

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

応用編

(2021/12/27 最終更新)



目次

1.はじめに	3
2.ステレオタイプによる拡張の例	3
3.ステレオタイプによる拡張の方法	4
4.ステレオタイプによる拡張の作成例	6
5. 応用例	9
6. 最後に	11

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) と呼ばれる機能が搭載されています。このドキュメントでは、この CTF の基本的な内容に 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(本ドキュメント)
 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
 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.ステレオタイプによる拡張の例

CTF を利用して、コード生成を行うことができるのは基礎編で説明したとおりです。た だ、実際には「xx の場合には yyy という出力をする」というようなケースが多くあります。 こういった条件の多くは、制御マクロの(if-elseIf-endIf)を利用します。この if 文の条件に、 クラスのプロパティや、あるいは EA のオプション設定情報などを利用することで、ソース コードの出力処理を分岐させることができます。

もうひとつ、条件として利用できるものに、ステレオタイプがあります。例えば、C++ における struct というステレオタイプがあります。このステレオタイプを下の図のように 指定すると、生成されるソースコードは以下のようになります。ステレオタイプの指定が ない場合と比較してみると違いがよくわかるでしょう。

	«struct» Parameters
-	p1: int
-	p2: long
-	p3: char

ステレオタイプありの場合	ステレオタイプなしの場合
struct Parameters	class Parameters
{	{
	public:
private:	Parameters();
int p1;	<pre>virtual ~Parameters();</pre>
long p2;	
char p3;	private:
	int p1;
};	long p2;
	char p3;
	};

CTF では、このようにステレオタイプによってソースコード生成の結果を変えることも できます。次の章では、今回の struct を例にとって、その方法を説明します。

なお、この例で挙げている struct については、既定のテンプレートでは制御マクロを利 用して実装されています。4 章で、ステレオタイプを利用した拡張方法の例として実際に説 明します。

3.ステレオタイプによる拡張の方法

まずは、コードテンプレートエディタを起動します。今回は、C++におけるステレオタイ プなので、言語から C++を選択します。次に、テンプレート一覧から対象のテンプレート を選択します。多くの場合、「ステレオタイプ指定のテンプレート」欄は空白ですが、中に は項目が表示されるものもあります。例えば、次の図のような場合です。

TIN		
C++		1 %if classIsInstantiated"T"%
100mm41 2540	- Britten Chatter	2 SendtemplateS
1000000 Film	- 36 88 30 9453	
品約	死死	+ Spuceatures = Slist= InnerClass geparator= \n\n gindent= \t classcope== Public or class
h	this .	Spretratures - Mist InterLats gaparater '() in indet '(Classoper Policet + 1)
He lines	16.52	7 Standardors VILLET Clairflormater & Generator" "
Levennee	10657	R R R R R R R R R R R R R R R R R R R
lanestace Inci	DESE	
Lamenta Rody	0.04	10 % classStereotype = "struct" and classStereotype = "union"%
tanespace Body Insti	116.52	11
Langemann Darlar altern	14.50	12 % if genOptStmCPPUseNewTemplate is "T" or stmContextHasStatewachine is "T"%
Isnessare Decision Isni	100.52	15 %if genOptSenConstructor == "T" and classMesConstructor 1= "T"%
Tase	1432	14 Scons = "\n\t" + NgenOptGenConstructorInline == "T"?"inline ":"%
Care Torri	14.50	15 Scons += %classNameS+"()"
Class Rivers	Later.	16 Sif genOptGenConstructorInline "T" or elemType "Interfate" or StemplateArgil-""%
Case Pare	14.68	17 Scons += " (Unin's)in"
Anna fatas face	1010	18 Kelsek
LIBER LITTLEY THE R	L % 34	10 \$cons += ";"
2001 P201	10112	20 %endEF%
THEN DOLLA THEN	C.C.C.	21 XendIfX
Jass Declaration	LEUR	22 %if gen0ptDefaultConstructorScope "Public" or gen0ptDefaultConstructorScope "Package"%
Jass Declaration Inge	55.02	23 Spubfeatures += \$cons
Jana Initakeer	1.6.12	24 SeiseIf genOptDefaultConstructorScope ++ "Protected"%
Class Innerits	100.02	25 SprotFeatures += Scons
Gees Palameter	1.6.12	20 Selses
Gass Parameter Constraint	CER	27 Spraveatures += Scons
Attriaute	1.6.15	28 Sendits
Attribute limpl	CN-NR	20 Million and a second
Attribute Notes	0612	Si Sar generative a la ana classicale france () a
Attribute Dectaration	0.625	32 Side = UNV
Attribute Declaration Ingl	0652	22 Grant an Polytonia "
Inport	0.642	34 Sent (A
import limpi	1002	35 Scoot += ResolutionDestructorInline += "T"2"inline ":""%
Import Section	1452	36 Strong + a "-"+" fastlane"+" (1)"
Import Section Impl	いたな	37 Sif genOntienDestructorinine == "T" or elemine == "Interface" or Stepplateares! +""
日前4月4日前5月一	1	MR Scons +e " (\n\n\tE\n"
and in the second	and a	39 Selses
(e.e.)	1000	48 Scons ++ ";"
inumeration.	21213	- 41 Kendl Fh
		42 %if genOptDefaultDestructorScope "Public" or genOptDefaultDestructorScope "Package"%
		43 Spubfeatures ++ Scons
		44 %elself genOptDefaultDestructorScope "Protocted"%
	and an and a second	45 SprotFeatures += Scons
販売チンプレート	020000	
ステレナタイフ指定のかい	CL-Months	
	11010000	

この例では、C++言語の Class Body テンプレートで、クラスのステレオタイプが enumeration に設定されている場合には、別のテンプレートが呼び出されるということに なります。この「ステレオタイプ指定のテンプレート」一覧にある項目を選択すると、右 側のエディタ欄には、指定されたステレオタイプの場合に出力される処理内容が表示され ます。

この、ステレオタイプを指定して処理内容を変える条件として、

- クラスに特定のステレオタイプが設定されている
- ・ 属性あるいは操作に特定のステレオタイプが設定されている

のいずれかの条件か、あるいは両方を組み合わせて設定することもできます。よって、 ステレオタイプAが設定されたクラスの、ステレオタイプBが設定されている操作のみに 適用されるテンプレート、という条件も可能になります。

新規に追加する場合には、一覧から対象のテンプレートを選択した後、ダイアログ左下

の「ステレオタイプ指定のテンプレートの追加」ボタンを押します。下の図のようなダイ アログが表示されますので、クラスあるいは属性・操作(か、場合によっては両方)のステレ オタイプを指定してください。

ステレオタイプ指定	ミのテンプレートの追加	×
₋ ステレオタイプの	選択:	
クラス:	•	
属性·操作:	•	
	<u>O</u> K ++>\tell	

この画面で内容を指定すると、先ほどの「ステレオタイプ指定のテンプレート」一覧に 追加されますので、あとはエディタで編集することになります。

なお、選択肢に含まれないステレオタイプを指定したい場合には、事前に「プロジェクト」リボン内の「リファレンス情報」パネルにある「UMLに関連する定義」ボタンを押す と表示される画面から、「ステレオタイプ」グループで定義する必要があります。

4.ステレオタイプによる拡張の作成例

それでは、実際に struct を生成する例として、テンプレートを編集していきます。最初 は、クラスの宣言部です。これは、Class Declaration テンプレートに相当しますので、こ のテンプレートを一覧から選択します。その後、先ほど説明した手順に従って、struct ステ レオタイプの場合のテンプレートを追加します。内容としては、

struct (クラス名)

となるはずですので、ただ1行

struct %className%

と記入すればOKです。下の図のようになるでしょう。なお、入力した内容が正しければ、

CTF で定義されているキーワードや特定の文字は、それぞれ固有の色で表示されます。 (上記の例で、例えば classNamae のように間違った文字列を入力すると、紫色で表示され ません。入力中は入力支援機能が利用できますので便利です。)

言語:				「テンフ°レー	k:					1
C++			•		1 st	ruct	%clas	sName	%	
テンプレートៈ	新規に	言語を作成								
名前		変更								
File		はい								
File Impl		いいえ								
Namespace		いいえ								
Namespace Impl		いいえ								
Namespace Body	r	いいえ								
Namespace Body	/ Impl	いいえ								
Namespace Decla	aration	いいえ								
Namespace Decla	aration Impl	いいえ								
Class		いいえ								
Class Impl		いいえ								
Class Notes		いいえ		-						
Class Base		いいえ								
Class Interface		いいえ								
Class Body		いいえ								
Class Body Impl		いいえ								
Class Declaration	1	いいえ								
Class Declaration	1 Impl	いいえ								
Class Initializer		いいえ								
ステレオタイプ指定(のテンプレート									
クラス 特	射性	変更								
alias		いいえ								
enumeration		007								
struct		はい								
typedef		いいえ								
			_							
¥6,±8	テレビリートル	1-etin								

っぎに、実際の中身を出力する部分の編集です。これは、属性の宣言ということで、 Attribute Declaration テンプレートになります。同様にして、struct 対応のテンプレート を生成します。そして、中身を書いていくことになりますが、最初から全部書くのは時間 がかかります。今回のような場合には、元々のテンプレートをコピーしてから、それを編 集していくと良いでしょう。

結果から先に書きますと、今回は以下のようになります。

```
%PI=" "%
%attType=="" ? "int" : value%
%PI=""%
%attContainment=="By Reference" ? "*" : ""%
%attName%
```

この処理は、元々のテンプレートから必要な部分だけを抜き取ったものです。最初の行 は、PIマクロを変更しています。この PI 制御マクロは、それぞれの出力の後に出力される 文字を格納しています。初期値では、改行が格納されていますので、例えば

hoge;

fuga;

とエディタで書いた場合には、そのまま2行で出力されます。これを、

%PI=" "% hoge; fuga;

とした場合には、出力される結果は

hoge; fuga;

と1行になります。hoge;と fuga;の間には、PIマクロで指定した空白1つが挿入されています。このように、1行に複数の結果を書く場合や、各行の後を複数行の改行にしたい場合には、PIマクロを利用します。

今回の場合には、最初に PI マクロの中身を空白 1 つに変更し、型指定がない場合には int 型と仮定して、int の文字を出力します。その後、PI マクロの値(=空白文字 1 つ)がソースコ ードに出力されます。その後、今度は PI マクロの値を空文字列に指定しています。そして、 参照型に設定されている場合には、頭に*をつけます。その後、実際の属性名を出力します が、ここで PI マクロを空文字列に指定しましたので、*をつけた場合には、*と属性名の間 にはスペースが追加されずにソースコードに出力されます。

以上のような編集をすると、以下の2つの疑問を持つ方がいるかもしれません。

- この編集結果で出力すると、「private:」も出力されてしまうが、これはおかしいので はないか
- ・ 先ほどの属性の出力は、1つの属性に対しての処理しか記述していないのに、どうして複数の属性がある場合にも正しく処理されるのか

これらの疑問を解決する鍵は、Class Body テンプレートにあります。ここでは、struct ステレオタイプを指定した場合の処理として、以下のような内容を記述しました。

{ %list="Attribute" @separator="¥n" @indent="¥t"% };

このテンプレートを追加すると、先ほどの private の文字列が消えるようになります。そ して、ここで使われているのが、複数の属性を処理するための制御マクロであるリストマ クロです。

このリストマクロの詳しい説明はヘルプファイルに載っていますが、上記の例である程度 推測できるように、全ての属性に対して、区切り文字を¥n(改行)、インデントを¥t(タブ)に して Attribute テンプレートを繰り返し呼ぶという指示をしているのです。これにより、定 義されている属性の内容がすべて出力されます。

5. 応用例

このステレオタイプごとのカスタマイズの方法を応用すると、次のようなモデルベース開発(狭義の MDA)を実践することもできます。

まず、詳細設計段階のクラス図において、それぞれのクラス図に対象のシステムに応じた ステレオタイプを定義し、クラスに割り当てていきます。例として、冷蔵庫を考えます。 そして、次のような詳細設計クラス図を考えました。

(注:現実の冷蔵庫にこのモデルが合致するかどうかはわかりません。あくまでも例として ご覧ください。また、下の図はクラスの「別名」に日本語名を入力し、別名で表示してい る状態です。)



そして、ここで定義されているすべてのステレオタイプについて、ある程度までの実装 を含めたテンプレートを定義します。実装については、ステレオタイプという形でクラス が分類されていますので、ある程度まではテンプレート内に記入することができます。

上記の例ではクラスのみですが、この方法が最も役立つのは、クラスに定義される操作(メ ソッド)にも同様にステレオタイプを定義し、そのステレオタイプに対して共通の実装をテ ンプレートとして作成した場合です。例えば、Operation Body Impl テンプレートに対し

7	テレオタイプ指定	のテンプレートの追加	Ω	×
	ィステレオタイプの	選択:		
		_		
	クラス:	Light		
	属性·操作:	StopCooler	•	
		<u>O</u> K	キャンセル(©)	

のようにして新しいテンプレートを追加し、ソースコードの実装をテンプレートとして記 入します。

このようにテンプレートを作成すると、それ以降はクラスに<<Light>>のステレオタイプ をつけ、操作(メソッド)に<<StopCooler>>のステレオタイプをつけると、実装コードまで ある程度出力することができるようになる、ということになります。

このような方法を利用してクラス図内でクラスやクラス間の関係をモデリングすること により、実装工程での作業を大幅に省略することができます。さらに、独自にテンプレー トをカスタマイズしたり、EXEC_ADDIN マクロを利用して UML モデルの情報をさらに 多く取得できるようにしたりすれば、UML だけで実装のほとんどを出力することも不可能 ではありません。

もちろん、このようなテンプレートを作成するには多くの時間と作業量が必要になりま す。作成したテンプレートを今後のプロジェクトにおいても継続的に使うことで、効果が 出てくるかと思います。

6. 最後に

いかがでしたでしょうか?基礎編よりもだいぶ難しい内容になりましたが、プログラム 言語を知っている方であれば、それほど違和感なく CTF の処理を読んで理解することがで きるのではないか、と考えています。

○改版履歴

- 2007/01/09 5章「応用例」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。
- 2013/01/24 画像の差し替え。細かい補足説明を追加。
- 2015/10/01 バージョン 12.1 のリリースに伴い内容を更新。
- 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。
- 2021/12/27 説明文章の簡潔化。