特急券予約システムシステムの問題点を推測した仕様

佐原　伸

法政大学情報科学研究科

目次

1　発端

鉄道会社 A の特急券予約システムで特急券を 2 枚予約し、以下のような「事件」が起こったのが本モデルを書いた理由である。

以下のやりとりは、電話でのやりとりをかなり (主として用語を中心として）整理したものである。実際には、例えば、鉄道会社 A サポートセンターの担当者は「カード」という言い方をするのだが、それがクレジットカードの場合と、予約会員証の場合と、IC カードあるいは予約カードの場合があったので、個々の用語の正確な名前と意味を調べるだけでかなりの日数と時間を要した問答であった。

IC カードと予約カードは、結局、この問題とは直接は関係なかったので、この後、本稿に登場することはない。

- おサイフケータイ（鉄道会社 B）で特急券予約システム（鉄道会社 A）を使っていた
- ロンロン View カードが廃止になりアトレクラブ View カードに変更して下さいとの連絡があった
- 予約できなかったので 鉄道会社 A サポートセンターに電話
  客　「特急券を 2 枚予約しようとしたが、予約できなかったんですが？」
  JR　「カードを変更したら、特急券予約システムを新規に契約して下さい。」
  客　「新規にすると会費がかかるのでは？」
  JR　「はい。」
  客　「カード会社の都合で変更するのに、それはおかしいでしょう？」
  JR　「では、無料にします。」
- 特急券に引換えできなかったので 鉄道会社 A サポートセンターに電話
  客　「新しい予約はできたんですが、クレジットカード変更前に予約した特急券に引換えようとしたらできないのですが？」
  JR　「予約会員証で引換できるようになったので、それで引換えて下さい。暗証番号はクレジットカードのものを使って下さい。」
- T 駅での問答
  客　「会員証で特急券に引換えできないのですが？」
  JR　「会員証で引換えできないですねー。おかしいな。クレジットカードでやってみましょう。駄目ですねー。」
  客　「古いクレジットカードでは駄目ですか？」
  JR　「あ、できましたね。はい、切符です。」

2　モデル化の範囲

この問題で登場する用語には以下がある。

- 鉄道会社 A、鉄道会社 B
- おサイフケータイ
- 特急券予約システム

- 予約会員証、IC カード、予約カード
- クレジットカード
  - ロンロン View カード、アトレクラブ View カード
- 特急券

「鉄道会社 B」「IC カード」「予約カード」は、モデルを単純化するため対象としなかった。
「鉄道会社 A」の「特急券予約システム」システムのうち、以下の機能だけを VDM++[?] で要求仕様としてモデル化し、問題点を明確化することにした。

- 予約する
- 特急券を得る
- クレジットカードを切り替える

従って、モデル化する範囲に登場する public な用語は以下だけである。

- おサイフケータイ
- 特急券予約システムシステム
- 特急券予約システム
- 予約会員証
- クレジットカード
- 特急券

2.0.1 特急券予約システムクラス

最初に、業務論理階層の仕様である特急券予約システムクラスの仕様を記述するが、まず、クラスが果たすべき責任 (responsibility[*1]) を記述することで、属性 (インスタンス変数) や、機能 (操作と関数) を決定する判断基準とする。このような基準で記述したクラスは、再利用性や保守性を満たす可能性が大きい。

2.0.1.1 責任

特急券予約システムの主要機能を、要求辞書階層で定義されたクラスや型あるいは操作や関数を使用して、構造化日本語仕様と言える形の VDM++ で記述する。

2.0.1.2 クラス定義と構成子定義

本クラスのスーパークラス「特急券予約 DomainData」は、特急券予約に関するドメインデータ (今の場合、インスタンス変数「s 予約表」) を定義している。

構成子「特急券予約システム」は、予約表をパラメータとして受け取り、特急券予約システムクラスのインスタンスを生成する。

```
.....................................................................................
class
特急券予約システム is subclass of 特急券予約 DomainData
operations
public
  特急券予約システム : 予約表 ==> 特急券予約システム
  特急券予約システム (a 予約表) ==
    (   s 予約表 := a 予約表
    );
.....................................................................................
```

2.0.1.3 操作：予約する

「特急券予約 DomainData」のスーパークラスであり、本クラスのスーパークラスでもある、要求辞書階層の「特急券予約 Domain」クラスの関数「予約表を更新する」と「予約表に追加する」を呼び出して、予約を行う。

```
.....................................................................................
public
```

[*1] 責務と訳すこともある。

```
予約する : 契約 * ID * クレジットカード * 特急券予約 ‘予約内容 ==> 特急券予約
予約する (a 契約, anID, a クレジットカード, a 予約内容) ==
  (   def w 特急券予約 = new 特急券予約 (anID, a クレジットカード, a 予約内容) in
      (   if 予約がある契約である (a 契約, s 予約表)
          then s 予約表 := 予約表を更新する (s 予約表, a 契約, w 特急券予約)
          else s 予約表 := 予約表に追加する (s 予約表, a 契約, w 特急券予約);
          return w 特急券予約
      )
  )
 post if 予約がある契約である (a 契約, s 予約表~)
      then 予約表が更新されている (s 予約表~, a 契約, RESULT, s 予約表)
      else 予約表に追加されている (s 予約表~, a 契約, RESULT, s 予約表) ;
```

2.0.1.4 操作：特急券を得る

クレジットカードで特急券を得る。「特急券を得る」ことは、予約内容を返すということで抽象化した。

```
public
 特急券を得る : ID * クレジットカード ==> 特急券予約 ‘予約内容
 特急券を得る (anID, a クレジットカード) ==
  (   def w 予約 = 予約を得る (s 予約表, anID, a クレジットカード) in
      return w 予約.予約内容を得る ()
  )
 pre  let w 予約 = 予約を得る (s 予約表, anID, a クレジットカード) in
     a クレジットカード = w 予約.クレジットカードを得る () ;
```

2.0.1.5 操作：特急券を得る

予約会員証で特急券を得ることもできるので、前に記述したクレジットカードを使う「特急券を得る」操作のオーバーロード操作[*2]を定義する。

```
public
```

[*2] オーバーロード (overload) とは、同一の名前の関数や操作を複数定義し、使用時に文脈に応じて選択することで複数の動作を行わせる仕組みである。

```
  特急券を得る : ID * 予約会員証 ==> 特急券予約 ‘予約内容
  特急券を得る (anID, a 会員証) ==
    (   def w 予約 = 予約を得る (s 予約表, anID, a 会員証) in
        return w 予約.予約内容を得る ()
    )
  pre  let w 予約 = 予約を得る (s 予約表, anID, a 会員証) in
       a 会員証.クレジットカードを得る () = w 予約.クレジットカードを得る () ;
```

2.0.1.6 操作：クレジットカードを切り替える

実システムでは、本操作を一般ユーザーが使うことはできず、鉄道会社 A サポートセンターの担当者だけが使えるが、本モデルでは、誰がある操作を行えるかのチェックは捨象して対象外としている。

```
public
  クレジットカードを切り替える : 契約 * クレジットカード ==> ()
  クレジットカードを切り替える (a 契約, a 新しいクレジットカード) == a 契約.会員証を得る
                          ( )  .クレジットカードを設定する (a 新しいクレジットカード)
end
```

特急券予約システム

以下の表は、VDMTools が生成した VDM++ 仕様実行時の VDM++ の命令レベルで何 % 実行したかを表すカバレージ情報である。

Test Suite : vdm.tc

Class : 特急券予約システム

Name	#Calls	Coverage
特急券予約システム ‘予約する	10	√
特急券予約システム ‘特急券予約システム	5	√
特急券予約システム ‘クレジットカードを切り替える	3	√
特急券予約システム ‘特急券を得る	2	√
特急券予約システム ‘特急券を得る	16	√
Total Coverage		**100%**

命令レベルのコードカバレージとは、各式や文を実行したか否かを表し、ループや分岐によって生じるすべての組合せが実行されたことは保証されないが、通常、95% 以上のカバレージが達成されていれば、ほぼ信頼できる品質であることが分かっている。

Name カラムは関数ないし操作の名前、#Calls カラムは呼び出し回数、Coverage カラムは各関数あるいは操作内の命令の何 % を実行したかを表す。Coverage がチェックマークの場合は、100% 実行したことを表す。

2.0.2 共通定義クラス

2.0.2.1 責任

特急券予約モデルで共通に使用する定義を表す。

```
class
共通定義
types
public ID = [token];
public 暗証番号 = [token]
operations
public
print : seq of char ==> ()
print(a 文字列) ==
  let - = new IO().echo(a 文字列) in
  skip
end
共通定義
```

Test Suite : vdm.tc

Class : 共通定義

Name	#Calls	Coverage
共通定義 ‘print	3	√
Total Coverage		**100%**

2.0.3 特急券予約 Domain クラス

2.0.3.1 責任

特急券予約に関わる問題領域 (domain) の用語を定義する、要求辞書階層の仕様である。

2.0.3.2 クラス定義

```
.....................................................................................
class
特急券予約 Domain is subclass of 共通定義
.....................................................................................
```

2.0.3.3 型およびインスタンス変数定義 : 予約表

予約情報を持つ予約表は、契約から (個々の予約を表す) 特急券予約の集合への写像として定義した。

写像はキーとデータが対応した表と考えていただければよい。ただし、同じキーから異なるデータに対応してはいけないという制約がある

```
.....................................................................................
types
public 予約表 = map 契約 to set of 特急券予約
.....................................................................................
```

2.0.3.4 関数 : 予約を得る

予約 ID を指定して、予約を得る。[*3]

```
.....................................................................................
functions
public
予約を得る : 予約表 * ID -> 特急券予約
予約を得る (a 予約表, anID) ==
  (let  w 予約 in set dunion rng a 予約表 be st  w 予約.ID を得る () = anID in
   w 予約)
pre  exists w 予約 in set dunion rng a 予約表 &
        w 予約.ID を得る () = anID ;
.....................................................................................
```

[*3] 以下の VDM++ ソースで赤く表示されることがあるのは、この部分が実行されていないことを示している。

2.0.3.5 関数：予約を得る

クレジットカードを指定して、予約を得る。

```
public
予約を得る : 予約表 * ID * クレジットカード -> 特急券予約
予約を得る (a 予約表, anID, a クレジットカード) ==
  (let  w 予約 in set dunion rng a 予約表 be st
           w 予約.クレジットカードを得る () = a クレジットカード and
          w 予約.ID を得る () = anID in
   w 予約)
pre  exists w 予約 in set dunion rng a 予約表 &
        w 予約.クレジットカードを得る () = a クレジットカード and
        w 予約.ID を得る () = anID ;
```

2.0.3.6 関数：予約を得る

予約会員証を指定して、予約を得る。

```
public
予約を得る : 予約表 * ID * 予約会員証 -> 特急券予約
予約を得る (a 予約表, anID, a 予約会員証) ==
  (let  w 予約 in set dunion rng a 予約表 be st
           w 予約.クレジットカードを得る () = a 予約会員証.クレジットカードを得る () and
          w 予約.ID を得る () = anID in
   w 予約)
pre  exists w 予約 in set dunion rng a 予約表 &
        w 予約.クレジットカードを得る () = a 予約会員証.クレジットカードを得る () and
        w 予約.ID を得る () = anID ;
```

2.0.3.7 関数：予約表を更新する

契約と特急券予約を指定して、予約表を更新する。

```
public
```

```
予約表を更新する : 予約表 * 契約 * 特急券予約 -> 予約表
予約表を更新する (a 予約表, a 契約, a 特急券予約) ==
  a 予約表 ++ {a 契約 |-> 予約集合に追加する (a 予約表 (a 契約), {a 特急券予約 })}
pre  a 契約 in set dom a 予約表
post 予約表が更新されている (a 予約表, a 契約, a 特急券予約, RESULT) ;
public
予約表が更新されている : 予約表 * 契約 * 特急券予約 * 予約表 +> bool
予約表が更新されている (a 予約表, a 契約, a 特急券予約, a 更新後予約表) ==
  a 更新後予約表 = a 予約表 ++ {a 契約 |-> 予約集合に追加する (a 予約表 (a 契約), {a 特急券予約 })};
```

2.0.3.8 関数：予約表に追加する

契約と特急券予約を指定して、予約表に追加する。

```
public
予約表に追加する : 予約表 * 契約 * 特急券予約 -> 予約表
予約表に追加する (a 予約表, a 契約, a 特急券予約) ==
  a 予約表 munion {a 契約 |-> {a 特急券予約 }}
pre  not 予約がある契約である (a 契約, a 予約表) and
     forall w 契約 1 in set dom a 予約表, w 契約 2 in set dom {a 契約 |-> {a 特急券予約 }} &
        w 契約 1 = w 契約 2 => a 予約表 (w 契約 1) = {a 契約 |-> {a 特急券予約 }} (w 契約 2)
post 予約表に追加されている (a 予約表, a 契約, a 特急券予約, RESULT) ;
public
予約表に追加されている : 予約表 * 契約 * 特急券予約 * 予約表 +> bool
予約表に追加されている (a 予約表, a 契約, a 特急券予約, a 更新後予約表) ==
  a 更新後予約表 = a 予約表 munion {a 契約 |-> {a 特急券予約 }};
```

2.0.3.9 関数：予約がある契約である

予約がある契約か判定する。

```
public
予約がある契約である : 契約 * 予約表 +> bool
予約がある契約である (a 契約, a 予約表) ==
  a 契約 in set dom a 予約表;
```

2.0.3.10 関数：予約集合に追加する

既存予約集合と新規予約集合を指定して、予約集合に追加する。

```
public
予約集合に追加する : set of 特急券予約 * set of 特急券予約 +> set of 特急券予約
予約集合に追加する (a 既存予約集合, a 新規予約集合) ==
  a 既存予約集合 union a 新規予約集合
end
特急券予約 Domain
```

Test Suite : vdm.tc

Class : 特急券予約 Domain

Name	#Calls	Coverage
特急券予約 Domain‘予約表に追加する	5	65%
特急券予約 Domain‘予約表を更新する	5	√
特急券予約 Domain‘予約集合に追加する	15	√
特急券予約 Domain‘予約がある契約である	25	√
特急券予約 Domain‘予約表が更新されている	10	√
特急券予約 Domain‘予約表に追加されている	10	√
特急券予約 Domain‘予約を得る	0	0%
特急券予約 Domain‘予約を得る	3	√
特急券予約 Domain‘予約を得る	30	√
Total Coverage		**80%**

2.0.4 特急券予約 DomainData クラス

2.0.4.1 責任

特急券予約に関わる問題領域 (domain) の、状態を持つデータを表す。

2.0.4.2 クラス定義

```
class
特急券予約 DomainData is subclass of 特急券予約 Domain
instance variables
protected s 予約表 : 予約表 := { |-> };
```

2.0.4.3 操作：予約集合を得る

契約を指定して、予約集合を得る。

```
operations
public
予約集合を得る : 契約 ==> set of 特急券予約
予約集合を得る (a 契約) ==
  if dom s 予約表 = {}
  then return {}
  else return s 予約表 (a 契約)
pre  dom s 予約表 <> {} => a 契約 in set dom s 予約表
end
特急券予約 DomainData
```

Test Suite : vdm.tc

Class : 特急券予約 DomainData

Name	#Calls	Coverage
特急券予約 DomainData‘予約集合を得る	5	60%
Total Coverage		**60%**

2.0.5 特急券予約クラス

2.0.5.1 責任

特急券予約 1 件を表す、要求辞書階層の仕様である。

2.0.5.2 クラス定義

```
.....................................................................................
class
特急券予約 is subclass of 共通定義
.....................................................................................
```

2.0.5.3 型定義 : 予約内容

予約内容の型の詳細は、今回の問題に関わりがなく、仕様を詳細化していった時のみ必要なので、こういった場合の定石として token 型[*4]で定義する。

```
.....................................................................................
types
public 予約内容 = token
.....................................................................................
```

2.0.5.4 インスタンス変数定義 : ID, クレジットカード, 予約内容

```
.....................................................................................
instance variables
sID : ID;
s クレジットカード : クレジットカード;
s 予約内容 : 予約内容;

.....................................................................................
```

2.0.5.5 構成子

```
.....................................................................................
operations
public
```

[*4] token 型は VDM++ に存在する、抽象化のために使用する型である。システムの本質と関係ない詳細な記述を回避し、短時間で主要な仕様を記述し検証するために使う。

```
特急券予約 : ID * クレジットカード * 特急券予約 ‘ 予約内容 ==> 特急券予約
特急券予約 (anID, a クレジットカード, a 予約内容) == atomic
  (  sID := anID;
     s クレジットカード := a クレジットカード;
     s 予約内容 := a 予約内容
  );
```

2.0.5.6 アクセッサー

```
public
ID を得る : () ==> ID
ID を得る () ==
  return sID;
public
クレジットカードを得る : () ==> クレジットカード
クレジットカードを得る () ==
  return s クレジットカード;
public
予約内容を得る : () ==> 予約内容
予約内容を得る () ==
  return s 予約内容
end
特急券予約
```

Test Suite : vdm.tc

Class : 特急券予約

Name	#Calls	Coverage
特急券予約 ‘ID を得る	63	√
特急券予約 ‘特急券予約	10	√
特急券予約 ‘予約内容を得る	15	√
特急券予約 ‘クレジットカードを得る	97	√
Total Coverage		**100%**

2.0.6 契約クラス

2.0.6.1 責任

特急券予約の客と鉄道会社 A の契約を表すドメイン・オブジェクトで、要求辞書階層の仕様である。

2.0.6.2 クラス定義

```
.....................................................................................
class
契約 is subclass of 共通定義
.....................................................................................
```

2.0.6.3 インスタンス変数定義 : おサイフケータイ, 予約会員証

```
.....................................................................................
instance variables
s おサイフケータイ : おサイフケータイ;
s 会員証 : 予約会員証;

.....................................................................................
```

2.0.6.4 構成子

実際には、契約が結ばれてから予約会員証が送られてくるまでの時差があるが、簡単化のため無視する。

```
.....................................................................................
operations
public
契約 : おサイフケータイ * 予約会員証 ==> 契約
契約 (a おサイフケータイ,a 会員証) == atomic
  (  s おサイフケータイ := a おサイフケータイ;
     s 会員証 := a 会員証
  );
.....................................................................................
```

2.0.6.5 アクセッサー

```
.....................................................................................
public
```

```
会員証を得る : () ==> 予約会員証
会員証を得る () ==
  return s 会員証
end
契約
```

Test Suite : vdm.tc

Class : 契約

Name	#Calls	Coverage
契約 ‘契約	5	√
契約 ‘会員証を得る	8	√
Total Coverage		**100%**

2.0.7　カードクラス

2.0.7.1　責任

カードの抽象クラスで、要求辞書階層の仕様である。

2.0.7.2　クラス定義

```
.....................................................................................
class
カード is subclass of 共通定義
.....................................................................................
```

2.0.7.3　インスタンス変数定義 : ID, 暗証番号

```
.....................................................................................
instance variables
protected sID : ID := nil;
protected s 暗証番号 : 暗証番号 := nil;

end
カード
.....................................................................................
```

Test Suite :　vdm.tc

Class :　カード

Name	#Calls	Coverage
Total Coverage		undefined

2.0.8 クレジットカードクラス

2.0.8.1 責任

クレジットカードを表すドメイン・オブジェクトで、要求辞書階層の仕様である。

2.0.8.2 クラス定義

```
class
クレジットカード is subclass of カード
```

2.0.8.3 構成子

```
operations
public
クレジットカード : ID * 暗証番号 ==> クレジットカード
クレジットカード(anID, a 暗証番号) == atomic
  (  sID := anID;
     s 暗証番号 := a 暗証番号
  )
end
クレジットカード
```

Test Suite : vdm.tc

Class : クレジットカード

Name	#Calls	Coverage
クレジットカード ‘クレジットカード	9	√
Total Coverage		**100%**

2.0.9　予約会員証

2.0.9.1　責任

予約会員証を表すドメイン・オブジェクトで、要求辞書階層の仕様である。

2.0.9.2　クラス定義

```
.....................................................................................
class
予約会員証 is subclass of カード
.....................................................................................
```

2.0.9.3　インスタンス変数定義 : クレジットカード

```
.....................................................................................
instance variables
s クレジットカード : クレジットカード;

.....................................................................................
```

2.0.9.4　構成子

```
.....................................................................................
operations
public
予約会員証 : ID * 暗証番号 * クレジットカード ==> 予約会員証
予約会員証 (anID, a 暗証番号, a クレジットカード) == atomic
  (  sID := anID;
     s 暗証番号 := a 暗証番号;
     s クレジットカード := a クレジットカード
  );
.....................................................................................
```

2.0.9.5　アクセッサー

```
.....................................................................................
public
```

```
クレジットカードを得る : () ==> クレジットカード
クレジットカードを得る () ==
  return s クレジットカード;
public
クレジットカードを設定する : クレジットカード ==> ()
クレジットカードを設定する (a クレジットカード) ==
  s クレジットカード := a クレジットカード
end
```

予約会員証

Test Suite : vdm.tc

Class : 予約会員証

Name	#Calls	Coverage
予約会員証 ‘予約会員証	5	√
予約会員証 ‘クレジットカードを得る	11	√
予約会員証 ‘クレジットカードを設定する	3	√
Total Coverage		**100%**

2.0.10　おサイフケータイクラス

2.0.10.1　責任

おサイフケータイに搭載されているスマートカードを表すドメイン・オブジェクトで、要求辞書階層の仕様である。

2.0.10.2　クラス定義

```
.....................................................................................
class
おサイフケータイ is subclass of カード
.....................................................................................
```

2.0.10.3　インスタンス変数定義 : クレジットカード

```
.....................................................................................
instance variables
s クレジットカード : クレジットカード;

.....................................................................................
```

2.0.10.4　構成子

本モデルでは、単純化のため、おサイフケータイで改札を通る処理は対象としていないため、暗証番号設定は省略した。

```
.....................................................................................
operations
public
おサイフケータイ : ID * クレジットカード ==> おサイフケータイ
おサイフケータイ(anID,a クレジットカード) == atomic
  (  sID := anID;
     s クレジットカード := a クレジットカード
  );
.....................................................................................
```

2.0.10.5　アクセッサー

```
.....................................................................................
public
```

```
クレジットカードを得る : () ==> クレジットカード
クレジットカードを得る () ==
  return s クレジットカード
end
おサイフケータイ
```

Test Suite : vdm.tc

Class : おサイフケータイ

Name	#Calls	Coverage
おサイフケータイ ‘おサイフケータイ	9	√
おサイフケータイ ‘クレジットカードを得る	9	√
Total Coverage		**100%**

2.0.11 TestApp クラス

2.0.11.1 責任

特急券予約モデルの回帰テスト (**??**クラス参照) を行う、テスト階層の仕様である。

2.0.11.2 クラス定義

```
.....
class
TestApp
.....
```

2.0.11.3 操作 : run

回帰テストケースを TestSuite に追加し、実行し、成功したか判定する。

```
.....
operations
public static
run : () ==> ()
run () ==
  (   dcl ts : TestSuite := new TestSuite ("特急券予約モデルの回帰テスト \n"),
          tr : TestResult := new TestResult ();
      tr.addListener(new PrintTestListener ());
      ts.addTest(new TestCaseT0001 ("TestCaseT0001 通常の予約成功 \n"));
      ts.addTest(new TestCaseT0002 ("TestCaseT0002 クレジットカードを切り替えたときの予約成
功 \n"));
      ts.addTest(new TestCaseT0003 ("TestCaseT0003 クレジットカードを切り替え、予約からクレ
ジットカードで特急券を得るが、予約と異なるクレジットカードでは失敗 \n"));
      ts.addTest(new TestCaseT0004 ("TestCaseT0004 クレジットカードを切り替え、予約から予約
会員証で特急券を得るが失敗 \n"));
      ts.addTest(new TestCaseT0005 ("TestCaseT0005 クレジットカードを切り替え、予約から古い
クレジットカードで特急券を得る事に成功 \n"));
      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
```

Test Suite : vdm.tc

Class : TestApp

Name	#Calls	Coverage
TestApp`run	1	86%
Total Coverage		**86%**

2.0.12 TestCaseT0001 クラス

2.0.12.1 責任

通常の予約をテストする、テスト階層の仕様である。

```
.....................................................................................
class
TestCaseT0001 is subclass of TestCase, 共通定義
operations
public
TestCaseT0001 : seq of char ==> TestCaseT0001
TestCaseT0001(name) ==
  setName(name);
public
test01 : () ==> ()
test01() ==
  (   def w クレジットカード = new クレジットカード(mk_token(<クレジット ID01>),mk_token(<クレジット PW01>));
          w おサイフケータイ = new おサイフケータイ(mk_token(<おサイフケータイ ID01>),w クレジットカード);
          w 会員証 = new 予約会員証(mk_token(<会員証 ID01>),mk_token(<会員証 PW01>),w クレジットカード);
          w 契約 = new 契約(w おサイフケータイ,w 会員証);
```

```
        w システム = new 特急券予約システム ({ |-> }) in
    (   assertTrue("test01 契約に失敗",
            w システム.予約集合を得る (w 契約) = {} and
            w 契約.会員証を得る () = w 会員証) ;
        def w おサイフケータイクレジットカード = w おサイフケータイ.クレジットカードを得る ()
in
        (   def - = w システム.予約する (w 契約, mk_token (<予約 ID01>), w おサイフケータイクレジットカード, mk_token (<最初の予約内容>)) in
            assertTrue("test01 予約に失敗",
                w システム.特急券を得る (mk_token (<予 約 ID01>), w クレジットカード) = mk_token (<最初の予約内容>)) ;
            def - = w システム.予約する (w 契約, mk_token (<予約 ID02>), w おサイフケータイクレジットカード, mk_token (<次の予約内容>)) in
            assertTrue("test01 2 回目の予約に失敗",
                w システム.特急券を得る (mk_token (<予 約 ID02>), w クレジットカード) = mk_token (<次の予約内容>))
        )
    )
 )
end
TestCaseT0001
```

Test Suite : vdm.tc

Class : TestCaseT0001

Name	#Calls	Coverage
TestCaseT0001‘test01	1	√
TestCaseT0001‘TestCaseT0001	1	√
Total Coverage		**100%**

2.0.13 TestCaseT0002 クラス

2.0.13.1 責任

クレジットカードを切り替えたときの予約をテストする、テスト階層の仕様である。。

```
class
TestCaseT0002 is subclass of TestCase, 共通定義
operations
public
```

```
TestCaseT0002 : seq of char ==> TestCaseT0002
TestCaseT0002(name) ==
  setName(name);
public
test01 : () ==> ()
test01() ==
  (   def w クレジットカード = new クレジットカード(mk_token(<クレジット ID01>),mk_token(<クレジット PW01>));
          w おサイフケータイ = new おサイフケータイ(mk_token(<おサイフケータイ ID01>),w クレジットカード);
          w 会員証 = new 予約会員証(mk_token(<会員証 ID01>),mk_token(<会員証 PW01>),w クレジットカード);
          w 契約 = new 契約(w おサイフケータイ,w 会員証);
          w システム = new 特急券予約システム({ |-> }) in
      (   assertTrue("test01 契約に失敗",
            w システム.予約集合を得る(w 契約) = {} and
            w 契約.会員証を得る() = w 会員証);
          def w 古いクレジットカード = w おサイフケータイ.クレジットカードを得る() in
          (   def - = w システム.予約する(w 契約,mk_token(<予約 ID01>),w 古いクレジットカード,mk_token(<最初の予約内容>)) in
              assertTrue("test01 予約に失敗",
                w システム.特急券を得る(mk_token(<予 約 ID01>),w クレジットカード) = mk_token(<最初の予約内容>));
              def w2 クレジットカード = new クレジットカード(mk_token(<クレジットカード ID02>),mk_token(<クレジットカード PW02>));
                  w2 おサイフケータイ = new おサイフケータイ(mk_token(<おサイフケータイ ID01>),w2 クレジットカード);
                  w 変更後のクレジットカード = w2 おサイフケータイ.クレジットカードを得る() in
              (   def - = w システム.予約する(w 契約,mk_token(<予約 ID02>),w 変更後のクレジットカード,mk_token(<次の予約内容>)) in
                  assertTrue("test01 2 回目の予約に失敗",
                    w システム.特急券を得る(mk_token(<予 約 ID02>),w2 クレジットカード) = mk_token(<次の予約内容>))
              )
          )
      )
  )
end
TestCaseT0002
.....
```

Test Suite : vdm.tc
Class : TestCaseT0002

Name	#Calls	Coverage
TestCaseT0002‘test01	1	√
TestCaseT0002‘TestCaseT0002	1	√
Total Coverage		**100%**

2.0.14 TestCaseT0003 クラス

2.0.14.1 責任

クレジットカードを切り替え、予約した特急券を得る場合をテストする、テスト階層の仕様である。古いクレジットカードでは「Error 58: 事前条件の評価結果が false です」という実行時エラーが発生し、変更後のクレジットカードでは、特急券を得ることができる。

```
class
TestCaseT0003 is subclass of TestCase, 共通定義
operations
public
TestCaseT0003 : seq of char ==> TestCaseT0003
TestCaseT0003(name) ==
  setName(name);
public
test01 : () ==> ()
test01() ==
  (   def w クレジットカード = new クレジットカード(mk_token(<クレジット ID01>),mk_token(<クレジット PW01>));
          w おサイフケータイ = new おサイフケータイ(mk_token(<おサイフケータイ ID01>),w クレジットカード);
          w 会員証 = new 予約会員証(mk_token(<会員証 ID01>),mk_token(<会員証 PW01>),w クレジットカード);
          w 契約 = new 契約(w おサイフケータイ,w 会員証);
```

```
        w システム = new 特急券予約システム ({ |-> }) in
    (   assertTrue("test01 契約に失敗",
            w システム.予約集合を得る (w 契約) = {} and
            w 契約.会員証を得る () = w 会員証) ;
        def w 古いクレジットカード = w おサイフケータイ.クレジットカードを得る () in
        (   def - = w システム.予約する (w 契約,mk_token (<予約 ID01>),w 古いクレジットカード
,mk_token (<最初の予約内容>)) in
            assertTrue("test01 予約に失敗",
                w システム.特急券を得る (mk_token (<予約 ID01>),w 古いクレジットカード
) = mk_token (<最初の予約内容>)) ;
            def w2 クレジットカード = new クレジットカード (mk_token (<クレジットカード ID02>),mk_token (<クレジットカード PW02>));
            w2 おサイフケータイ = new おサイフケータイ (mk_token (<おサイフケータイ ID01>),w2 クレジットカード);
            w 変更後のクレジットカード = w2 おサイフケータイ.クレジットカードを得る () in
            (   def - = w システム.予約する (w 契約,mk_token (<予約 ID02>),w 変更後のクレジットカード,mk_token (<次の予約内容>)) in
            assertTrue("test01 2 回目の予約に失敗",
                w システム.特急券を得る (mk_token (<予約 ID02>),w 変更後のクレジットカード) = mk_token (<次の予約内容>)) ;
            def w2 予約内容 = w システム.特急券を得る (mk_token (<予約 ID02>),w 変更後のクレジットカード) in
            assertTrue("test01 変更後のクレジットカードで特急券を得ることに失敗",
                w2 予約内容 = mk_token (<次の予約内容>)) ;
            def w2 予約内容 = w システム.特急券を得る (mk_token (<予約 ID01>),w 会員証) in
            assertTrue("test01 予約会員証で特急券を得ることに失敗",
                w2 予約内容 = mk_token (<最初の予約内容>)) ;
            trap <RuntimeError>
            with print("\ttest01 テスト意図通り、古いクレジットカードで特急券を得ることに失敗 \n") in
            def w3 予約内容 = w システム.特急券を得る (mk_token (<予約 ID02>),w 古いクレジットカード) in
            assertTrue("\ttest01 テスト意図に反し、古いクレジットカードで特急券を得ることに成功 \n",
                w3 予約内容 = mk_token (<次の予約内容>))
        )
    )
)
```

```
end
TestCaseT0003
```

Test Suite : vdm.tc
Class : TestCaseT0003

Name	#Calls	Coverage
TestCaseT0003‘test01	1	94%
TestCaseT0003‘TestCaseT0003	1	√
Total Coverage		**94%**

2.0.15 TestCaseT0004 クラス

2.0.15.1 責任

クレジットカードを切り替え、古いクレジットカードで予約した特急券を得る場合をテストする、テスト階層の仕様である。

古いクレジットカードで、特急券を得ることができる。

予約会員証および変更後のクレジットカードでは「Error 58: 事前条件の評価結果が false です」という実行時エラーが発生するが、ログに「test01 テストの意図通り、予約会員証で特急券を得ることに失敗」と表示され、回帰テスト自体は成功する。

```
class
TestCaseT0004 is subclass of TestCase, 共通定義
operations
public
TestCaseT0004 : seq of char ==> TestCaseT0004
TestCaseT0004(name) ==
  setName(name);
public
test01 : () ==> ()
test01() ==
  (   def w クレジットカード = new クレジットカード(mk_token(<クレジット ID01>),mk_token(<クレジット PW01>));
      w おサイフケータイ = new おサイフケータイ(mk_token(<おサイフケータイ ID01>),w クレジットカード);
      w 会員証 = new 予約会員証(mk_token(<会員証 ID01>),mk_token(<会員証 PW01>),w クレジットカード);
      w 契約 = new 契約(w おサイフケータイ,w 会員証);
```

```
        w システム = new 特急券予約システム ({ |-> }) in
    (   assertTrue("test01 契約に失敗",
            w システム.予約集合を得る (w 契約) = {} and
            w 契約.会員証を得る () = w 会員証) ;
        def w 古いクレジットカード = w おサイフケータイ.クレジットカードを得る () in
        (   def - = w システム.予約する (w 契約,mk_token (<予約 ID01>),w 古いクレジットカード
,mk_token (<最初の予約内容>)) in
            assertTrue("test01 予約に失敗",
                w システム.特急券を得る (mk_token (<予約 ID01>),w 古いクレジットカード
) = mk_token (<最初の予約内容>)) ;
            def w2 クレジットカード = new クレジットカード (mk_token (<クレジットカード ID02>),mk_token (<クレジットカード PW02>));
                w2 おサイフケータイ = new おサイフケータイ (mk_token (<おサイフケータイ ID01>),w2 クレジットカード);
                w 変更後のクレジットカード = w2 おサイフケータイ.クレジットカードを得る () in
            (   def - = w システム.予約する (w 契約,mk_token (<予約 ID02>),w 変更後のクレジットカード,mk_token (<次の予約内容>)) in
                assertTrue("test01 2 回目の予約に失敗",
                    w システム.特急券を得る (mk_token (<予約 ID02>),w 変更後のクレジットカード) = mk_token (<次の予約内容>)) ;
                def w2 予約内容 = w システム.特急券を得る (mk_token (<予約 ID01>),w 古いクレジットカード) in
                assertTrue("test01 古いクレジットカード特急券を得ることに失敗",
                    w2 予約内容 = mk_token (<最初の予約内容>)) ;
                trap <RuntimeError>
                with print("\ttest01 テストの意図通り、予約会員証で特急券を得ることに失敗 \n") in
                (   w システム.クレジットカードを切り替える (w 契約,w 変更後のクレジットカード) ;
                    def w3 予約内容 = w システム.特急券を得る (mk_token (<予約 ID01>),w 会員証) in
                    assertTrue("\ttest01 テスト意図に反し、予約会員証で特急券を得ることに成功するが...\n",
                        w3 予約内容 = mk_token (<最初の予約内容>))
                )
            )
        )
    )
end
```

TestCaseT0004

Test Suite : vdm.tc

Class : TestCaseT0003

Name	#Calls	Coverage
TestCaseT0003‘test01	1	94%
TestCaseT0003‘TestCaseT0003	1	√
Total Coverage		**94%**

2.0.16 TestCaseT0005 クラス

2.0.16.1 責任

クレジットカードを切り替え、古いクレジットカードで予約した特急券を得る場合をテストする、テスト階層の仕様である。

古いクレジットカードで、特急券を得ることができる。

予約会員証および変更後のクレジットカードでは「Error 58: 事前条件の評価結果が false です」という実行時エラーが発生するが、ログに「test01 テストの意図通り、古いクレジットカードで特急券を得ることに成功」と表示され、回帰テスト自体は成功する。

```
class
TestCaseT0005 is subclass of TestCase, 共通定義
operations
public
TestCaseT0005 : seq of char ==> TestCaseT0005
TestCaseT0005(name) ==
  setName(name);
public
test01 : () ==> ()
test01() ==
  (   def w クレジットカード = new クレジットカード(mk_token(<クレジット ID01>),mk_token(<クレジット PW01>));
          w おサイフケータイ = new おサイフケータイ(mk_token(<おサイフケータイ ID01>),w クレジットカード);
          w 会員証 = new 予約会員証(mk_token(<会員証 ID01>),mk_token(<会員証 PW01>),w クレジットカード);
          w 契約 = new 契約(w おサイフケータイ,w 会員証);
```

```
        w システム = new 特急券予約システム ({ |-> }) in
    (   assertTrue("test01 契約に失敗",
            w システム.予約集合を得る (w 契約) = {} and
            w 契約.会員証を得る () = w 会員証) ;
        def w 古いクレジットカード = w おサイフケータイ.クレジットカードを得る () in
        (   def - = w システム.予約する (w 契約,mk_token (<予約 ID01>),w 古いクレジットカード
,mk_token (<最初の予約内容>)) in
            assertTrue("test01 予約に失敗",
                w システム.特急券を得る (mk_token (<予約 ID01>),w 古いクレジットカード
) = mk_token (<最初の予約内容>)) ;
            def w2 クレジットカード = new クレジットカード (mk_token (<クレジットカード ID02>),mk_token (<クレジットカード PW02>));
            w2 おサイフケータイ = new おサイフケータイ (mk_token (<おサイフケータイ ID01>),w2 クレジットカード);
            w 変更後のクレジットカード = w2 おサイフケータイ.クレジットカードを得る () in
        (   def - = w システム.予約する (w 契約,mk_token (<予約 ID02>),w 変更後のクレジットカード,mk_token (<次の予約内容>)) in
            assertTrue("test01 2 回目の予約に失敗",
                w システム.特急券を得る (mk_token (<予約 ID02>),w 変更後のクレジットカード) = mk_token (<次の予約内容>)) ;
            def w2 予約内容 = w システム.特急券を得る (mk_token (<予約 ID01>),w 古いクレジットカード) in
            assertTrue("test01 古いクレジットカード特急券を得ることに失敗",
                w2 予約内容 = mk_token (<最初の予約内容>)) ;
            trap <RuntimeError>
            with print("test01 テストの意図通り、変更後のクレジットカードで特急券を得ることに失敗 \n") in
            (   w システム.クレジットカードを切り替える (w 契約,w 変更後のクレジットカード
) ;
                def w3 予約内容 = w システム.特急券を得る (mk_token (<予約 ID01>),w 変更後のクレジットカード) in
                assertTrue("\ttest01 テスト意図に反し、古いクレジットカードで特急券を得ることに失敗するが...\n",
                    w3 予約内容 = mk_token (<最初の予約内容>))
            ) ;
            trap <RuntimeError>
            with print("test01 テストの意図反し、古いクレジットカードで特急券を得ることに失敗 \n") in
```

```
                    (   w システム.クレジットカードを切り替える (w 契約,w 変更後のクレジットカード
);
                        def w3 予約内容 = w システム.特急券を得る (mk_token (<予約 ID01>),w 古い
クレジットカード) in
                        assertTrue("\ttest01 テスト意図に反し、古いクレジットカードで特急券を
得ることに失敗するが...\n",
                            w3 予約内容 = mk_token (<最初の予約内容>))
                    )
                )
            )
        )
    )
end
TestCaseT0005
```

.....

Test Suite : vdm.tc

Class : TestCaseT0004

Name	#Calls	Coverage
TestCaseT0004‘test01	1	94%
TestCaseT0004‘TestCaseT0004	1	√
Total Coverage		**94%**

2.0.17　まとめ

2.0.17.1　問題は何だったのか？

結局、問題は何だったのかといえば、特急券予約システムのクレジットカードによる決済のモデルが考慮不足だったということになる。

まず、おサイフケータイのクレジットカードを変更すると、特急券予約システムを新規に契約せねばならず、予約会員証も新規に作成しなければならない。

しかし、変更前の予約を行った古いクレジットカードは、その予約の特急券を受け取るときに必要になる。何故なら、クラス図 1 で見るように、個々の予約を表す特急券予約システムからクレジットカードにリンクが設定されているからである。

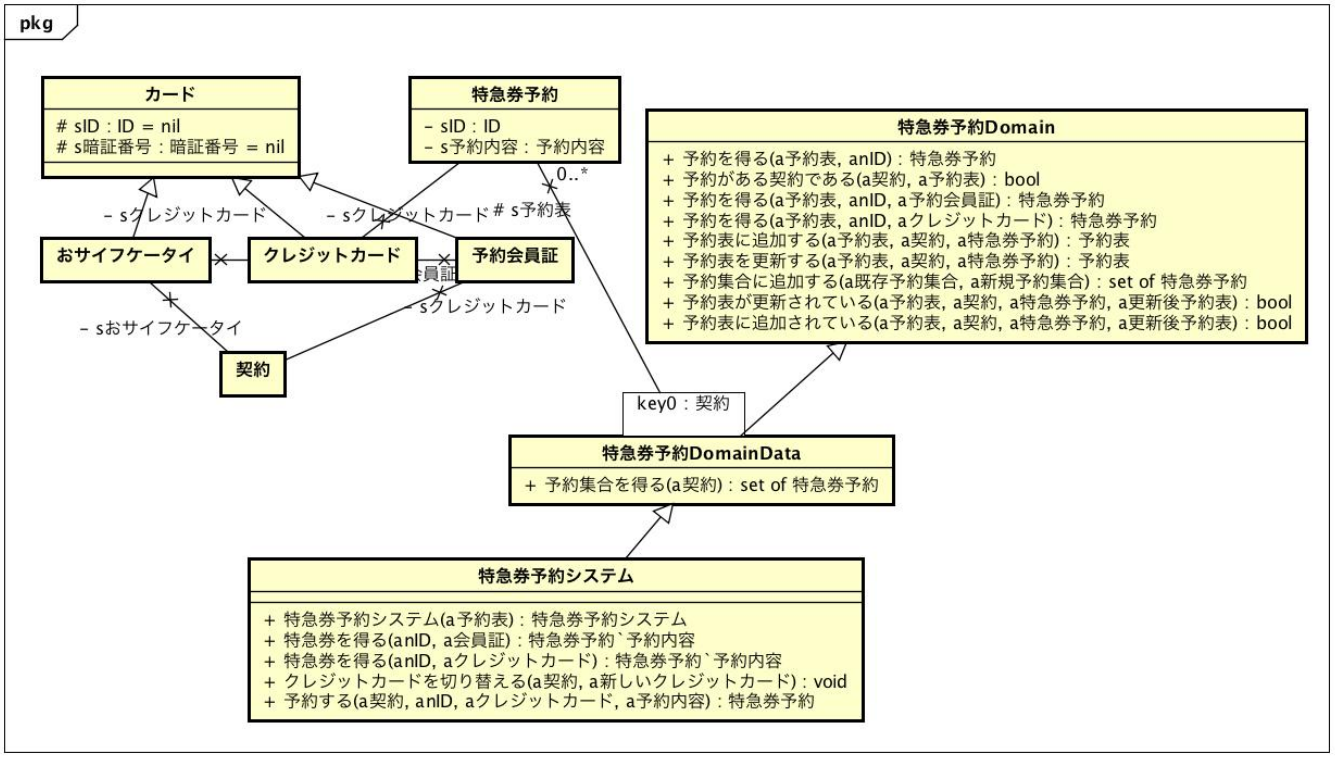

図 1　クラス図 (特急券予約システム)

このような構造でも、変更後にすべての特急券予約システムからクレジットカードへのリンクを新しいクレジットカードに変更していれば問題はないはずだが、恐らく、変更に掛かる効率を理由に、古いクレジットカードとリンクしたままになっているのであろう。設計上の理由で、ユーザーに迷惑を掛ける仕様となっている訳である。

さらに、モデルに問題があった。予約会員証は新規に作るのが普通であるが、クレジットカード会社の都合による変更のため、新規に契約する費用は無料となった。そのため、予約会員証は古いものを使用することに

なった。契約は新規だが、予約会員証は古く、そしてクラス図 1 から分かるように、予約会員証からリンクしているクレジットカードは、変更前の古いものであるという訳である。

これではまずいと思って、鉄道会社 A の係員がリンクされているクレジットカードを新しいものにしたようである。その結果、変更前の特急券予約システムを予約会員証で特急券を得ることができなかった。

2.0.17.2　本来どうすべきだったのか？

本来、特急券予約システムは契約[*5]とリンクが設定されているべきであり、契約がクレジットカードとリンクしているべきである。

予約会員証も契約を介してクレジットカードを参照できるようにすべきである。

このようにしておけば、契約に変更があっても、古い特急券予約システムは新しい契約を介して、新しいクレジットカードにアクセスでき、同じく予約会員証も新しいクレジットカードにアクセスできる。

本来どうすべきだったのかを検証する VDM++ モデルは、1 日で記述および検証ができた。

詳細は、「特急券予約システムシステムの問題点を推測し、修正した仕様」を参照のこと。

2.0.17.3　統計情報

注釈抜きの VDM++ ソース行数は、492 行、モデル作成工数は約 1 日、発表用の資料作成と VDM++ モデルの読みやすさのための整形清書に約 2 日かかった。

なお、上記モデルの作成工数は、筆者が問題を解決するために消費した工数より少ない。

[*5] 口座と言ってもよいが、本モデルでは契約という名前にした

3　参考文献等

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オープンソース[*6]の形式仕様記述言語である。

参考文献

[1] CSK システムズ. VDM-SL 言語マニュアル. CSK システムズ, 第 1.1 版, 2007. Revised for VDMTools V7.1.

[2] CSK システムズ. VDM++ 言語マニュアル. CSK システムズ, 第 1.1 版, 2007. Revised for VDMTools V7.1.

[*6] 使用に際しては、（株）CSK システムズとの契約締結が必要になる。

索引