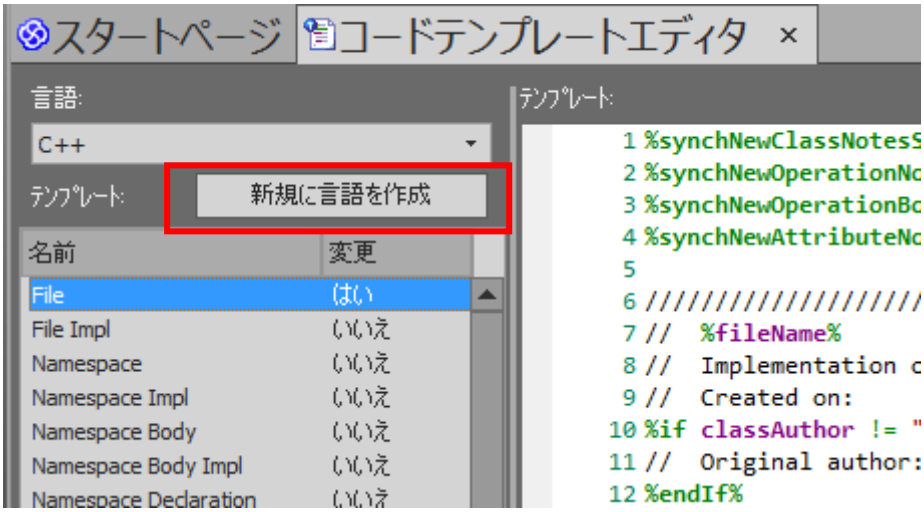
なお、CTF は、基本的には UML のクラス図の情報を元にテキストファイルを生成する機能ですので、これ以外のプログラム言語に対応したソースコードや、独自のルールをクラス図に定義して、既存の言語への拡張を行うこともできます。

また、ユニファイド版あるいはアルティメット版を利用すると、ステートマシン図・アクティビティ図・シーケンス図の内容を CTF から取得できるようになります。これにより、これらの図で表現された内容を操作(メソッド)の実装としてコード生成するなどの対応も可能になります。この概要は「振る舞い図からのコード生成編」のドキュメントをご覧下さい。

3.独自言語のための準備

最初は、CTF とは関係なく、Enterprise Architect に独自の言語の情報を定義しなければなりません。そのための方法を説明します。

最初に、コード生成テンプレートエディタのタブから「新規に言語を作成」ボタンを押して、プログラム言語に関する設定ダイアログを表示させます。



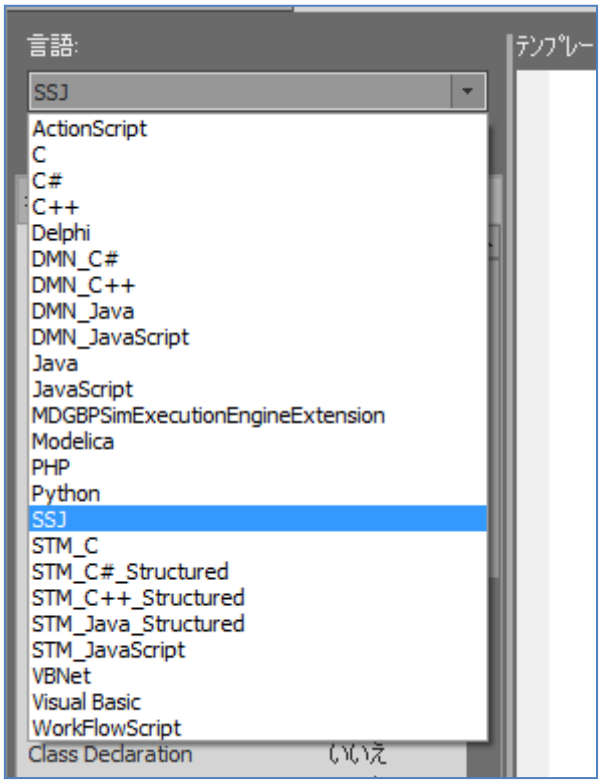
すると、「プログラム言語のデータ型」画面が表示されますので、「新規追加」ボタンを押して新規にプログラム言語を追加します。

その後、そのプログラム言語の型を定義していきます。以下のダイアログは、入力した後の状態です。なお、変数の型は少なくとも 1 つは定義する必要があります。今回の例では、次の画像のように 2 つ入力しました。なお、「共通の型」とは、言語にかかわらず存在する主な種類について、主に MDA 変換の実施時などに利用するための情報です。例えば、整数の型であれば Integer、文字列であれば String などとなります。ここで入力可能な値は、既存の他のプログラム言語の定義を参考にしてください。今回のドキュメントの範囲では、この「共通の型」の指定は必須ではありません。

準備は以上です。これで、クラスのプロパティや CTF の設定画面に定義した SSJ が選択肢として表示されるようになります。

4.生成コードの作成

次に、コードテンプレートエディタを起動します。「コード」リボン内の「ソースコード」パネルにある「設定」ボタンを押すと表示されるメニューから「コード生成テンプレート」を選択します。ここで、「言語」のドロップダウンリストから、先ほど作成した SSJ が選択できることを確認してください。ただし、SSJ を選択した場合には、どのテンプレートを選択しても右側のエディタ欄には何も表示されません。つまり、現在はテンプレートの中身は空になっています。



ここで、最初から全てを作成しても良いのですが、適当なプログラム言語のテンプレートをコピーするという方法もあります。この場合には、その後、必要なテンプレートを適切に書き換えていきます。

なお、今回はサンプル言語の SSJ について検討・説明するドキュメントではありませんので、この先の出力内容の検討については省略いたします。

4.テンプレートの呼び出し関係

CTF では、プログラム言語の種類を問わず、最初に File テンプレートを呼び出します。そして、File テンプレート内ではその他のテンプレートを呼びだすようになっています。

C++言語のようにヘッダファイルとソースファイルが分かれている場合、ソースファイルは File Impl テンプレートから生成されます。File Impl テンプレートからは、同様に Impl が名前に含まれるテンプレートを呼び出しています。

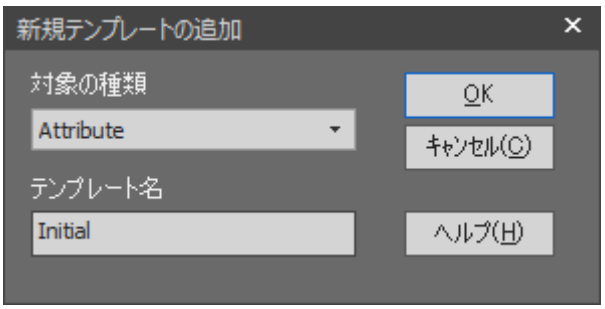
基本的なテンプレート間の依存関係は、次のようになっています。(プログラム言語によ

って異なりますが、多くの場合には以下のような構成です)

- File
  - ImportSection
  - Namespace
    - NamespaceDeclaration
    - NamespaceBody
      - Class
        - ClassNotes
        - ClassDeclaration
        - ClassBody
          - Attribute
            - AttributeNotes
            - AttributeDeclaration
          - Operation
            - OperationNotes
            - OperationDeclaration
              - Parameter
            - OperationBody

この構成を参考に、必要なテンプレートを他の言語からコピーしたり、あるいは作成したりしてください。

また、コード生成テンプレートで利用できるテンプレートには、既定のテンプレートのほかに、独自に定義したテンプレートを利用して拡張することもできます。「新規テンプレートの追加」ボタンを押すと、以下のような画面が表示されます。



ここで、「対象の種類」として既存のテンプレートの名前を指定すると、そのテンプレー

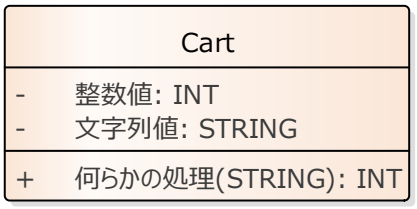
トと同じように使用することができます。上記の例の場合には Attribute を選択していますので、Class Body などクラス関係のテンプレートから、list マクロを利用して独自のテンプレートを呼び出すことができます。

対象の種類が「<None>」のテンプレートの利用方法については、「振る舞い図からのコード生成編」をご覧下さい。

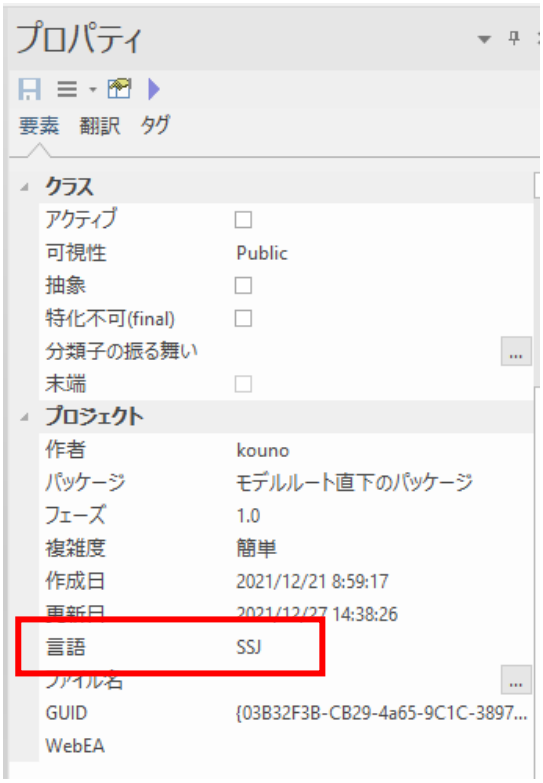
5.コードの生成

それでは、作成したテンプレートを利用して実際にコード生成を行います。

例えば、下のようなごく簡単な Cart クラスを例にします。



ダイアグラム内で Cart クラスを選択し、プロパティサブウィンドウから「言語」を「SSJ」に設定してください。この状態でクラスを選択してコード生成を実行すると、ファイルの入力画面になります。独自に定義した言語は拡張子が表示されませんので、適切な拡張子を付けて保存するファイル名を指定してください。
(独自の拡張子を設定するためには、「MDG テクノロジーファイル」を作成し、その中で定義する必要があります。C 言語や C++言語のように、ヘッダファイルとソースファイルを別々に生成したい場合も同様に「MDG テクノロジーファイル」での設定が必要です。ヘルプに「MDG テクノロジーファイル」への設定方法が記載されています。)



なお、ここで注意すべき点があります。このように独自に定義したプログラム言語から、逆にクラス図を生成することはできません。独自に定義したプログラム言語の内容を読み込んでクラス図を生成したい場合には、「読み込みルールエディタ」を利用して読み込みルールを記述する必要があります。
(この読み込みルールは、CTF とは全く異なる文法になります。概要はヘルプの「コード読み込みの独自定義」のページをご覧下さい。)

6.最後に

以上で、CTF についての説明を終わります。CTF を利用すれば、今回紹介したような新規のプログラム言語に Enterprise Architect を対応させることもできます。

7. 付録1 コード生成テンプレートの詳細説明

ここでは、コード生成テンプレートへの理解を深めるひとつの方法として、実際にEnterprise Architect が利用している既定のテンプレートについて、その意味を説明しています。これにより、コード生成テンプレートの理解が深まると考えます。

ここでは、C++言語の「Operation Declaration」テンプレートを題材とします。
(最新のテンプレートとは内容が異なるかもしれません。)

赤字の内容が説明になります。

```
%PI=" "%
```

PI マクロの内容を設定します。PI マクロとは、テンプレートの各行が処理された後に追加される区切りです。ここでは、空白文字を指定していますので、下記のそれぞれのテンプレートの行が処理されるごとに、(改行ではなく)空白文字を付加することを指定していることになります。

```
%opTag:"inline"=="true" ? "inline" : ""%
```

操作のタグ付き値(opTag)の中の、inline の値を確認しています。定義されていてその内容が true の場合には、inline という文字列を表示することになります。定義されていない場合や、定義されていても内容が true ではない場合には、コロンの後ろに書かれている文字列を出力します。ここでは、空文字列になっているので、何も出力しないことになります。

```
%opTag:"afx_msg"=="true" ? "afx_msg" : ""%
%opTag:"explicit"=="true" ? "explicit" : ""%
%opStereotype=="friend" ? "friend" : ""%
%opAbstract=="T" ? "virtual" : ""%
%opStatic=="T" ? "static" : ""%
%opConst=="T" ? "const" : ""%
%opTag:"volatileReturn"=="true" ? " volatile" : ""%
```

上記の内容は全て先ほどの inline の場合とほぼ同じです。ただ、Enterprise Architect の共通のプロパティ項目の場合には、opTag ではなく、opAbstract などのような固有の名前が定義されています。また、これらの値は、T あるいは F となる点についても注意して

ください。

```
%RESOLVE_QUALIFIED_TYPE("::")%
```

このマクロは、型名の解決を行うためのマクロです。現在のテンプレートは操作に関するテンプレートですので、このマクロは操作の戻り値に関して処理を行い、結果を出力しています。

```
%opTag:"callback"=="true" ? "CALLBACK" : ""%
%opMacros%
```

（上記説明なし:他の項目と同様の処理）

```
%PI=""%
```

ここで、PI マクロの値を空文字列に変更しています。つまり、これ以降はテンプレートの各行を処理した後、空白文字や改行を追加しないことを示しています。

```
%opName%(%list="Parameter" @separator=", "%)
```

opName は定義済みの項目(操作の名前)です。その後の list マクロは、定義済みの他のテンプレートを呼び出すためのマクロです。ここでは、Parameter テンプレートを呼び出すことを指定しています。list マクロを利用すると、テンプレートが対象にしている項目を繰り返し呼び出すことになります。ここでは、操作のパラメータが対象です。操作にパラメータが 3 つある場合には、それぞれのパラメータに対して Parameter テンプレートの内容が 3 回処理されることになります。

```
%if opIsQuery=="T" or opStereotype=="const"%
 const
%endIf%
```

条件分岐については、if と endIf で囲みます。if と endTemplate を利用した場合には、if 文の条件が真の場合には、endTemplate に到達した時点でテンプレートの処理を終了します。

```
%opTag:"volatile"=="true" ? " volatile" : ""%
%opTag:"throws"=="" ? "" : " throw " value%
%if opPure=="T"%
 =0;
%endTemplate%
```

（上記説明なし:他の項目で説明済みの処理。以下の内容についても同様なので、説明を

省略します。)

次に、同じく C++言語の「Class Body Impl」テンプレートを題材とします。「Impl」という名前が付いているテンプレートは、C++であればソースファイル(cpp)に生成される内容であることを示しています。

ここでは、いくつかの処理について説明します。

```
$templateArgs=%list="ClassParameter" @separator=", "%
```

コード生成テンプレートでの変数は、最初に $ 記号を付加します。事前の宣言は不要です。ここでは、list マクロを利用して、list マクロで得られた結果を", "でつなげた文字列を格納しています。

```
%if $templateArgs != ""%
$templateArgs="<" + $templateArgs + ">"
$templ="template" + $templateArgs
%endIf%
```

テンプレート宣言に対応する部分の処理です。最初の if 文で、変数$templateArgs の値が空でない場合のみ処理されます。

```
$consPrefix=$templ + "¥n" + %classQualName% + "::"
```

コンストラクタの宣言で共通に利用するための文字列を作成し、変数$consPrefix に保管しています。

```
%PI="¥n¥n¥n"%
```

PI マクロの値を改行３つに変更しています。これにより、これ以降のそれぞれの出力結果の後に、空行が２つ追加されることになります(３つの改行のうち、出力結果の末尾で１つ改行する。残りの２つの改行で空行が生成される)。

```
%if genOptGenConstructor == "T" and genOptGenConstructorInline != "T"
and classHasConstructor != "T"%
$consPrefix%className%(){¥n¥n}
%endIf%
```

ここでは、Enterprise Architect のユーザーのオプションで設定できる項目の値を確認し、条件を満たす場合にソースコードを生成しています。

8. 付録2 よくある変更

以下の内容は、テンプレートをカスタマイズして出力結果を変えたいという場合に、よく行われる変更の例です。参考にしてみてください。

○ 括弧｛の位置を変更したい

C++言語において、｛の位置を

```
void function() {
…
}
```

としたいか、あるいは

```
void function()
{
…
}
```

としたいがあると思います。このような場合には、

Class Body Impl の 12, 16, 21 行目：
)¥n{¥n¥n} →)**¥n**{¥n¥n} に変更
(コンストラクタ・デストラクタ・コピーコンストラクタ)

Opreation Body の 1 行目：
{¥n→**¥n**{¥n に変更
（その他の通常メソッド）

のようにして改行(¥n)を追加します。

○ public，protected，private フィールドの出力順を変えたい

Class Body の 68 行目以降でそれぞれ定義されていますので、この内容を入れ替えます。例えば、public の内容を出力している範囲は、

$pubFeatures = %TRIM($pubFeatures, "¥n")%
$pubFeatures += "¥n" + %list="Attribute" @separator="¥n" @indent="¥t" attScope=="Public" or
linkAttAccess=="Public" or attScope=="Package" or linkAttAccess=="Package"%
・
・（途中省略，Public フィールドのブロック）
・
%endIf%

となります。

○ コメントの出力形式を//形式に変えたい

この場合、Attribute Notes の記述を以下のように変更します。

%if genOptGenComments != "T"%
%endTemplate%

%PI=""%
$style = %genOptCPPCommentStyle%

$wrapLen = %genOptWrapComment%
%if $style == "XML.NET"%
%XML_COMMENT($wrapLen)%
%elseIf $style == "JavaDoc"%
%JAVADOC_COMMENT($wrapLen)%
%else% （注：C++時の記述の場合の処理はここから）
$wrapLen = %genOptWrapComment=="-1" ? "-1" : "40"%
$attribute = %WRAP_LINES(attNotes, $wrapLen, "//", "")%
%if $attribute != ""%
¥n$attribute（注：直前に空白行を追加）
%endIf%

○改版履歴

2007/01/09 8 章「付録 2 よくある変更」を追加
2008/03/06 一部補足情報を追加
2009/03/24 バージョン 7.5 のリリースに伴い、画像を更新。
2009/08/31 ドキュメントのタイトルを変更。
2011/05/18 バージョン 9.0 のリリースに伴い、画像を更新。
2012/12/14 バージョン 10.0 のリリースに伴い、内容を更新。
2013/01/24 7 章の例題の内容を最新バージョンのものに差し替え。補足説明の追加。
2013/04/22 バージョン 11.0 のリリースに伴い、内容を更新。
2015/12/01 バージョン 12.1 のリリースに伴い、内容を更新。
2016/02/15 利用することができない情報が残っていた箇所を削除。
2016/10/07 バージョン 13.0 のリリースに伴い、内容を更新。
2018/05/16 バージョン 14.0 のリリースに伴い、内容を更新。
2019/08/22 バージョン 15.0 のリリースに伴い、内容を更新。
2022/04/27 バージョン 16.0 のリリースに伴い、内容を更新。