VDM++入門セミナー　演習問題

佐原伸

SCSK株式会社
ソリューション・機能事業部門
開発ソリューション事業本部
開発部
基盤開発課
VDM推進担当
2012年6月21日

要求概要

要求聞き出し(elicitation)の結果

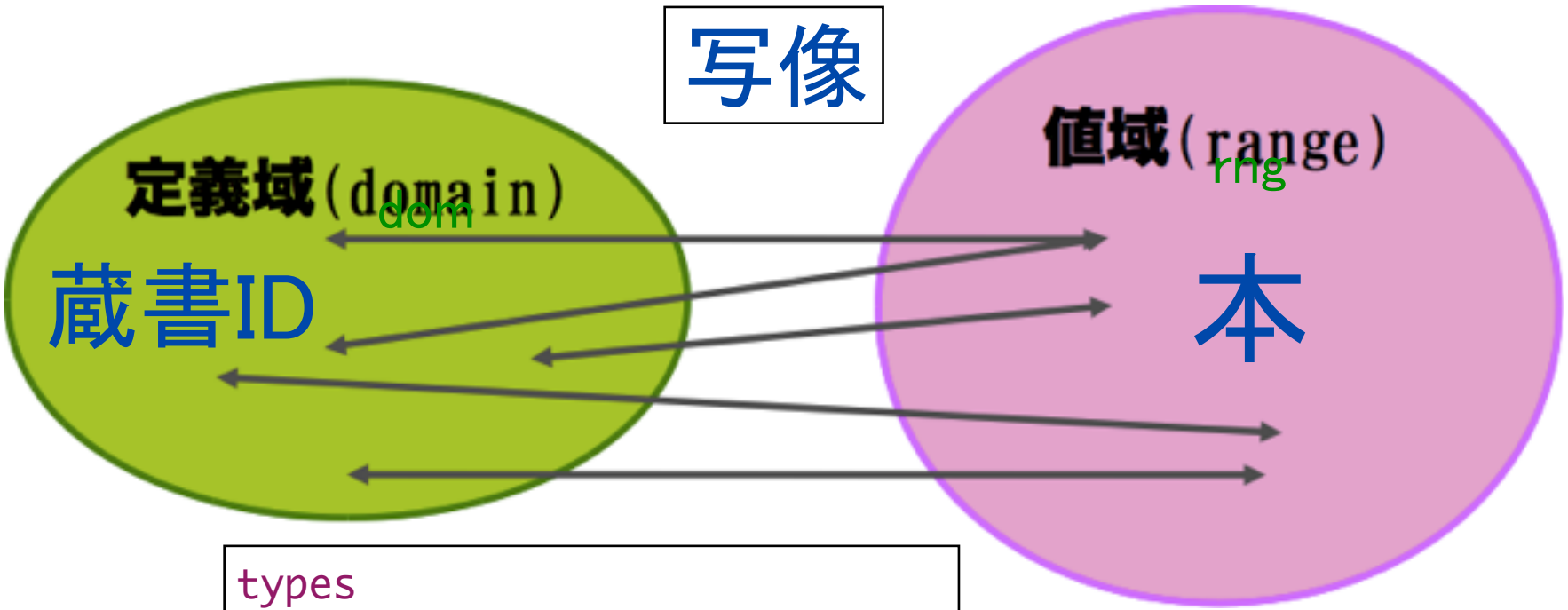
- 本を図書館の蔵書として追加、削除する。
- 題名と著者と分野のいずれかで本を検索する。
- 利用者が、蔵書を借り、返す。
- 蔵書の追加・削除や、利用者への貸出・返却は職員が行う。
- 利用者や職員の権限などは考慮しなくてよい。
- 最大蔵書数は10000、利用者一人への最大貸出冊数は3冊とする。

要求仕様 ver.0（非オブジェクトモデル）

- VDM++はオブジェクト指向仕様記述言語であるが、関数型言語の機能も持っているので、オブジェクト指向にこだわる必要はない
  - 一般に、関数型指向でモデル化した方が抽象度が高く、記述行数も少なくて済む
    - オブジェクト指向は、設計仕様以降に適用した方がよい
- 最初は、クラスをモジュールと考えて、事後条件・事前条件・不変条件だけを考えたモデルを作成してみる
  - このような仕様を、実行可能な陽仕様に対して、陰仕様と呼ぶ

要求仕様 ver.0 (Library0)

- 日本語仕様では、抽象的な意味の本と、図書館の蔵書として管理すべき個々の本が明確に区別されていない。
- 図書館では、同じ本を何冊か蔵書として持つことがあるので、以下のモデルでは、著者や題名という属性を持つ本に対し、
  - 蔵書を蔵書IDから本への写像とする。




```
types
public 蔵書 = map 蔵書ID to 本;
```

仕様作成に必要な型と値を定義せよ

- クラスは図書館0とし、そこで、すべての必要な型を定義する。
  - 本を図書館の蔵書として追加、削除する。
  - 題名と著者と分野のいずれかで本を検索する。
  - 利用者が、蔵書を借り、返す。
  - 蔵書の追加・削除や、利用者への貸出・返却は職員が行う。
  - 利用者や職員の権限などは考慮しなくてよい。
  - 最大蔵書数は10000、利用者一人への最大貸出冊数は3冊とする。

本を図書館の蔵書として追加、削除する、の陰仕様を作成せよ

```
class 図書館0

types
public 蔵書 = map 蔵書ID to 本;

operations
public 蔵書を追加する : 蔵書 ==> ()
蔵書を追加する(a追加本) == is not yet specified
pre ???
post ???;

public 蔵書を削除する: 蔵書 ==> ()
蔵書を削除する(a削除本) == is not yet specified
pre ???
post ???;

end 図書館0
```

1. ???に当てはめるべき日本語の文章を考えよ。
2. その日本語名を持つ関数のインタフェースを定義せよ。
3. 必要なインスタンス変数があれば定義せよ。
4. 関数の陽仕様を定義せよ。

本を検索する、の陰仕様を作成せよ

```
public 本を検索する: (題名 | 著者 | 分野) ==> 蔵書
本を検索する(a検索キー) ==  is not yet specified
pre
    ???
post
    ???;
```

本を貸す、の陰仕様を作成せよ

```
types
public 貸出 = map 利用者 to 蔵書;

instance variables
i貸出 : 貸出 := {|->};

operations
public 本を貸す: 蔵書 * 利用者 * 職員 ==> ()
本を貸す(a貸出本, a利用者, -) ==  is not yet specified
pre
    ???
post
    ???;
```

本を貸す、の事前条件と事後条件の関数を作成せよ

```
public 貸出可能である : 利用者 * 蔵書 * 貸出 * 蔵書 * nat1 +>
bool
貸出可能である(a利用者, a貸出本, a貸出, a図書館蔵書, a最大貸出数) ==
    蔵書に存在する(a貸出本, a図書館蔵書) and
    貸出に存在しない(a貸出本, a貸出) and
    最大貸出数を超えていない(a利用者, a貸出本, a貸出, a最大貸出数);

貸出に存在する : 蔵書 * 貸出 +> bool
貸出に存在する(a貸出本, a貸出) == ??? merge rng a貸出;

貸出に存在しない : 蔵書 * 貸出 +> bool
貸出に存在しない(a貸し出す本, a貸出) ==
    not 貸出に存在する(a貸し出す本, a貸出);
```

本を返す、の陰仕様を作成せよ

```
operations
public 本を返す: 蔵書 * 利用者 * 職員 ==> ()
本を返す(a返却本, a利用者, -) ==  is not yet
specified
pre
    ???
post
    ???;
```

要求仕様 ver.0 (Library0)

- 以上で作成した陰仕様は、要求仕様として、必要十分である。
  - しかし、要求仕様の正当性検証は静的チェックしか行えない
  - また、妥当性検証は事後条件をレビューするしかない
- VDMは本来段階的洗練(refinement)を使って、事後条件から手続き的仕様に証明しながら変換して、正当性検証を行う
  - しかし、現場で証明を行うのは、技術的。工数的に難しいため、VDMToolsやOverturetoolといったVDMの処理系では、仕様を実行して検証することを行っている。
  - そこで、陰仕様モデルの次に、陽仕様モデルを作成し、動的チェックを行うことで、正当性検証と妥当性検証を行う。

要求仕様 ver.1 (Library1)

- 図書館0クラス(Library0.vpp)の陰仕様(下請け関数は陽仕様)を元に、陽仕様を作成せよ。本を返す、は下記の通り。
  - 本を検索する、の操作本体には写像内包式を使うと良い。
  - 利用者が既に本を借りている場合と、そうでない場合に場合分けする必要がある。

```
public 本を返す: 蔵書 * 利用者 * 職員 ==> ()
本を返す(a返却本, a利用者, -) ==
    let w利用者への貸出本 = i貸出(a利用者),
        w利用者への貸出本new = dom a返却本 <-: w利用者への貸出本 in
    if w利用者への貸出本new = {|->} then
        i貸出 := {a利用者} <-: i貸出
    else
        i貸出 := i貸出 ++ {a利用者 |-> w利用者への貸出本new}
pre 貸出に存在する(a返却本, i貸出)
post 貸出から削除されている(a返却本, i貸出);
```

要求仕様 ver.1 (Library1)

```
public 蔵書を得る : () ==> 蔵書
蔵書を得る() == return i蔵書;

public 貸出を得る : () ==> 貸出
貸出を得る() == return i貸出;
```

要求仕様 ver.1 (Library1) 仕様の実行テスト

- 検証は仕様の内部矛盾をチェックするだけであり、ユーザーの要求に合致するかという妥当性のチェックはできない。
- そこで、仕様実行を使用した実行可能仕様の検証と妥当性チェックが必要になる。

要求仕様 ver.1 (Library1) 回帰テスト

- 回帰テスト・ライブラリー
  - VDMUnit.vpp
- 回帰テストソース参照
  - MyTest.vpp
  - MyTestCase.vpp
- 回帰テスト実行
  - debug new TestApp().run()

MyTest.vpp

```
class TestApp
operations
public run : () ==> ()
run() == (
      dcl ts : TestSuite := new TestSuite("図書館貸し出しモデルの回帰テスト\n"),
          tr : TestResult := new TestResult();
      tr.addListener(new PrintTestListener());
      ts.addTest(new TestCaseUT0001("TestCaseUT0001:\tノーマルケースの単体テスト"));
      ts.addTest(new TestCaseUT0002("TestCaseUT0001:\tエラーケースの単体テスト"));
      ts.run(tr);
      if tr.wasSuccessful() = true then
        def - = new IO().echo("*** 全回帰テストケース成功。 ***") in skip
      else
        def - = new IO().echo("*** 失敗したテストケースあり!! ***") in skip
);

end TestApp
```

MyTestase.vpp その1

```
class TestCaseComm is subclass of TestCase

operations
public print : seq of char ==> ()
print(a文字列) ==
    let - = new IO().echo(a文字列) in skip;
end TestCaseComm

class TestCaseUT0001 is subclass of TestCaseComm
operations

public TestCaseUT0001: seq of char ==> TestCaseUT0001
TestCaseUT0001(name) == setName(name);

public test01: () ==> ()
test01 () == (
    let w図書館 = new 図書館1(),
```

MyTestase.vpp その2

```
        分野1 = "ソフトウェア",
        分野2 = "工学",
        分野3 = "小説",
        著者1 = "佐原伸",
        著者2 = "井上ひさし",
        本1 = mk_図書館1`本("デザインパターン",著者1,{分野1,分野2}),
        本2 = mk_図書館1`本("紙屋町桜ホテル",著者2,{分野3}),
        蔵書1 = {mk_token(101) |-> 本1},
        蔵書2 = {mk_token(102) |-> 本1},
        蔵書3 = {mk_token(103) |-> 本2},
        蔵書4 = {mk_token(104) |-> 本2},
        利用者1 = mk_token("利用者1"),
        利用者2 = mk_token("利用者2"),
        職員1 = mk_token("職員1")
    in (
        w図書館.蔵書を追加する(蔵書1);
```

MyTestase.vpp その3

```
    w図書館.蔵書を追加する(蔵書2);
    w図書館.蔵書を追加する(蔵書3);
    w図書館.蔵書を追加する(蔵書4);
    assertTrue("test01 蔵書の追加がおかしい。",
        w図書館.蔵書を得る() = {mk_token(101) |-> 本1, mk_token(102) |-> 本1,
                              mk_token(103) |-> 本2, mk_token(104) |-> 本2}
    );
    let w見つかった本1 = w図書館.本を検索する("井上ひさし"),
        w見つかった本2 = w図書館.本を検索する("工学")
    in
    assertTrue("test01 本の検索がおかしい。",
        w見つかった本1 = {mk_token(103) |-> 本2, mk_token(104) |-> 本2} and
        w見つかった本2 = {mk_token(101) |-> 本1, mk_token(102) |-> 本1}
    );
```

MyTestase.vpp その4

```
    w図書館.本を貸す({mk_token(101) |-> 本1}, 利用者1, 職員1);
    w図書館.本を貸す({mk_token(103) |-> 本2}, 利用者1, 職員1);
    w図書館.本を貸す({mk_token(104) |-> 本2}, 利用者2, 職員1);
    assertTrue("test01 本の貸出がおかしい。",
        let w貸出 = w図書館.貸出を得る() in
        w貸出 = {利用者1 |-> {mk_token(101) |-> 本1, mk_token(103) |-> 本2},
                 利用者2 |-> {mk_token(104) |-> 本2}}
    );
```

MyTestase.vpp その4

```
        w図書館.本を返す({mk_token(103) |-> 本2}, 利用者1, 職員1);
        w図書館.本を返す({mk_token(104) |-> 本2}, 利用者2, 職員1);
        assertTrue("test01 本の返却がおかしい。",
            let w貸出 = w図書館.貸出を得る() in
            w貸出 = {利用者1 |-> {mk_token(101) |-> 本1}}
        );
        w図書館.蔵書を削除する({mk_token(104) |-> 本2});
        assertTrue("test01 蔵書の削除がおかしい。",
            w図書館.蔵書を得る() =
            {mk_token(101) |-> 本1, mk_token(102) |-> 本1,
             mk_token(103) |-> 本2}
        );
    )
);

end TestCaseUT0001
```

MyTestase.vpp その5（想定したエラー発生）

```
class TestCaseUT0002 is subclass of TestCaseComm
operations

public TestCaseUT0002: seq of char ==> TestCaseUT0002
TestCaseUT0002(name) == setName(name);

public test01: () ==> ()
test01 () == (
    trap <RuntimeError> with
                print("\ttest01 意図通り、最大貸出数を超えたエラー発生。\n")
    in
    let w図書館 = new 図書館1(),
```

MyTestase.vpp その6（想定したエラー発生）

```
        分野1 = "ソフトウェア",
        分野2 = "工学",
        分野3 = "小説",
        著者1 = "佐原伸",
        著者2 = "井上ひさし",
        本1 = mk_図書館1`本("デザインパターン",著者1,{分野1,分野2}),
        本2 = mk_図書館1`本("紙屋町桜ホテル",著者2,{分野3}),
        蔵書1 = {mk_token(101) |-> 本1},
        蔵書2 = {mk_token(102) |-> 本1},
        蔵書3 = {mk_token(103) |-> 本2},
        蔵書4 = {mk_token(104) |-> 本2},
        利用者1 = mk_token("利用者1"),
        職員1 = mk_token("職員1")
    in (
        w図書館.蔵書を追加する(蔵書1);
        w図書館.蔵書を追加する(蔵書2);
        w図書館.蔵書を追加する(蔵書3);
        w図書館.蔵書を追加する(蔵書4);
```

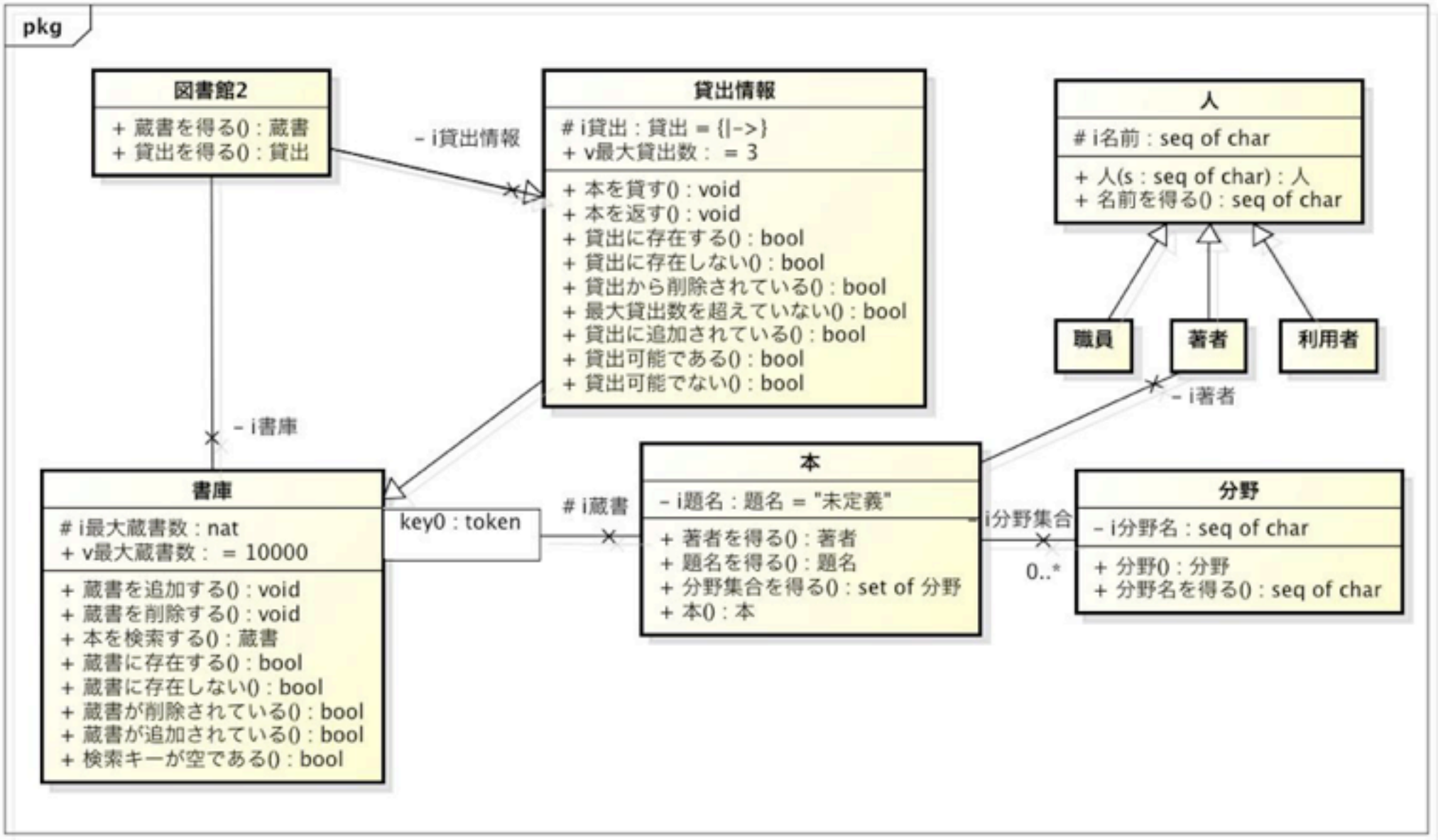
要求仕様 ver.1 (Library1) 回帰テスト

- VPP-Intro.pdfファイルのコードカバレージ情報を見て、テストされていない仕様を実行するノーマルテストケースとエラーテストケースを、各1件追加して回帰テストを実行せよ。

要求仕様 ver.2 (Library2) オブジェクトモデル

- ソースを参照のこと
  - Library2.vpp
  - ファイル名に2が付いた.vdmppファイル
  - BookStacks.vdmpp
  - 回帰テストは、MyTest2.vppとMyTestCase2.vpp
    - 回帰テスト実行
      - debug new TestApp2().run()

要求仕様 ver.2 (Library2) オブジェクトモデル



要求仕様 ver.2 (Library2) オブジェクトモデル

- オブジェクトモデルの問題点を指摘せよ。

演習終了