3J-06

VDM++仕様から C#コードを生成するツールの開発

千坂　優佑†　　岡本　圭史†

仙台高等専門学校†

１．はじめに

近年，情報システムの遍在化・大規模化・複雑化に伴い，形式手法と呼ばれるシステム開発手法が注目を浴びている．形式手法の 1 つである形式仕様記述は，仕様を自然言語ではなく専用の形式仕様記述言語で記述することにより，仕様の厳密化や機械的な検証を可能とする手法である．

形式仕様記述の手法の 1 つに VDM がある[1]．VDM はライトウェイトな形式手法と呼ばれ，証明などを用いたより厳密な手法と比べて導入が容易であるため，実際の製品開発にも用いられている[2]．

VDM を扱うためのツールである VDMTools[1]には，形式仕様記述言語 VDM++で記述された仕様から Java または C++のコードを生成する機能がある[3]．同様に，Overture[4]にも Java のコードを生成する機能がある．この機能により，仕様を基にコードを作成する工数を削減するだけでなく，その過程でのヒューマンエラーの混入を予防できる．

しかし，現状のコード生成機能では Java と C++しか出力できず，また操作定義の事後条件は変換できないなどの制約もある．そこで本研究では，VDM++仕様[5]から C#コード[6]を生成するツール（以下，本ツール）を開発する．C#では.NET の CodeContracts[7]を用いることにより，VDM++仕様における事前条件・事後条件・不変条件を生成コード中に契約として残すことができ，さらにそれらを用いた検証ができる．なお，本ツール自体は C#で開発する．

２．変換の方法

２．１　概要

VDM++仕様から C#コードへの変換は，おおまかに以下の図の 3 段階に分けて行う．枠内は処理の内容を，矢印は入出力を表している．

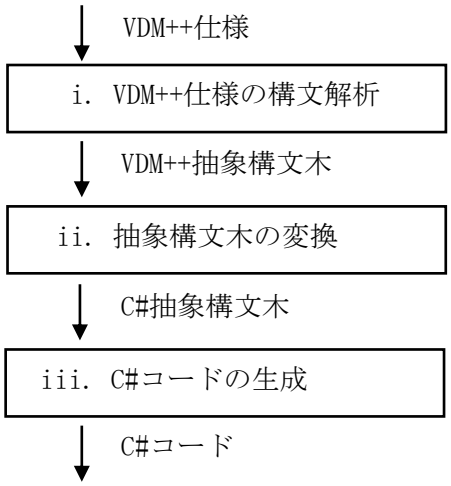

図１．変換の手順

i は VDM++仕様を VDM++の抽象構文木へと変換する処理である．実装には，既存のオープンソースソフトウェアである VDMJ[8]に含まれるパーサーを利用する．ただし VDMJ は Java で記述されているため，本ツールの実装においては C#コードへの変換を施して利用した．

ii は VDM++の抽象構文木を C#の抽象構文木へと変換する処理である．次節で詳しく説明する．

iii は C#の抽象構文木から C#コードを生成する処理である．実装には，.NET コンパイラープラットフォーム Roslyn[9]の機能を利用する．

２．２　抽象構文木の変換

VDM++仕様は 1 つ以上のクラスから成り，クラスは関数定義や値定義等の定義ブロックから成る．それらの項目ごとに，以下の表のように対応する C#の要素へと変換する．

表１．変換対応表

VDM++		C#
クラス		クラス
	関数定義	メソッド
	操作定義	メソッド
	値定義	フィールド
	インスタンス変数定義	フィールド

Development of VDM++ Specification to C# Code Generator
Yusuke CHISAKA†, Keishi OKAMOTO†
†National Institute of Technology, Sendai College

各定義ブロックには式や文が含まれうる．VDM++の式や文は C#の式や文へと変換する．VDM++の四則演算や if 式等の基本的な演算については，C#の四則演算や条件演算子等に変換する．VDM++の式中に含まれる限量式 forall や exists については，その束縛が集合束縛であれば，C#の LINQ のメソッドである All や Any に変換できる．

VDM++の関数定義および操作定義には，事前条件や事後条件を指定することができる．VDM++の事前条件 pre および事後条件 post は，それぞれ CodeContracts の Contract.Requires および Contract.Ensures に変換する．VDM++の事後条件において関数または操作の返り値を表すキーワード RESULT は，CodeContracts においてメソッドの返り値を表す Contract.Result に変換する．また，VDM++のインスタンス変数定義等における不変条件は，CodeContracts のクラス不変条件である Contract.Invariant に変換する．

VDM++では，関数定義および操作定義においてその実装を省略することができる（陰仕様）．陰仕様は，C#において未実装であることを表す NotImplemented 例外として変換する．

３．変換の結果

本ツールを用いて VDM++仕様を C#コードに変換した結果を以下に示す．上が VDM++仕様，下が本ツールを用いて生成した C#コードの一部である．

```
public static MySqrt: real -> real
MySqrt(x) ==
  innerMySqrt(x, x)
pre
  x >= 0
post
  NearlyEquals(RESULT * RESULT, x)
  and RESULT >= 0;
```

```
[Pure]
public static double MySqrt(double x)
{
    Contract.Requires(x >= 0);
    Contract.Ensures(NearlyEquals(
        Contract.Result<double>() *
        Contract.Result<double>(), x) &&
        Contract.Result<double>() >= 0);
    return innerMySqrt(x, x);
}
```

図２．関数定義の変換例（一部抜粋）

この例では，事前条件と事後条件を指定した関数定義の変換を行っている．

VDM++仕様に対して仕様アニメーションを用いたテストを行い，生成した C#コードに対しても同じ事前・事後条件を用いて単体テストを行ったところ，それぞれ同一の入力に対して同一の出力が得られ、同一のテスト結果が得られた．これにより，この例において本ツールが行った変換は妥当であると考えられる．

４．おわりに

本稿では，VDM++仕様から C#コードへの変換ツールを試作し，変換の妥当性を確認した．

今後は，VDM++のトークンやパターン等の本ツールによる変換処理が未実装の項目について検討し，実装およびその検証を行う．今後の課題として，本ツールを用いたシステム開発による実用性の確認や，VDM++仕様に修正があった場合に実装を維持しながら C#コード生成を行うこと等が挙げられる．

参考文献

[1] SCSK, VDM information web site, http://www.vdmtools.jp/

[2] T. Kurita and Y. Nakatsugawa (2009): "The Application of VDM++ to the Development of Firmware for a Smart Card IC Chip," Intl. Journal of Software and Informatics, Vol. 3, No. 2-3, pp.343-355.

[3] SCSK, The VDM++ to Java Code Generator, http://www.vdmtools.jp/uploads/manuals/javacgmanpp_a4E.pdf

[4] Overture Tool, http://overturetool.org/

[5] SCSK, The VDM++ Language Manual, http://www.vdmtools.jp/uploads/manuals/langmanpp_a4E.pdf

[6] ISO/IEC 23270:2003 C# Language Specification

[7] Microsoft Research, Contracts, http://research.microsoft.com/en-us/projects/contracts/

[8] Nick Battle, VDMJ, https://sites.google.com/site/nickbattle.

[9] GitHub, dotnet/roslyn, https://github.com/dotnet/roslyn