航空管制システム

(Revision : 0.1 – 2017 年 2 月 23 日)

佐原　伸

概要

VDM++ 言語 [2] 入門セミナーの演習問題で、航空管制システムのモデル作成例。

目次

1　ATC 要求辞書

私は、航空管制 (ATC = Air Traffic Control) の英語要求辞書である。英語の用語 (型名、関数名、操作名などの名称) を定義している。

明確になった自然言語によるシステムの要求は以下のとおりである。

R1　Only on-duty controllers can control airspaces.
R2　An airspace can be activated only if it is commissioned.
R3　All utilised airspaces in the ATC region are activated.
R4　The capacity of each utilised airspace is not exceeded.
R5　A controller cannot be assigned to two different airspaces simultaneously.
R6　Each known aircraft has a unique controller.
R7　Each known aircraft occupies a unique airspace.
R8　Each activated airspace has a unique controller.
R9　An airspace which is not activated contains no aircraft.
R10　The controller of a known aircraft is on duty.

```
.....................................................................................
class
ATC 英語要求辞書
types
  public Controller = token;
  public Space = token;
  public Aircraft = token;
public
  ATC::onduty : set of Controller
        control : inmap Space to Controller
        capacity : map Space to nat
        location : map Aircraft to Space
  inv mk_ATC(cs,con,cap,loc) ==
      rng con subset cs and
      dom con subset dom cap and
      rng loc subset dom con and
       forall s in set rng loc & numOfAircraft(s,loc) <= cap(s);
  public properties = <commissioned> | <activated> | <deactivated> | <utilised> | <assigned> | <availa
instance variables
  public sOnduty : set of Controller := {};
  public sControl : inmap Space to Controller := { |-> };
  public sCapacity : map Space to nat := { |-> };
  public sLocation : map Aircraft to Space := { |-> };
```

```
    functions
    public
      numOfAircraft : Space * map Aircraft to Space -> nat
      numOfAircraft (s, loc) ==
        card dom (loc :> {s});
    public
      isActivated : Space * ATC -> bool
      isActivated (s, δ) ==
        s in set dom (δ.control);
    public
      isKnown : Aircraft * ATC -> bool
      isKnown (p, δ) ==
        p in set dom (δ.location);
    public
      controllerOf : Aircraft * ATC -> Controller
      controllerOf (p, δ) ==
        δ.control (δ.location (p))
      pre isKnown (p, δ)
    operations
    public
      Commission (s : Space, n : nat) ==
        sCapacity := sCapacity ++ {s |-> n}ext wr sCapacity : map Space to nat

    pre s not in set dom sCapacity
    post sCapacity = sCapacity~ ++ {s |-> n} ;
    public
    ResetCapacity (s : Space, n : nat) ==
      sCapacity := sCapacity ++ {s |-> n}ext rd sLocation : map Aircraft to Space
        wr sCapacity : map Space to nat

  pre s in set dom sCapacity and numOfAircraft (s, sLocation) <= n
  post sCapacity = sCapacity~ ++ {s |-> n} ;
    public
  Decommission (s : Space) ==
    sCapacity := {s} <-: sCapacityext rd sControl : inmap Space to Controller
      wr sCapacity : map Space to nat

pre s in set (dom sCapacity \ dom sControl)
post sCapacity = {s} <-: sCapacity~ ;
```

```
            public
        ClockOn(c : Controller) ==
          sOnduty := sOnduty union {c}ext wr sOnduty : set of Controller

      pre  c not in set sOnduty
      post sOnduty = sOnduty~ union {c} ;
            public
      ClockOff(c : Controller) ==
        sOnduty := sOnduty \ {c}ext wr sOnduty : set of Controller
          rd sControl : inmap Space to Controller

    pre  c in set sOnduty \ rng sControl
    post sOnduty = sOnduty~ \ {c} ;
            public
    Activate(s : Space, c : Controller) c : Controller ==
      (   sControl := sControl ++ {s |-> c};
          return c
      ) ext rd sOnduty : set of Controller
        wr sControl : inmap Space to Controller
        rd sCapacity : map Space to nat

  pre  s in set (dom sCapacity \ dom sControl) and
       rng sControl <> sOnduty
  post sControl = sControl~ ++ {s |-> c} ;
            public
  Reassign(s : Space) c : Controller ==
    (   sControl := sControl ++ {s |-> c};
        return c
    ) ext rd sOnduty : set of Controller
      wr sControl : inmap Space to Controller

pre  s in set dom sControl and
     rng sControl <> sOnduty
post sControl = sControl~ ++ {s |-> c} and
      c <> sControl~(s) ;
            public
Deactivate(s : Space) ==
  sControl := {s} <-:  sControlext wr sControl : inmap Space to Controller
    rd sLocation : map Aircraft to Space
```

```
pre  s in set (dom sControl \ rng sLocation)
post sControl = {s} <-:  sControl~ ;
                public
AddFlight (p : Aircraft, s : Space) ==
  sLocation := sLocation ++ {p |-> s}ext rd sControl : inmap Space to Controller
    rd sCapacity : map Space to nat
    wr sLocation : map Aircraft to Space

e  s in set dom sControl and
  p not in set dom sLocation and
  numOfAircraft (s, sLocation) < sCapacity (s)
st sLocation = sLocation~ ++ {p |-> s} ;
                public
ndover (p : Aircraft, s : Space) ==
sLocation := sLocation ++ {p |-> s}ext rd sControl : inmap Space to Controller
  rd sCapacity : map Space to nat
  wr sLocation : map Aircraft to Space

 s in set dom sControl and
p in set dom sLocation and
sLocation (p) <> s and
numOfAircraft (s, sLocation) < sCapacity (s)
 sLocation = sLocation~ ++ {p |-> s} ;
                public
veFlight (p : Aircraft) ==
ocation := {p} <-:  sLocationext wr sLocation : map Aircraft to Space

 in set dom sLocation
sLocation = {p} <-:  sLocation~
                end
```

ATC 英語要求辞書

.....

2　航空管制システム

私は、航空管制 (ATC = Air Traffic Control) の要求辞書である。英語と対応する日本語の用語 (型名、関数名、操作名などの名称) を定義している。

明確になった自然言語によるシステムの要求は以下のとおりである。

R1　当直の管制官だけが空域を管制できる。
R2　空域は、委託中の場合にのみ活性にすることができます。
R3　ATC 領域内のすべての利用中の空域が活性です。
R4　利用中の各空域の許容数を超えない。
R5　管制官を 2 つの異なる空域に同時に割り当てることはできない。
R6　各既知の航空機には独自の管制官がいる。
R7　各既知の航空機は、固有の空域を占有します。
R8　活性化された各空域には固有の管制官がいる。
R9　活性化されていない空域に航空機はいない。
R10　既知の航空機の管制官は当直である。

.....

class
航空管制システム is subclass of ATC 英語要求辞書
types

```
c 管制官 = Controller;
c 空域 = Space;
c 航空機 = Aircraft;
c 当直集合 = set of 管制官;
c 管制 = inmap 空域 to 管制官;
c 許容数 = map 空域 to nat;
c 位置 = map 航空機 to 空域;
```

.....

以下の用語は、機能と安全性の特性を表す。

- 委託中: 空域は、それが ATC 領域の一部であれば委託中であると言われる。この用語は、ここで意図されていない意味を含む正式な法的用語である「制御された空域」よりも好ましい（例えば、航空機が公的な許可なしに空域に入ることはできない)。本仕様書の目的のために、空域は、その許容数が分かっている場合に限り、委託中とする。
- 活性: 空域は、管制官が割り当てられている場合に活性化されるという。
- 非活性: 空域は、委任されているが、割り当てられた管制官がいない場合、非活性化されていると言われます。例えば、ある空港が全体的または部分的に空港に関係する空域は、夜間に空港が閉鎖されたときに非活性化することがある。
- 利用中: 空域は、1 つ以上の航空機が占有している場合に利用中という。空域は活性化されていても現

在は利用されていないかもしれないが、その逆は無い。

- 割当中: 特定の空域の現在の管制官は、その空域に割当中という。管制官は、ATC 領域内のある空域の管制官であれば、単純に割当中にできる。
- 担当可能: 当直で現在割り当てられていない管制官が担当可能であるという。
- 既知: 航空機は、ATC 領域の一部の空域を利用していると既知という。

```
c 機能安全性特性 = <委託中> | <活性> | <非活性> | <利用中> | <割当中> | <担当可能> | <既知>
                instance variables
c s 当直集合 : 当直集合 := {};
c s 管制 : 管制 := { |-> };
c s 許容数 : 許容数 := { |-> };
c s 位置 : 位置 := { |-> };
ng s 管制 subset s 当直集合 and
om s 管制 subset dom s 許容数 and
ng s 位置 subset dom s 管制 and
orall s 空域 in set rng s 位置 & 航空機数(s 空域, s 位置) <= s 許容数(s 空域)

                operations
                public
制システム : 当直集合 * 管制 * 許容数 * 位置 ==> 航空管制システム
制システム(a 当直集合, a 管制, a 許容数, a 位置) == atomic
 当直集合 := a 当直集合;
s 管制 := a 管制;
s 許容数 := a 許容数;
s 位置 := a 位置

                public
トを追加する(a 航空機 : 航空機, a 空域 : 空域) ==
置 := s 位置 ++ {a 航空機 |-> a 空域}
 空域 in set dom s 管制 and
航空機 not in set dom s 位置 and
空機数(a 空域, s 位置) < s 許容数(a 空域)
s 位置 = s 位置~ ++ {a 航空機 |-> a 空域};
                public
トを削除する(a 航空機 : 航空機) ==
置 := {a 航空機} <-: s 位置
 航空機 in set dom s 位置
s 位置 = {a 航空機} <-: s 位置~;
                public
```

```
切り替える(a 航空機 : 航空機, a 空域 : 空域) ==
置 := s 位置 ++ {a 航空機 |-> a 空域 }
 空域 in set dom s 管制 and
航空機 in set dom s 位置 and
位置(a 航空機) <> a 空域 and
空機数(a 空域, s 位置) < s 許容数(a 空域)
s 位置 = s 位置~ ++ {a 航空機 |-> a 空域 } ;
                public
る : 空域 * nat ==> 許容数
る(a 空域, n) ==
s 許容数 := s 許容数 ++ {a 空域 |-> n};
return s 許容数

 空域 not in set dom s 許容数
s 許容数 = s 許容数~ ++ {a 空域 |-> n} ;
                public
を設定し直す : 空域 * nat ==> 許容数
を設定し直す(a 空域, n) ==
s 許容数 := s 許容数 ++ {a 空域 |-> n};
return s 許容数

 空域 in set dom s 許容数 and
空機数(a 空域, s 位置) <= n
s 許容数 = s 許容数~ ++ {a 空域 |-> n} ;
                public
やめる : 空域 ==> ()
やめる(a 空域) ==
容数 := {a 空域 } <-: s 許容数
 空域 in set (dom s 許容数 \ dom s 管制)
s 許容数 = {a 空域 } <-: s 許容数~ ;
                public
する : 空域 * 管制官 ==> 管制官
する(a 空域, a 管制官) ==
s 管制 := s 管制 ++ {a 空域 |-> a 管制官 };
return a 管制官
```

```
et p1 = a 空域 in set (dom s 許容数 \ dom s 管制) in
 and
et p2 = rng s 管制 <> s 当直集合 in
2
s 管制 = s 管制~ ++ {a 空域 |-> a 管制官 } ;
                public
化する : 空域 ==> ()
化する (a 空域) ==
制 := {a 空域 } <-:  s 管制
 空域 in set (dom s 管制 \ rng s 位置)
s 管制 = {a 空域 } <-:  s 管制~ ;
                public
当て (a 空域 : 空域, a 管制官 : 管制官) a 管制官 : 管制官 ==
s 管制 := s 管制 ++ {a 空域 |-> a 管制官 };
return a 管制官

 空域 in set dom s 管制 and
ng s 管制 <> s 当直集合
s 管制 = s 管制~ ++ {a 空域 |-> a 管制官 } and
a 管制官 <> s 管制~ (a 空域) ;
                public
る : 管制官 ==> ()
る (a 管制官) ==
直集合 := s 当直集合 union {a 管制官 }
 管制官 not in set s 当直集合
s 当直集合 = s 当直集合~ union {a 管制官 } ;
                public
る : 管制官 ==> ()
る (a 管制官) ==
直集合 := s 当直集合 \ {a 管制官 }
 管制官 in set s 当直集合 \ rng s 管制
s 当直集合 = s 当直集合~ \ {a 管制官 }
                functions
                public
数 : 空域 * 位置 -> nat
数 (a 空域, a 位置) ==
l dom (a 位置 :> {a 空域 })
                end
                航空管制システム
```

Test Suite : vdm.tc

Class : 航空管制システム

Name	#Calls	Coverage
航空管制システム ‘出勤する	**undefined**	**undefined**
航空管制システム ‘委託する	**undefined**	**undefined**
航空管制システム ‘航空機数	**undefined**	**undefined**
航空管制システム ‘退勤する	**undefined**	**undefined**
航空管制システム ‘再割り当て	**undefined**	**undefined**
航空管制システム ‘活性化する	**undefined**	**undefined**
航空管制システム ‘委託をやめる	**undefined**	**undefined**
航空管制システム ‘非活性化する	**undefined**	**undefined**
航空管制システム ‘空域を切り替える	**undefined**	**undefined**
航空管制システム ‘航空管制システム	**undefined**	**undefined**
航空管制システム ‘フライトを削除する	**undefined**	**undefined**
航空管制システム ‘フライトを追加する	**undefined**	**undefined**
航空管制システム ‘許容数を設定し直す	**undefined**	**undefined**
Total Coverage		**0%**

3 TestApp

```
class
TestApp
```

3.0.1 Operation : run

Main operation of the regression test.

```
operations
public static
() ==> ()
 ==
let tests : set of Test = {new TestCase0001 ()},
    ts : TestSuite = new TestSuite (tests),
    result : TestResult = new TestResult () in
(   result.addListener(new PrintTestListener ()) ;
    ts.run(result) ;
    IO`print(result.toString ())
)

end
TestApp
```

Test Suite : vdm.tc

Class : TestApp

Name	#Calls	Coverage
TestApp`run	**undefined**	**undefined**
Total Coverage		**0%**

4 TestCaseCom

私はテストケースの共通クラスである。

```
.....
class
TestCaseCom is subclass of TestCase
operations
public
: seq of char ==> ()
(a 文字列) ==
- = new IO().echo(a 文字列) in
)
values
public
管制システム 0 = new 航空管制システム ()
end
TestCaseCom
class
TestCaseOOO1 is subclass of TestCaseCom
.....
```

5 TestCase0001

I am a regression testcase for nromal case.

```
.....
operations
protected
```

```
_ : () ==> ()
_() ==
assertTrue
    ("\ttest01 嘉手納空域に許容数 2 で委託できない。\n",
     let m = v 航空管制システム 0.委託する (mk_token ("嘉手納"),2) in
     m (mk_token ("嘉手納")) = 2);
assertTrue
    ("\ttest01 嘉手納空域を許容数 5 で設定し直すことができない。\n",
     let m = v 航空管制システム 0.許容数を設定し直す (mk_token ("嘉手納"),5) in
     m (mk_token ("嘉手納")) = 5);
assertTrue
    ("\ttest01 横田空域に許容数 3 で委託できない。\n",
     let m = v 航空管制システム 0.委託する (mk_token ("横田"),3) in
     m (mk_token ("横田")) = 3);
assertTrue
    ("\ttest01 横田空域を許容数 4 で設定し直すことができない。\n",
     let m = v 航空管制システム 0.許容数を設定し直す (mk_token ("横田"),4) in
     m (mk_token ("横田")) = 4);
assertTrue
    ("\ttest01 航空機数が想定した 2 にならない。\n",
     v 航空管制システム 0.numOfAircraft (mk_token ("横田"),{mk_token ("F35_1") |-> mk_token ("横田"),mk_token ("F18_1") |->
              横田")}) = 2);
v 航空管制システム 0.出勤する (mk_token ("小田"));
assertTrue
    ("\ttest01 管制官小田が出勤できない。\n",
     mk_token ("小田") in set v 航空管制システム 0.s 当直集合);
assertTrue
    ("\ttest01 横田空域で、小田管制官を活性化できない。\n",
     v 航空管制システム 0.活性化する (mk_token ("横田"),mk_token ("小田")) = mk_token ("小田"));
v 航空管制システム 0.フライトを追加する (mk_token ("F35_1"),mk_token ("横田"));
assertTrue
    ("\ttest01 横田空域にフライト F35_1 を追加できない。\n",
     v 航空管制システム 0.s 位置 (mk_token ("F35_1")) = mk_token ("横田"));
v 航空管制システム 0.出勤する (mk_token ("佐原"));
assertTrue
    ("\ttest01 管制官佐原が出勤できない。\n",
     mk_token ("佐原") in set v 航空管制システム 0.s 当直集合);
assertTrue
    ("\ttest01 嘉手納空域で、佐原管制官を活性化できない。\n",
     v 航空管制システム 0.活性化する (mk_token ("嘉手納"),mk_token ("佐原")) = mk_token ("佐原"));
v 航空管制システム 0.空域を切り替える (mk_token ("F35_1"),mk_token ("嘉手納"));
v 航空管制システム 0.フライトを削除する (mk_token ("F35_1"));
assertTrue
    ("\ttest01 嘉手納空域でフライト F35_1 を追加できない。\n",
     mk_token ("F35_1") not in set rng v 航空管制システム 0.s 位置);
v 航空管制システム 0.非活性化する (mk_token ("嘉手納"));
```

```
end
TestCase0001
class
TestCaseE0001 is subclass of TestCaseCom
```

6 TestCaseE0001

I am a regression testcase for nromal case.

```
operations
protected
01 : () ==> ()
01() ==
assertTrue
    ("\ttestE01 航空機数が想定した 1 にならない。\n",
     v 航空管制システム 0.航空機数 (mk_token ("横田"), {mk_token ("F35_1") |-> mk_token ("嘉手納"), mk_token ("F18_1") |-> mk_to
              横田")}) = 1);
assertTrue
    ("\ttestE01 厚木空域を許容数 2 で委託できない。\n",
     let m = v 航空管制システム 0.委託する (mk_token ("厚木"), 2) in
     m (mk_token ("厚木")) = 2);
assertTrue
    ("\ttestE01 厚木空域で管制官小田を活性化できない。\n",
     let m = v 航空管制システム 0.活性化する (mk_token ("厚木"), mk_token ("小田")) in
     m = mk_token ("小田"));
assertTrue
    ("\ttestE01 岩国空域で管制官小田を活性化できない。\n",
     let m = v 航空管制システム 0.活性化する (mk_token ("岩国"), mk_token ("小田")) in
     m = mk_token ("小田"))

end
TestCaseE0001
```

Test Suite : vdm.tc

Class : TestCase0001

Name	#Calls	Coverage
TestCase0001‘test01	**undefined**	**undefined**
Total Coverage		**0%**

7　参考文献、索引

VDM++[2] は、1970 年代中頃に IBM ウィーン研究所で開発された VDM-SL[1] を拡張し、さらにオブジェクト指向拡張した

VDM++ の教科書としては [3] がある。

VDM++ を開発現場で実践的に使う場合の解説が [4] にある。

参考文献

[1] SCSK Corporation. VDM-SL 言語マニュアル. SCSK Corporation, 第 1.2 版, 2012. Revised for VDMTools V9.0.2.

[2] SCSK Corporation. VDM++ 言語マニュアル. SCSK Corporation, 第 1.2 版, 2012. Revised for VDMTools V9.0.2.

[3] ジョン・フィッツジェラルド, ピーター・ゴルム・ラーセン, ポール・マッカージー, ニコ・プラット, マーセル・バーホフ (著), , 酒匂寛 (訳). VDM++ によるオブジェクト指向システムの高品質設計と検証. IT architects' archive. 翔泳社, 2010.

[4] 佐原伸. 形式手法の技術講座—ソフトウェアトラブルを予防する. ソフトリサーチセンター, 2008.

索引