

pure::variants 継承・ポリモフィズムを用いたバリエーションの管理

継承やポリモフィズムを使用して、他のクラスのインターフェイスや実装を再利用することができます。pure::variants でこのようなバリエーションを管理し、製品バリエーションごとのフィーチャの選択に応じたコードを自動生成させる仕組みについて紹介します。

サンプルプロジェクトとして、“std_transformation-pvproject.zip” を使用しています。ご興味頂ける場合には、ご連絡頂けると幸いです。

(サンプルプロジェクトの設定： pure::variants の Variant Project のコンテキストメニューから、“インポート→既存のプロジェクトをワークスペースへ” を選択し、次へボタンをクリック。“アーカイブ・ファイルの選択” を選び、参照ボタンから “std_transformation-pvproject.zip” を選択しプロジェクトを読み込む。)

1. 対象ソースファイルごとに異なったフィーチャ名を設定する方法

以下、2つのファイルのいずれかを、フィーチャの選択有無によって “ClassB.h” に置き換えます。

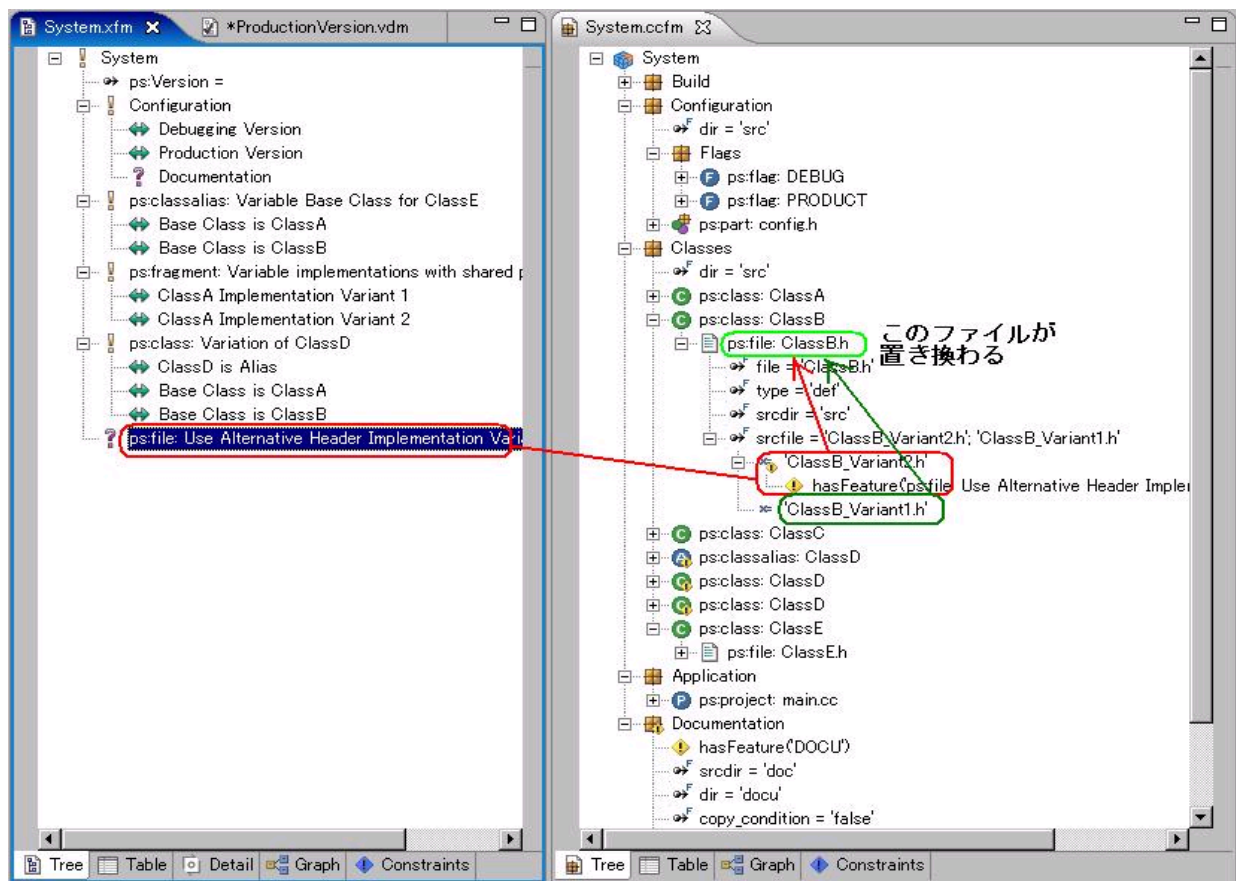
```
[ClassB_Variant1.h]
class ClassB {
public:
    ClassB() {
        std::cout<<(void*)this<<": Class B (Variant 1) created"<<std::endl;
    }
    ~ClassB() {
        std::cout<<(void*)this<<": Class B (Variant 1) deleted"<<std::endl;
    }
}
```

```
[ClassB_Variant2.h]
class ClassB {
public:
    ClassB() {
        std::cout<<(void*)this<<": Class B (Variant 2) created"<<std::endl;
    }
    ~ClassB() {
        std::cout<<(void*)this<<": Class B (Variant 2) deleted"<<std::endl;
    }
}
```

ファミリーモデル (System.ccfm) の “ps:class: ClassB → ps:file: ClassB.h” エレメントの “srcfile” アトリビュートに、上記両方のファイルを設定しています。

フィーチャーモデル (System.xfm) のフィーチャー “ps:file: Use Alternative Header Implementation Variant2 for ClassB using ps:srcfile” の選択有無によって、書き込むファイルを切り替えます。

このフィーチャーは、エレメント “ClassB_Variant2.h” にのみマッピングされていますが、今回のように “srcfile” アトリビュートに複数の値を設定した場合、上から調べて最初に見つかった設定のみが採用される仕組みです。その為、このフィーチャーが選択されると上 (ClassB_Variant2.h) が、選択されない場合は下 (ClassB_Variant1.h) が有効になります。



2. ポリモフィズムにおける共通部分と変動部分を分割して管理する方法

下記、3つのソースファイルを選択して結合する仕組みです。

“ClassA.h” に共通部分 (ClassA_Decl.h) を書き込み、さらにフィーチャーの選択に応じて、“ClassA_implVariant1.h” or “ClassA_implVariant2.h” を書き込みます。

<ClassA_Decl.h> クラスのデクラレーション (共通部分)

```
class ClassA {

public:
    ClassA();
```

```

    ~ClassA();
}

```

<ClassA_implVariant1.h> コンストラクタの定義 (バリエーション 1)

```

ClassA::ClassA() {
    std::cout<<(void*)this<<": Class A (Variant 1) created"<<std::endl;
}

ClassA::~ClassA() {
    std::cout<<(void*)this<<": Class A (Variant 1) deleted"<<std::endl;
}

```

<ClassA_implVariant2.h> コンストラクタの定義 (バリエーション 2)

```

ClassA::ClassA() {
    std::cout<<(void*)this<<": Class A (Variant 2) created"<<std::endl;
}

ClassA::~ClassA() {
    std::cout<<(void*)this<<": Class A (Variant 2) deleted"<<std::endl;
}

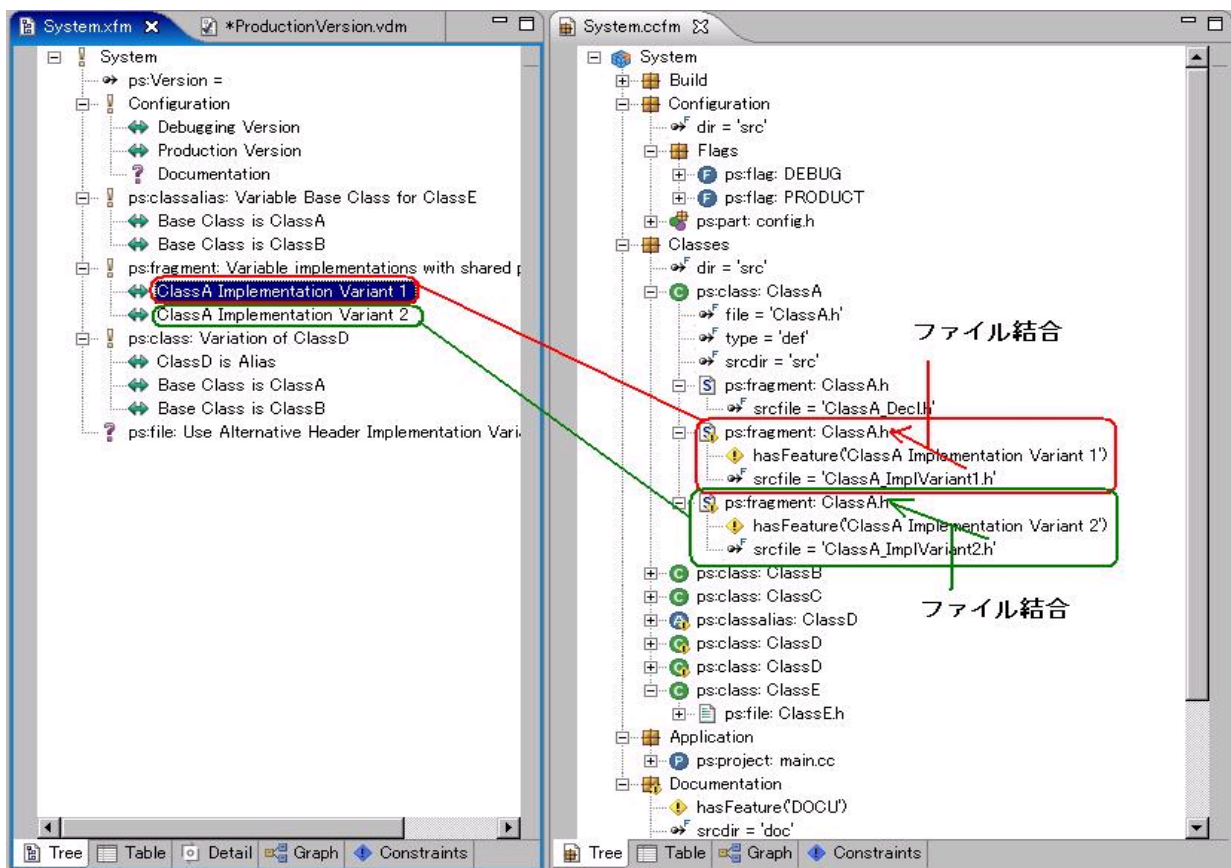
```

この例では、ファミリーモデル上に “ClassA_implVariant1.h” と “ClassA_implVariant2.h” を設定した “ps:fragments” という変数を作成し、フィーチャーモデル上の “ps:fragment: Variable implementations with shared parts” 配下のバリエーションの選択肢に関連付けを行います。そうすることで、フィーチャの選択に応じて結合されるファイルが切り替えられて、コンストラクタのポリモフィズムを実現しています。

<pure::variants の GUI 上の設定>

※ “ps:fragments” には更に、“file” アトリビュートとしてファイル (ClassA.h) を設定し、それに対して “srefile” アトリビュートとして設定したファイルを結合させています。

“ClassA.h” に対して、共通部分 “ClassA_Decl.h” の内容が書き込まれ、その後、“ClassA_implVariant1.h” または “ClassA_implVariant2.h” のどちらかが追加されることとなります。以下、2 種類のソースが出力されます。



```

1 class ClassA {↓
2 ↓
3 public:↓
4   ClassA();↓
5   ~ClassA();↓
6 ↓
7 }↓
8 ↓
9 ClassA::ClassA() {↓
10   std::cout<<(void*)this<<" : Class A (Variant 1) created"<<std::endl;↓
11 }↓
12 ↓
13 ClassA::~~ClassA() {↓
14   std::cout<<(void*)this<<" : Class A (Variant 1) deleted"<<std::endl;↓
15 }↓
16 ↓
17 [EOF]

```

バリエーション 1

共通部分

```

1 class ClassA {↓
2 ↓
3 public:↓
4   ClassA();↓
5   ~ClassA();↓
6 ↓
7 }↓
8 ↓
9 ClassA::ClassA() {↓
10   std::cout<<(void*)this<<" : Class A (Variant 2) created"<<std::endl;↓
11 }↓
12 ↓
13 ClassA::~~ClassA() {↓
14   std::cout<<(void*)this<<" : Class A (Variant 2) deleted"<<std::endl;↓
15 }↓
16 ↓
17 [EOF]

```

バリエーション 2

変動部分

3. 継承時の基本クラスを別名定義で管理する方法

クラス名が同じで、ベースクラスの異なる下記 2 つのソースファイルをそれぞれ変更し、フィーチャーの選択に応じて、ベースクラスを切り替える方法です。

<ソースファイル 1 >

```
class ClassA {
```

```
}
```

```
class ClassE : ClassA{
```

```
}
```

<ソースファイル 2 >

```
class ClassB {
```

```
}
```

```
class ClassE : ClassB{
```

```
}
```

<変更後>

```
class ClassA {
```

```
}
```

```
class ClassB {
```

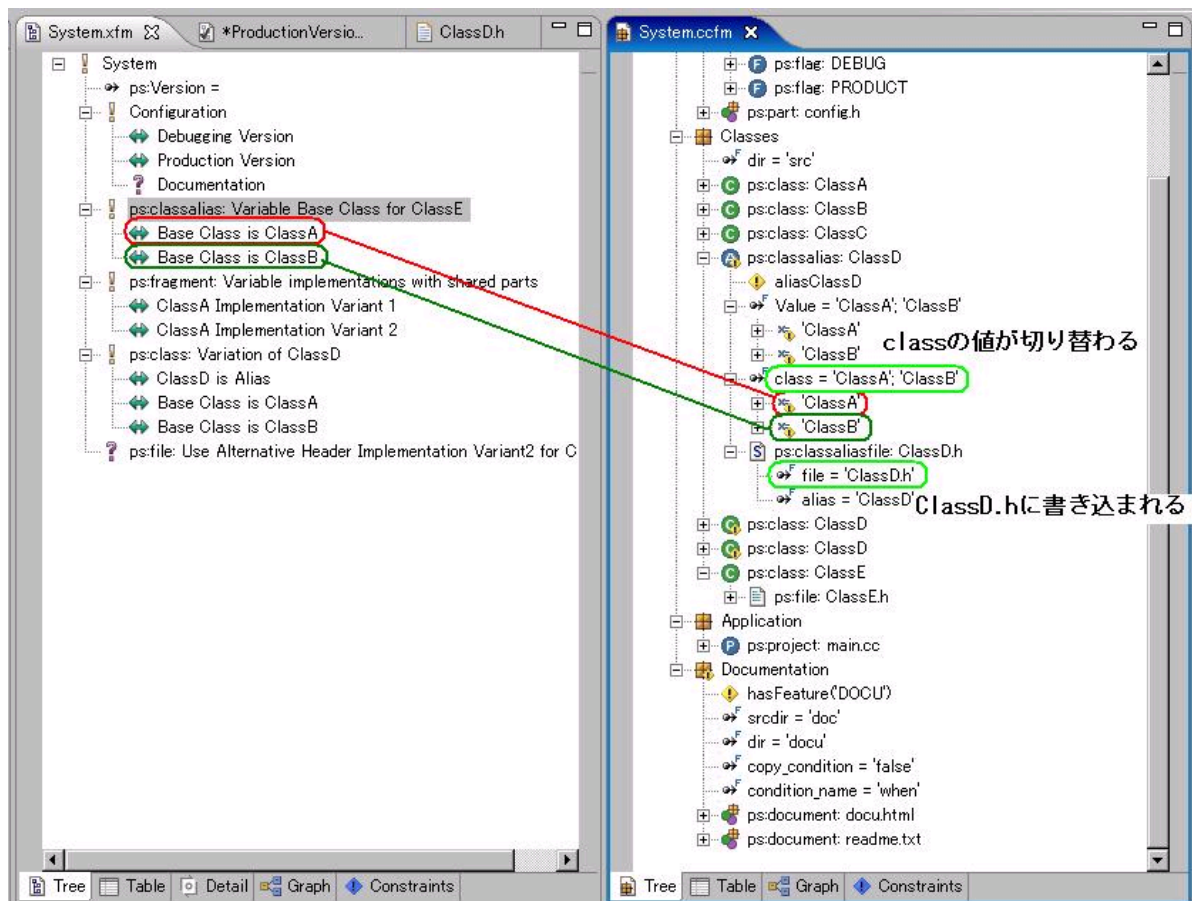
```
}
```

```
include "classD.h"
```

```
class ClassE : ClassD{
```

```
}
```

この例では、ファミリーモデル上に “ps:classalias: ClassD -> class” という変数を作成し、その値がフィーチャーモデル上の “ps:classalias: Variable Base Class for ClassE” 配下のバリエーションの選択で、ClassA と ClassB に切り替わるように設定しています。



<pure::variants の GUI 上の設定>

- ※ “ps:classalias” エレメントには、設定されたアトリビュートに応じて別名定義 (typedef) を生成する仕組みを設定しています。

ここでは、“ps:classalias: ClassD → ps:classaliasfile: ClassD.h” の “file” アトリビュートに設定されたファイル (ClassD.h) に、以下の構文を書き込みます。

```
構文 : typedef [ps:classalias->class] [ps:classalias->ps:classaliasfile->alias];
```

“ClassD.h” には、以下の 2 パターンが出力されます。

[パターン 1 : Base Class is ClassA を選択した場合]

```
typedef ::ClassA ClassD;
```

[パターン 2 : Base Class is ClassB を選択した場合]

```
typedef ::ClassB ClassD;
```

“ClassD.h” は、インクルードされているので、インクルードファイルを読み込むと、以下のようになります。

```
Base Class is ClassA
フィーチャーを選択
3 ↓
4 class ClassA {↓
5 ↓
6 }↓
7 ↓
8 class ClassB {↓
9 ↓
10 }↓
11 ↓
12 typedef ::ClassA ClassD;
13 ↓
14 class ClassE : ClassD{↓
15 ↓
16 }↓
17 [EOF]
```

```
Base Class is ClassB
フィーチャーを選択
3 ↓
4 class ClassA {↓
5 ↓
6 }↓
7 ↓
8 class ClassB {↓
9 ↓
10 }↓
11 ↓
12 typedef ::ClassB ClassD;
13 ↓
14 class ClassE : ClassD{↓
15 ↓
16 }↓
17 [EOF]
```

以上