九州大学
KYUSHU UNIVERSITY

VDMTools

VDM++向け ダイナミックリンク機能

ver.1.0

How to contact:

http://fmvdm.org/	VDM information web site(in Japanese)
http://fmvdm.org/tools/vdmtools	VDMTools web site(in Japanese)
inq@fmvdm.org	Mail

VDM++向け ダイナミックリンク機能 1.0

— Revised for VDMTools v9.0.6

目 次

1 導入

本書は、*VDM++ Toolbox* ユーザーマニュアル [4] を拡張したものである。C++ [5] と *VDM C++* ライブラリ [1] についての知識が、前提とされている。本書を通して “DL クラス” ¯ (“ダイナミックリンククラス”) とは、外部コードが実行されるべき対象となる VDM++クラスを意味するものとして用いられる。

ダイナミックリンク機能の利用を始めるのに、このマニュアル全部を読む必要はない。機能の概要全体を捉えるのには、第 ¯ 2 章から始めよう。第 ¯ 3 章では、機能を利用した例題を示す (この例題の全ファイルが付録中にもおかれている)。第 ¯ 4 章では、C++コードジェネレータレータによって、どのように DL クラスからコード生成されるかを提示する。第 ¯ 5 章では、ダイナミックリンク機能を用いるときに含まれる個々の構成要素について述べる。

付録 ¯ A では、システム要求の詳細情報を提供する。

2 概観

この章では DL クラス利用の概観を述べる。ここの目的は、機能の用い方に自由な発想を与えることである。さらに詳細な情報については、第 ¯ 5 章の参照ガイドで示す。

取り上げ方を示す図解が 図 ¯ 1 である。この図からわかる通り、多くの様々な構成要素がこの機能が働くために準備されなければならない:

- VDM++ レベルでアクセスされるべき C++ クラス各々に対し、VDM++ レベルで DL クラスが 1 つ書かれていなければならない。
- C++で、インターフェイス層が 1 つ書かれている必要がある。これは、VDM++ レベルでメソッド呼び出しを外部コードに翻訳し、結果を *VDM C++* ライブラリ 型に翻訳する。これは共有ライブラリ (Windows 上では 1 つの `.dll`) として、コンパイルされるべきである .

これらの各構成要素について、以下でさらに詳細を述べる。

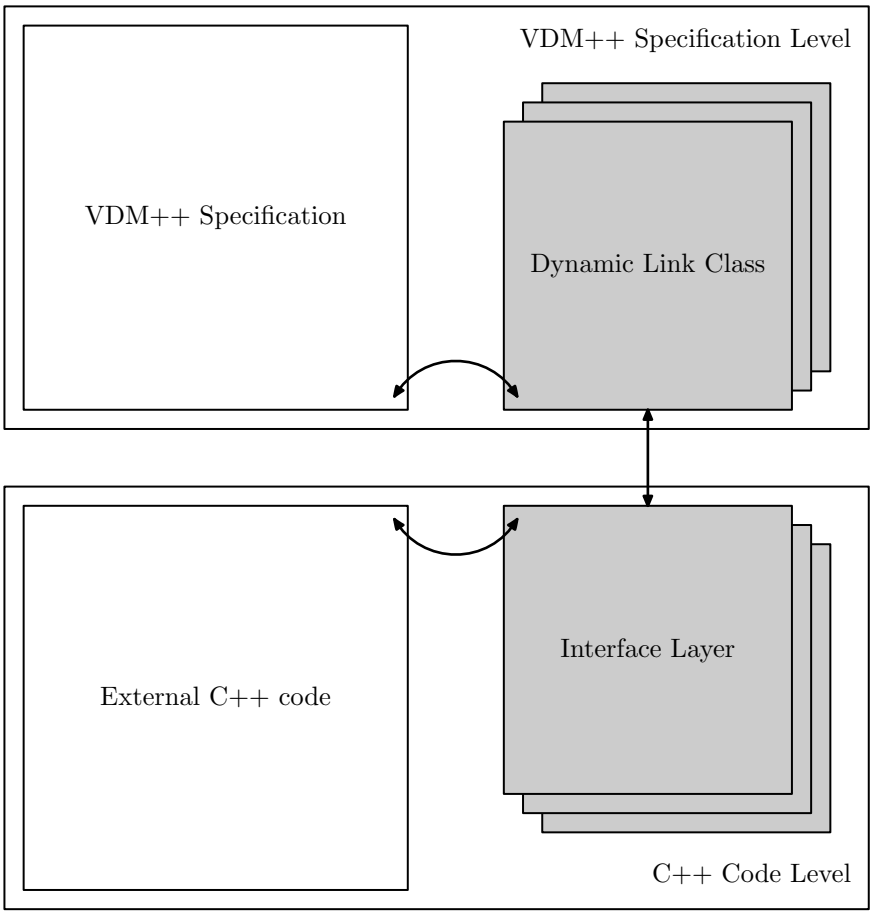

図 1: コードと VDM++ 仕様の連携

2.1 VDM++ における DL クラスの仕様

VDM++ において、 DL クラスとしてクラス仕様を定めるために、キーワード dlclass を class の代わりに用いるべきである。DL クラスには、DL クラスを実装する共有ライブラリの位置を直接示す命令が、含まれなければならない。これは uselib 命令を用いて指定される。このように、簡単な DL クラスは次のようになる

```
dlclass EmptyDLClass

uselib "EmptyDLClass.so"[1]

end EmptyDLClass
```

当たり前であるが、これはまったく機能をもたないため、興味対象となるクラスではない! 外部コードでメソッドが見えるようにするために、 is not yet specified という本体をもつメソッドが VDM++レベルで定義されるべきであろう。次は例題である:

```
dlclass SimpleDLClass

uselib "SimpleDLClass.so"

operations

public CallExternal : nat ==> nat
CallExternal (n) == is not yet specified;

end SimpleDLClass
```

したがって SimpleDLClass‘CallExternal の呼び出しは、共有ライブラリを通して、SimpleDLClass に相当する外部コードにリダイレクトされる。

[1]共有ライブラリは Unix 上でファイル型.so となるが、一方 Windows 上でそれらは .dll 型となることに注意したい

2.2 DL クラスとのインターフェイス

ユーザーは、 VDM++ DL クラスと DL クラスに対応する外部コードとの間に、インターフェイスを準備しなければならない。このインターフェイスはインターフェイス層として参照されるもので、 3 つの部分に分けられる:

1. 外部オブジェクト生成および削除のための関数が準備されていなくてはならない。これらの名称は各々 `DlClass_new` および `DlClass_delete` である;
2. メソッド呼び出しをリダイレクトする関数が定義されていなければならず、この名称は `DlClass_call` である;
3. 各外部クラスはクラス `DlClass` のサブクラスとして包含されていなければならない。つまり、すべての外部クラスは `DlClass` インターフェイスを実装しなければならない。

これらの関数に対するプロトタイプは、ファイル `dlclass.h` 中に定義されていて、 Toolbox 配布と共に提供される。これらは付録 ¯ B 中にも見つけることができる。

2.3 共有ライブラリの生成

インターフェイス層を生成するには、共有ライブラリが構築されていなければならない。インターフェイス層と外部のコードをコンパイルし、 1 つの共有ライブラリに入れることで、これが行われる。Unix 上では GNU g++を用いて、フラグ `-shared -fpic` を使用することでこれを行う。Windows 上では Microsoft Visual C++ を用いて、リンクのフラグ `/dll /incremental:no` を使用することでこれを行う。両ケースとも、VDM C++ ライブラリがリンクされていなければならないが、インターフェイス層がこのライブラリで定義された型を用いる必要があるからである。付録の ¯ D と ¯ E で提示される、Makefile 例題を参照しよう。

3 例題

この章では、DL クラスとインターフェイス層のコードをどのように書くかについて、小さな例題の一部を示す。例題は、任意の精度の数学的計算のための外部ライブラリ – “MAPM” – を基にする。このライブラリは、http://www.tc.umn.edu/∼ringx004/mapm-m から無料でダウンロード可能である。

VDM++ インタープリタは有限精度の計算能力を用いるため、無限精度の数学的対象を Toolbox で直接扱うと数学的値に妥協が生じるため、それは決して行わないことが重要である。

3.1 VDM++モデル

ここで、任意精度の整数を表すクラス BigInt を定義する。複数の BigInt を生成したり、１つの texttttBigInt に対しては文字列表現への変換を行い、さらに BigInt 上で加算を行うといった操作を、これが定義する。(SetVal 操作は唯一の値生成に用いられることに注意しよう。) このクラスは、 bigint_dl.so という名称のインターフェイス層と結合される。

```
dlclass BigInt

uselib "bigint_dl.so"

operations

public Create : nat ==> BigInt
Create(n) ==
( SetVal(n);
  return self
);

public toString : () ==> seq of char
toString() ==
  is not yet specified;
```

```
public plus : BigInt ==> BigInt
plus(i) ==
  is not yet specified;

protected SetVal : nat ==> ()
SetVal(n) ==
  is not yet specified;

end BigInt
```

さらに、仕様中では他のクラスの BigInt を用いることもできる。たとえば、クラス BankAccount は BigInt をあるインスタンス変数の型として用いる。Init と Deposit 操作で見てとれるように、このオブジェクトの操作は直接でなく操作を通してなされる。

```
class BankAccount

instance variables
  name : seq of char;
  number : nat;
  balance : BigInt

operations

public Init : seq of char * nat ==> ()
Init(nname, nnum) ==
( name := nname;
  number := nnum;
  balance := new BigInt().Create(nnum);
);

public Deposit : BigInt ==> ()
Deposit(bi) ==
  balance := balance.plus(bi);

public GetBalance : () ==> BigInt
```

```
GetBalance() ==
  return balance

end BankAccount
```

3.2 インターフェイス層

インターフェイス層は、VDM++ レベルの呼び出しを C++ 関数呼び出しにマップする。付録 ¯ C には完成ファイルが含まれる。ここでは選択的に抜粋を行う。

各 VDM++ BigInt オブジェクトに対して、相当する C++ BigIntDL オブジェクトが 1 つ存在する。これは 1 つの要素変数を通し、外部の MAPM ライブラリにリンクされる:

```
class BigIntDL : public DlClass
{

  MAPM val;

 public:
 ...
};
```

VDM++ レベルと C++ レベルの間でどのように情報伝達が介在するかを理解するために、 BigInt の列表現を返すクラス関数 toString を考えてみよう。

```
Sequence BigIntDL::toString()
{
#ifdef DEBUG
  cout << "BigIntDL::toString" << endl;
#endif //DEBUG
  char res[100];
  val.toIntegerString(res);
  return Sequence(string(res));
}
```

この関数は、ライブラリの独自の toIntegerString 関数を使用している。値は VDM++ Toolbox に返されるはずで、 1つの *VDM C++* ライブラリ 値がこの関数によって返されなければならないことに注意しよう。

VDM++ オブジェクトがパラメータとして渡されたり、あるいは結果として返されたりする場合に、さらに興味がある。この場合は、 *VDM C++* ライブラリクラスの DLObject が用いられるべきである。たとえば、加算の操作を考えよう:

```
DLObject BigIntDL::plus (Sequence &p)
{
#ifdef DEBUG
  cout << "BigIntDL::plus" << endl;
#endif //DEBUG
  DLObject result("BigInt", new BigIntDL);
  DLObject arg (p.Hd());
  BigIntDL *resPtr = (BigIntDL*) result.GetPtr(),
           *argPtr = (BigIntDL*) arg.GetPtr();
  resPtr->setVal( val + argPtr->getVal());
  return result;
}
```

ここで、操作の結果を保存するために、 BigInt のインスタンスとして新規に DLObject が生成される。DLObjects から BigIntDL オブジェクトへのポインタが抽出されるそのパラメータリストから、引数オブジェクトが抽出され、結果が計算される。最後に結果のオブジェクトが返される。

この方法で生成され返されるオブジェクトは、VDM++ Toolbox の通常のオブジェクト管理の下におかれる。このように、VDM++ レベルでの参照がなくなるまで続き、なくなった段階で それらオブジェクトに対し DlClass_delete が呼び出される。

4 C++ コードジェネレータと組み合わせた DL クラスの利用

インターフェイス層と共に生成されるコードと外部コードとの間で、仮想的に境なき利用が可能になるよう、DL クラスはコードジェネレータにより処理される。もちろん、is not yet specified と示される VDM++操作に対するユーザー実装は、インターフェイス層を通してよりはむしろこのように外部コードとの直接的な通信を許すことで、通常の方法で提供可能である。この章では、これら両方法を掲示する。

4.1 生成コードの利用

生成コードは、DL クラスで利用されるのと同じ呼び出し構造を用いる。これを許すために、各クラスが DL クラスへのポインタであるパブリック要素をもつ:

```
class vdm_BigInt : public virtual CGBase {
...
public:
  DlClass * BigInt_dlClassPtr;
}
```

DL クラスへの操作や関数の呼び出しに相当する C++関数呼び出しは、その後、このポインタを用いてコード生成される。たとえば、第 3 章における操作 toString を考えてみよう。これは BigInt_dlClassPtr を用いて、DlClass_call への呼び出しとしてコード生成されることになる。

```
#ifndef DEF_BigInt_toString

type_cL vdm_BigInt::vdm_toString () {  Sequence parmSeq_2;
  int success_3;

  return DlClass_call(BigInt_dlClassPtr, "toString", parmSeq_2, success_3);

}
```

```
#endif
```

このように生成コード内における vdm_BigInt::vdm_toString の呼び出しは、インターフェイス層によりスムーズに行えるようになるだろう。

オブジェクトの受け渡しを行う VDM++の関数や操作においては、状況は少し複雑になる。これは VDM++ コードジェネレータが、 C++ 関数の間で受け渡されたオブジェクトを示すために、 *VDM C++* ライブラリ から ObjectRef を利用するからである。したがって、インターフェイス中のコードでこの処理を行う必要がある。たとえば、以下のように BigIntDL::plus を修正することもある:

```
#ifdef CG
ObjectRef BigIntDL::plus (const Sequence &p)
#else
DLObject BigIntDL::plus (const Sequence &p)
#endif //CG
{
#ifdef DEBUG
  cout << "BigIntDL::plus" << endl;
#endif //DEBUG
  // Extract arguments
  BigIntDL *argPtr = GetDLPtr(p.Hd());

  // Set up result object
#ifdef CG
  ObjectRef result (new vdm_BigInt);
#else
  DLObject result("BigInt", new BigIntDL);
#endif

  BigIntDL *resPtr = GetDLPtr(result);

  // Perform manipulation on pointers, as needed for function
  resPtr->setVal( val + argPtr->getVal());

  return result;
}
```

この定義は関数 BigIntDL::GetDLPtr を有効に用いる:

```
#ifdef CG
BigIntDL *BigIntDL::GetDLPtr(const ObjectRef& obj)
#else
BigIntDL *BigIntDL::GetDLPtr(const DLObject& obj)
#endif //CG
{
#ifdef CG
  vdm_BigInt *objRefPtr = ObjGet_vdm_BigInt(obj);
  BigIntDL *objPtr = (BigIntDL*) objRefPtr->BigInt_dlClassPtr;
#else
  BigIntDL *objPtr = (BigIntDL*) obj.GetPtr();
#endif
  return objPtr;
}
```

4.2 ユーザー実装の提供

インターフェイス層を経て外部コードと通信するのではなく、生成コードが直接外部コードを呼び出すことができる。これは [3] に記述されるが、生成コードがコード置換を行う標準機構を利用する。

たとえば、加算操作を直接呼び出したいとする。このためには、vdm_BigInt::vdm_plus の実装が必要である。BigInt_userimpl.cc ファイル中に、この関数は定義されるはずである:

```
#include "bigint_dl.h"

type_ref_BigInt vdm_BigInt::vdm_plus (const type_ref_BigInt &vdm_i)
{
  type_ref_BigInt result (new vdm_BigInt);
  vdm_BigInt *vdm_r_ptr = ObjGet_vdm_BigInt(result),
             *vdm_i_ptr = ObjGet_vdm_BigInt(vdm_i);
  BigIntDL *argPtr = (BigIntDL*) vdm_i_ptr->BigInt_dlClassPtr,
```

```
            *thisPtr = (BigIntDL*) BigInt_dlClassPtr,
            *resPtr = (BigIntDL*) vdm_r_ptr->BigInt_dlClassPtr;
  resPtr->setVal(thisPtr->getVal() + argPtr->getVal());
  return result;
}
```

関数の適切な版がコンパイルされることを確認するため、`BigInt_userdef.h` ファイルは以下の内容を含む必要がある:

```
#define DEF_BigInt_USERIMPL
#define DEF_BigInt_plus
```

ファイルがコンパイルされると実行では、インターフェイス層に直行する代わりに、ユーザー提供関数を用いる。

5 参照ガイド

5.1 DL クラス定義

DL クラスは、構文を用いて仕様が示される

dlclass = 'dlclass', identifier, [inheritance clause]
uselib clause,
[class body],
'end' identifier ;

uselib clause = 'uselib', text literal ;

ここで定義されない構文クラスは、[2] で定義されていることに注意しよう。DL クラスは他のクラスとまったく同様に Toolbox で処理されるが、例外として、まだ本体仕様の示されない 関数や操作に対する呼び出しはいずれも、 uselib パスで指定された外部コードにリダイレクトされる。

5.2 uselib パス環境

仕様中で、ライブラリ名称は uselib オプションで与えられる。この場所の与え方についてはいくつかの方法が可能である:

- ライブラリに対する完全パス名称で与える (たとえば /home/foo/libs/libmath.so)。
- パスなしで、環境変数 VDM_DYNLIB でライブラリを検索するディレクトリ一覧を設定する。たとえば環境変数は、次のようになる: /home/foo/libs:/usr/lib:. (この変数設定で、カレントディレクトリを検索対象に含めたい場合は、この例題中にあるような '.' の指定が必要であることに注意しよう)。
- パスなしで VDM_DYNLIB 環境変数集合も与えられない。この場合は、ライブラリがカレントディレクトリ内に置かれている仮定がなされていることを意味する。

5.3 関数 DlClass_call

この関数は、VDM++ レベルのメソッド呼び出しを、外部コードレベルの適切なメソッド呼び出しにリダイレクトするために用いる。パラメータとして次を取り込む:

`DlClass* c` 呼び出しを行うオブジェクトへのポインタ;

`const char* name` 呼び出されているメソッドの名称;

`const Sequence& params` メソッド呼び出しに対するパラメータで、*VDM C++* ライブラリ 値として渡される;

`int& success` 成功フラグに対する参照パラメータ。このフラグを 0 に設定することは失敗; 1 に設定することは成功を示す。

関数はメソッド呼び出しの結果を、*VDM C++* ライブラリ Generic オブジェクトの形式で返す。

この関数の実装を行うことは、ユーザーの責任である。

5.4 関数 DlClass_delete

DL クラスのインスタンスが破棄される場合は、常にこの関数が呼び出される。Toolbox インタープリタは、参照カウンタを基にガーベージ収集構想を用いているため、この関数 DL クラスのインスタンスに対する参照カウンタが 0 になると、常に呼び出される。この関数はまた、オブジェクトが dlclose 呼び出し (第 5.9 章参照) の結果として破棄された場合にも、呼び出される。

この関数の実装を行うことは、ユーザーの責任である。

5.5 関数 DlClass_new

DL クラスのインスタンスが生成されるときは常に、この関数が呼び出される。インスタンス化しているクラスの名称を表す文字列をパラメータとして取り上げ、DlClass のインスタンスを指すポインタを返す。

この関数の実装を行うことは、ユーザーの責任である。

5.6 クラス DlClass

クラス DlClass は 1 つのメソッド: DlMethodCall に対してプロトタイプを指定する抽象クラスであって、 VDM++ レベルでのメソッド呼び出しを外部コードを呼び出すメソッドに変換するための特別なクラスで用いられる。Toolbox インタープリタに呼び出されるインターフェイスは C コードでなければならないため、 DlClass_call を経る回り道が必要であることに注意しよう。

VDM++ レベルでは影となる外部クラスに対してこのクラスのサブクラスの実装を提供することが、ユーザーの責任である。

5.7 DLObject クラス

DLObject は *VDM C++* ライブラリ の拡張であり、Toolbox インタープリタと外部コードとの間でのオブジェクトのやり取りを可能とする。各 DLObject は DL

クラスの１つのインスタンスに相当し、DlClass ポインタ含める。このクラスは以下のメソッドをもつ:

`DLObject(const string & name, DlClass *object)` object により与えられ DlClass へのポインタと共に初期設定される name で呼ばれる、DL クラスの DLObject を生成するコンストラクタ。

`DLObject(const DLObject & dlobj)` 既存のものから新しい DLObject を生成するコンストラクタ。

`DLObject(const Generic & dlobj)` Generic を縮小するために用いられるコンストラクタ。

`DLObject & operator=(const DLObject & dlobj)` このオブジェクトに dlobj の値を与える。

`DLObject & operator=(const Generic & gen)` このオブジェクトに gen の値を与える。

`string GetName() const` このオブジェクトが相当する VDM++ DL クラスの名称を与える。

`DlClass * GetPtr() const` DlClass の外部インスタンスに対するポインタを返す。

`DLObject::~DLObject()` デストラクタ。

5.8 ライブラリのオープン

Toolbox コマンド init が使われるときは常に、DL クラスに相当するすべての共有ライブラリがオープンされる。これらは、 dlclose コマンドが発行される (第 5.9 章参照) かまたは init が再呼び出しされるまで、オープンしたままである。後者の場合は、現時点でオープンしているライブラリはすべてクローズされ、その後に初期化の一部として再びオープンされる。

5.9 ライブラリのクローズ

Windows 上で dll は、他のアプリケーションによりロードされない場合、上書されるほかない。この目的のために、ロードされた dll をクローズするために、インタープリタウィンドウ (あるいは Toolbox コマンドライン版向けのコマンドプロンプト) で dlclose コマンドを使用することができる。

dlclose は、外部 C++ オブジェクトを削除した後に dll をクローズする。VDM++ オブジェクトは、 Toolbox において削除されることはないが、関連する外部 C++ オブジェクトはないという印は付けられる。その後もし現存の dlclass オブジェクト上で外部メソッドを発動させるという仕様を実行しようとすれば、実行時エラーを起こすことになるだろう。しかしながら、非 dlclass メソッド呼び出しは、dlclose コマンド後になされることも可能である。

おそらく dlclose の後および新しい dll をコピーした後は十中八九、 init を用いなければならなくなるであろう。

5.10 共有ライブラリの生成

共有ライブラリは、第 A 章で要求される適切なコンパイラを用いてコンパイルされなければならない。実行可能な共有ライブラリを生成するために、インターフェイス層が以下の要求項目を満たす必要がある:

- `dlclass.h` で定義されたプロトタイプの C 関数は、インターフェイス層で実装されなければならない;
- これに対し相当する DL クラスが存在するような外部コード中の各クラスは、`DlClass` のサブクラスとしてまとめられる必要があり、クラスメソッドである `DlMethodCall` が提供される必要がある。
- `dlclass.h` ヘッダーファイルは、配布の一部として提供されるものであるが、インターフェイス層コードをもつファイルに含まれていなければならないが、これが *VDM C++* ライブラリ の型指定関数のプロトタイプを含むからである。

使用する実際のコンパイラフラグについては、付録 ¯ D や ¯ E にある例題の makefile を参照しよう。

共有ライブラリは、Unix 上ではファイル拡張子として `".so"` を付けるべきであり、Windows 上では `.dll` となる。

参考文献

[1] CSK. *The VDM C++ Library*. CSK.

[2] CSK. *The VDM++ Language*. CSK.

[3] CSK. *The VDM++ to C++ Code Generator*. CSK.

[4] CSK. *VDM++ Toolbox User Manual*. CSK.

[5] Stroustrup, B. *The C++ Programming Language, 2nd edition*. Addison Wesley Publishing Company, 1991.

A システム要求

VDM++ Toolbox のダイナミックリンク特性を利用するために、ダイナミックリンク特性 (ライセンスファイル中に `vppdl` 特性をもつ行が存在しなければならない) を含めた Toolbox ライセンスが必要である。*VDM C++* ライブラリ と、統合されたコードから実行可能な共有ライブラリを生成するためのコンパイラが、共に要求される。

この特性は以下の組み合わせで動く:

- Microsoft Windows 2000/XP/Vista 上の Microsoft Visual C++ 2005 SP1
- Mac OS X 10.4, 10.5
- Linux Kernel 2.4, 2.6 上の GNU gcc 3, 4
- Solaris 10

共有ライブラリは 極めて コンパイラ依存であることに注意しよう。上記の組み合わせからの逸脱は、DL クラスを用いたときにたいがい実行時エラーを引き起こすことになる。

統合されたコードの実行可能な共有ライブラリを生成するために、コンパイラが要求される。

Toolbox 自身のインストールについては [4] の第 2 章を、*VDM C++* ライブラリのインストールについては [3] の第 2 章を参照のこと。

B dlclass.h

```
////////////////////////////////////////////////////////////////////////
// $Id: dlclass.h,v 1.4 2006/04/20 07:38:35 vdmtools Exp $
// dlclass.h
// Interface header file for VDM++ dlclass'es.
////////////////////////////////////////////////////////////////////////

#include "metaiv.h"

#ifndef _dlclass_h_
#define _dlclass_h_

#ifdef _MSC_VER
#define DLLFUN __declspec(dllexport)
#else
#define DLLFUN
#endif // _MSC_VER

#include <sstream>

class DlClass;

extern "C" {
  /**
   * This method is used to create a class.
   * @param name Name of the class as a String (plain class, no packages!)
   * @returns Class instance if successful, else 0
   */
  DLLFUN DlClass* DlClass_new (const wchar_t* name);

  /**
   * This method is used to delete a class
   * @param c C++ pointer to the class
   * @returns true, if successful, 0 if not
   * (anyway, we cannot do anything if an error occurs)
```

```
   */
  DLLFUN int DlClass_delete (DlClass* c);

  /**
   * Method used to call methods on a class.
   * @param c Pointer to the class instance
   * @param name name of the method to be called as a wstring.
   * @param params Parameters to the call as MetaIV-value
   * @success reference to int, that indicates if the call was
   * successful (boolean)
   */
  DLLFUN Generic DlClass_call (DlClass* c, const wchar_t * name, const Sequence &
}

/**
* Interface for the DLClass
* We could return void-pointers instead, but if
* all used DLClasses inherit from DlClass, the
* customer can use dynamic_cast
*/
class DlClass
{
public:
  virtual Generic DlMethodCall (const wchar_t * n, const Sequence & p) = 0;

  virtual ~DlClass() {};
};

#endif // _dlclass_h_
```

C 例題ファイル

C.1 VDM++層

```
dlclass BigInt

--uselib "bigint_dl.so" -- Unix
uselib "bigint_dl.dll"  -- Windows

operations

public Create : nat ==> BigInt
Create(n) ==
( SetVal(n);
  return self
);

protected SetVal : nat ==> ()
SetVal(n) ==
  is not yet specified;

public toString : () ==> seq of char
toString() ==
  is not yet specified;

public plus : BigInt ==> BigInt
plus(i) ==
  is not yet specified;

public minus : BigInt ==> BigInt
minus(i) ==
  is not yet specified;

public gt : BigInt ==> bool
gt(i) ==
  is not yet specified;
```

```
end BigInt

class BankAccount

instance variables
  name : seq of char;
  number : nat;
  balance : BigInt

operations

public Init : seq of char * nat ==> ()
Init(nname, nnum) ==
( name := nname;
  number := nnum;
  balance := new BigInt().Create(nnum);
);

public Deposit : BigInt ==> ()
Deposit(bi) ==
  balance := balance.plus(bi);

public Withdraw : BigInt ==> ()
Withdraw(bi) ==
  balance := balance.minus(bi)
pre balance.gt(bi);

public GetBalance : () ==> BigInt
GetBalance() ==
  return balance

end BankAccount

class A
```

```
operations

public Test : () ==> seq of char
Test() ==
( dcl b : BankAccount := new BankAccount();
  dcl bi : BigInt := new BigInt().Create(400);
  b.Init("customer", 100);

  b.Deposit(bi);
  return b.GetBalance().toString();
)

end A
```

C.2 bigint_dl.h

```
#ifndef _bigint_dl_h_
#define _bigint_dl_h_

#include "dlclass.h"
#include "m_apm.h"
#ifdef CG
#include "BigInt.h"
#endif //CG

class BigIntDL : public DlClass
{
  MAPM val;

 private:
#ifdef CG
  BigIntDL *GetDLPtr(const ObjectRef& obj);
#else
```

```
  BigIntDL *GetDLPtr(const DLObject& obj);
#endif //CG

 public:
  void setVal(MAPM);
  MAPM getVal();
  Generic DlMethodCall (const wchar_t* name, const Sequence &p);
  Generic SetVal (const Sequence &p);
#ifdef CG
  ObjectRef plus (const Sequence &p);
#else
  DLObject plus (const Sequence &p);
#endif //CG
  Sequence toString();
};

#endif //_bigint_dl_h_
```

C.3 bigint_dl.cc

```
#include "bigint_dl.h"

DlClass* DlClass_new (const wchar_t* name)
{
#ifdef DEBUG
  wcerr << L"DlClass_new called" << endl;
#endif //DEBUG
  if (!wcscmp(name, L"BigInt"))
    return new BigIntDL ();
  else
    return 0;
}

int DlClass_delete (DlClass* c)
```

```
{
#ifdef DEBUG
  wcerr << L"DlClass_delete called" << endl;
#endif //DEBUG
  try {
    delete c;
  } catch (...) {
    return 0;
  }
  return 1;
}

Generic DlClass_call (DlClass* c, const wchar_t* name, const Sequence& params, int
{
#ifdef DEBUG
  wcerr << L"DlClass_call: " << name << L"(" << params << L")" << endl;
#endif //DEBUG
  Generic result;
  try {
      result = c->DlMethodCall (name, params);
  } catch (...) {
    success = 0;
    return result;
  }
  success = 1;
  return result;
}

Generic BigIntDL::DlMethodCall (const wchar_t* name, const Sequence &p)
{
  Generic res;

  if (!wcscmp (name, L"SetVal"))
    res = this->SetVal(p);
  else if (!wcscmp(name, L"plus"))
    res = this->plus(p);
```

```
  else if (!wcscmp(name, L"toString"))
    res = this->toString();
  else {
      // the method does not exist
  }

  return res;
}

Generic BigIntDL::SetVal (const Sequence &p)
{
#ifdef DEBUG
  wcout << L"BigIntDL::SetVal" << endl;
#endif //DEBUG
  Int n(p.Hd());
  int nVal = n.GetValue();
  val = MAPM(nVal);
  return Generic();
}

#ifdef CG
BigIntDL *BigIntDL::GetDLPtr(const ObjectRef& obj)
#else
BigIntDL *BigIntDL::GetDLPtr(const DLObject& obj)
#endif //CG
{
#ifdef CG
  vdm_BigInt *objRefPtr = ObjGet_vdm_BigInt(obj);
  BigIntDL *objPtr = (BigIntDL*) objRefPtr->BigInt_dlClassPtr;
#else
  BigIntDL *objPtr = (BigIntDL*) obj.GetPtr();
#endif
  return objPtr;
}
```

```
#ifdef CG
ObjectRef BigIntDL::plus (const Sequence &p)
#else
DLObject BigIntDL::plus (const Sequence &p)
#endif //CG
{
#ifdef DEBUG
  wcout << L"BigIntDL::plus" << endl;
#endif //DEBUG
  // Extract arguments
  BigIntDL *argPtr = GetDLPtr(p.Hd());

  // Set up result object
#ifdef CG
  ObjectRef result (new vdm_BigInt);
#else
  DLObject result(L"BigInt", new BigIntDL);
#endif

  BigIntDL *resPtr = GetDLPtr(result);

  // Perform manipulation on pointers, as needed for function
  resPtr->setVal( val + argPtr->getVal());

  return result;
}

Sequence BigIntDL::toString()
{
#ifdef DEBUG
  wcout << L"BigIntDL::toString" << endl;
#endif //DEBUG
  char res[100];
  val.toIntegerString(res);
  return Sequence(string2wstring(string(res)));
```

```
}

void BigIntDL::setVal(MAPM newVal)
{
  val = newVal;
}

MAPM BigIntDL::getVal()
{
  return val;
}
```

D Unix上の Makeファイル

```
##-------------------------------------------------------------
##                    Make file for Linux
##-------------------------------------------------------------

OSTYPE=$(shell uname)

MAPMDIR = ./mapm_3.60
INCL    = -I../../cg/include -I${MAPMDIR}

VPPDE = ../../bin/vppde
CXXFLAGS = $(INCL)
DEBUG =-g
VDMLIB  = ../../cg/lib
CGFLAGS = -DCG #-DDEBUG

ifeq ($(strip $(OSTYPE)),Darwin)
OSV = $(shell uname -r)
OSMV = $(word 1, $(subst ., ,$(strip $(OSV))))
CCPATH = /usr/bin/
ifeq ($(strip $(OSMV)),12) # 10.8
CC      = $(CCPATH)clang++
CXX     = $(CCPATH)clang++
else
ifeq ($(strip $(OSMV)),11) # 10.7
CC      = $(CCPATH)clang++
CXX     = $(CCPATH)clang++
else
ifeq ($(strip $(OSMV)),10) # 10.6
CC      = $(CCPATH)g++
CXX     = $(CCPATH)g++
else
ifeq ($(strip $(OSMV)),9) # 10.5
CC      = $(CCPATH)g++-4.2
CXX     = $(CCPATH)g++-4.2
```

```
else
ifeq ($(strip $(OSMV)),8) # 10.4
CC      = $(CCPATH)g++-4.0
CXX     = $(CCPATH)g++-4.0
else
CC      = $(CCPATH)g++
CXX     = $(CCPATH)g++
endif
endif
endif
endif
endif

LIB     = -L. -L$(VDMLIB) -lCG -lvdm -lm -liconv

#ifeq ($(strip $(GCC_VERSION)),-3.3)
#MAPMLIB = lib_mapm_darwin_ppcg3.a
#else
MAPMLIB = lib_mapm_darwin.a
#endif
endif

ifeq ($(strip $(OSTYPE)),Linux)
CPUTYPE = $(shell uname -m)
CC      = /usr/bin/g++
CXX     = /usr/bin/g++
LIB     = -L. -L$(VDMLIB) -lCG -lvdm -lm
ifeq ($(strip $(CPUTYPE)),x86_64)
MAPMLIB = lib_mapm_linux_x86_64.a
else
MAPMLIB = lib_mapm.a
endif
endif

ifeq ($(strip $(OSTYPE)),SunOS)
CC      = /usr/sfw/bin/g++
```

```
CXX     = /usr/sfw/bin/g++
LIB     = -L. -L$(VDMLIB) -L../../lib -lCG -lvdm -lm
MAPMLIB = lib_mapm_solaris.a
endif

ifeq ($(strip $(OSTYPE)),FreeBSD)
CC      = /usr/bin/g++
CXX     = /usr/bin/g++
LIB     = -L. -L$(VDMLIB) -L../../lib -lCG -lvdm -lm -L/usr/local/lib -liconv
MAPMLIB = lib_mapm_freebsd.a
endif

OSTYPE2=$(word 1, $(subst _, ,$(strip $(OSTYPE))))
ifeq ($(strip $(OSTYPE2)),CYGWIN)
all: winall
else
all: cg.stamp
        $(MAKE) cgex bigint_dl.so
endif
        $(MAKE) CGEX

CGSOURCES = A.cc A_anonym.cc BankAccount.cc BankAccount_anonym.cc \
            BigInt.cc BigInt_anonym.cc CGBase.cc
CGOBJS = $(CGSOURCES:%.cc=%.o)

winall:
        make -f Makefile.win32

cgex:   cgex.o $(CGOBJS) bigint_cg.o $(MAPMDIR)/$(MAPMLIB)
ifeq ($(strip $(OSTYPE)),Darwin)
        ranlib $(MAPMDIR)/$(MAPMLIB)
endif
        $(CC) $(DEBUG) $(INCL) $(CGFLAGS) -o $@ $^ $(LIB)

cgex.o: cgex.cc A_userdef.h BankAccount_userdef.h cg.stamp
        $(CC) $(DEBUG) $(INCL) $(CGFLAGS) -c -o $@ cgex.cc
```

```
%.o: %.cc
	$(CC) $(DEBUG) $(CCFLAGS) $(INCL) -c -o $@ $^

%_shared.o: %.cc
	$(CC) $(DEBUG) -fPIC $(CCFLAGS) $(INCL) -c -o $@ $^

cg.stamp: bigint.vpp
	$(VPPDE) -c $^
	touch $@

jcg.stamp: bigint.vpp
	$(VPPDE) -j $^
	touch $@

%_userdef.h:
	touch $@

bigint_dl.so: bigint_dl.o ${MAPMDIR}/$(MAPMLIB)

bigint_dl.o: bigint_dl.cc
	$(CC) $(DEBUG) -fPIC $(CCFLAGS) $(INCL) -c -o $@ $^

bigint_cg.o: bigint_dl.cc
	$(CC) $(DEBUG) $(INCL) $(CGFLAGS) -c -o $@ $^

%.so:
ifeq ($(strip $(OSTYPE)),Darwin)
	ranlib ${MAPMDIR}/$(MAPMLIB)
	$(CXX) -dynamiclib -fPIC -o $@ $^ $(LIB)
else
	$(CXX) -shared -fPIC -o $@ $^ $(LIB)
endif

GENJAVAFILES = A.java BankAccount.java BigInt.java
```

```
CGEX.class: CGEX.java $(GENJAVAFILES)
CGEX: jcg.stamp CGEX.class $(GENJAVAFILES:%.java=%.class)

ifeq ($(strip $(OSTYPE)),Darwin)
%.class : %.java
        javac -J-Dfile.encoding=UTF-8 -encoding UTF8 -classpath ../../javacg/VDM.j
else
ifeq ($(strip $(OSTYPE2)),CYGWIN)
%.class : %.java
        javac -classpath "../../javacg/VDM.jar;." $<
else
%.class : %.java
        javac -classpath ../../javacg/VDM.jar:. $<
endif
endif

ifeq ($(strip $(OSTYPE)),Darwin)
jrun:   jcg.stamp CGEX
        java -Dfile.encoding=UTF-8 -classpath ../../javacg/VDM.jar:. CGEX
else
ifeq ($(strip $(OSTYPE2)),CYGWIN)
jrun:   jcg.stamp CGEX
        java -classpath "../../javacg/VDM.jar;." CGEX
else
jrun:   jcg.stamp CGEX
        java -classpath ../../javacg/VDM.jar:. CGEX
endif
endif

clean:
        rm -f bigint_dl.so bigint_dl.o bigint_cg.o
        rm -f cgex cgex.o cg.stamp $(CGOBJS) $(CGOBJS:%.o=%_shared.o)
        rm -f $(CGSOURCES)
        rm -f A.h A_anonym.h A_userdef.h
        rm -f BankAccount.h BankAccount_anonym.h BankAccount_userdef.h
        rm -f BigInt.h BigInt_anonym.h
```

```
rm -f CGBase.h
rm -f *.obj *.cpp *.exe *.exp *.lib
rm -f A.java BankAccount.java BigInt.java *.class *.java.bak jcg.stamp
```

E Windows上の Makeファイル

```
##-------------------------------------------------------------
##                    Make file for Windows 32bit
##             This Makefile can only be used with GNU make
##-------------------------------------------------------------

TBDIR = ../..
WTBDIR = ../..

#TBDIR = /cygdrive/c/Program Files/The VDM++ Toolbox v2.8
#WTBDIR = C:/Program Files/The VDM++ Toolbox v2.8

VPPDE = "$(TBDIR)/bin/vppde"

CC      = cl.exe
MAPMDIR = ./mapm_3.60
CGLIBS = "$(WTBDIR)/cg/lib/CG.lib" "$(WTBDIR)/cg/lib/vdm.lib"

CFLAGS   = /nologo /c /MD /W0 /EHsc /D "WIN32" /TP
INCPATH  = -I"$(WTBDIR)/cg/include" -I./mapm_3.60

LINK     = link.exe
LPATH    = /LIBPATH:"$(WTBDIR)/cg/lib"

WINMTLIBS = libcpmt.lib libcmt.lib
WINMDLIBS = msvcrt.lib msvcprt.lib
WINLIBS  = $(WINMDLIBS) \
        kernel32.lib user32.lib gdi32.lib comdlg32.lib \
        advapi32.lib shell32.lib uuid.lib winspool.lib \
        oldnames.lib comctl32.lib

LDFLAGS = /nologo /NODEFAULTLIB "$(MAPMDIR)/mapm.lib" $(CGLIBS) $(WINLIBS)

## DLL Specific binary, path and flag
DLLFLAGS = /D "_USRDLL"
```

```
DLL_LFLAGS = /nologo /dll /incremental:no /NODEFAULTLIB $(CGLIBS) #/DEBUG
DLLWINLIBS  = $(WINLIBS)

## CG Version Files

CGSOURCES = A.cpp A_anonym.cpp BankAccount.cpp BankAccount_anonym.cpp \
            BigInt.cpp BigInt_anonym.cpp CGBase.cpp
CGOBJ = A.obj A_anonym.obj BankAccount.obj BankAccount_anonym.obj BigInt.obj \
        BigInt_anonym.obj CGBase.obj
CGFLAGS = /D CG #-DDEBUG

## CG Version specific rules

all: cgex.exe bigint_dl.dll

cgex.obj: cg.stamp A_userdef.h BankAccount_userdef.h

cg.stamp: bigint.vpp
        $(VPPDE) <cg.script
        echo  >cg.stamp

cgex.exe: cgex.obj bigint_cg.obj $(CGOBJ)

bigint_cg.obj: bigint_dl.cpp
        $(CC) $(CFLAGS) $(CGFLAGS) $(INCPATH) bigint_dl.cpp -Fo"bigint_cg.obj"

A.obj: A.cpp
A_anonym.obj: A_anonym.cpp
BigInt.obj: BigInt.cpp
BigInt_anonym.obj: BigInt_anonym.cpp
BankAccount.obj: BankAccount.cpp
BankAccount_anonym.obj: BankAccount_anonym.cpp
CGBase.obj: CGBase.cpp

## DLL specific  rules
```

```
bigint_dl.obj: bigint_dl.cpp
        $(CC) $(CFLAGS) $(DLLFLAGS) $(INCPATH) /Fo"$@" $<

bigint_dl.dll: bigint_dl.obj $(MAPMDIR)/mapm.lib

## Rules

%.dll:
        $(LINK) $(DLL_LFLAGS) /out:"$@" $^  $(LPATH) vdm.lib $(DLLWINLIBS)

%.obj: %.cpp
        $(CC) $(CFLAGS) $(INCPATH) /Fo"$@" $<

%.exe: %.obj
        $(LINK) /OUT:$@ $^ $(LDFLAGS)

%.cpp: %.cc
        cp -f $^ $@

%_userdef.h:
        touch $@

clean:
        rm -f cg.stamp
        rm -f $(CGOBJ) bigint_cg.obj cgex.obj cgexe.exe
        rm -f bigint_dl.dll bigint_dl.obj
        rm -f *.cpp
        rm -f A.h A_anonym.h A_userdef.h
        rm -f BankAccount.h BankAccount_anonym.h BankAccount_userdef.h
        rm -f BigInt.h BigInt_anonym.h
        rm -f CGBase.h
        rm -f cgex.exe cgex.obj cgex.lib cgex.exp
        rm -f bigint_dl.lib bigint_dl.exp  bigint_dl.pdb
```