Vol.2016-SE-191 No.12
2016/3/14

自動販売機システム開発文書への形式仕様記述の適用事例

寺西 祐斗[1,a)]　張 漢明[1,b)]　沢田 篤史[1,c)]

概要：本稿は，モデル規範型形式仕様言語 VDM-SL を用いた機能仕様のための仕様記述法を提示して，実用的な既存の自動販売機システムの仕様書にその記述法を適用した形式仕様記述を通して，ソフトウェアの仕様記述において形式仕様言語を導入する効果と課題を考察する．

キーワード：形式手法，形式仕様言語，VDM，VDM-SL, 自動販売機

A Case Study of Introducing Formal Specification Language in Practical Vending Machine Development Documents

Yuto Teranishi[1,a)]　Han-Myung Chang[1,b)]　Atsushi Sawada[1,c)]

Abstract: In this paper, simple formal specification templates are presented using model based formal specification language VDM-SL for functional specification. We discuss the effects and technical issues of introducing the formal specification language in software specification by applying the templates to a practical existing vending machine example.

Keywords: Formal Methods, Formal Specification Language, VDM, VDM-SL, Vending Machine

1. はじめに

大規模なシステム開発におけるソフトウェアの信頼性を向上させるために，形式的なアプローチは必要不可欠であり，形式手法は有望な技術である [1]．高い信頼性が求められるソフトウェア開発において，形式手法の有効性の事例が報告されている [2]．開発文書に形式仕様言語を導入して設計文書の信頼性を向上させることが，ソフトウェア全体の信頼性向上に寄与する．これは一般的なソフトウェア開発にも当てはまると考えられる．実用的なソフトウェア開発において，現実的には，形式仕様言語は普及していない．

ソフトウェアの仕様記述における形式仕様言語の導入の障壁として，「具体的な簡単な仕様記述の手順がない」と「実用的な形式仕様記述の事例の公開が少ない」ということが考えられる．論文や参考書から形式仕様言語の解説と洗練された形式仕様記述を得ることができる．ソフトウェア開発技術者は，教科書で扱える範囲の例題だけでは，実際の開発現場で形式仕様言語を利用することはできない．形式仕様言語を普及するためには，実用的な仕様書に対する形式仕様記述の導入方法とその結果が必要である．

本研究の目的は，ソフトウェアの仕様記述において形式仕様言語の導入を促進するために，実用的な仕様書を対象とした形式仕様記述の事例を通して，形式仕様言語導入の意義と課題を提示することである．仕様書の「機能仕様」に着目して，概念と表記法が単純な具体的な形式仕様記述法を提示する．形式仕様記述の効果を評価できるように，例題のために作られた仕様ではなく，一般に入手可能な実用的な仕様書を事例の対象とする．

モデル規範型形式仕様言語 VDM-SL を用いた仕様記述法を提示して，ASTER[*1] テスト設計コンテスト'15 における自動販売機の仕様書 [3] にその記述法を適用する．VDM-SL は形式手法 VDM の形式仕様言語の中の基礎と

[1] 南山大学
Nanzan University, Showa, Nagoya 466–8673, Japan
[a)] m15se014@nanzan-u.ac.jp
[b)] chang@nanzan-u.ac.jp
[c)] sawada@nanzan-u.ac.jp

[*1] ソフトウェアテスト技術振興協会

なる言語である．その中でも，関数の概念だけを用いた単純な仕様記述法を提示する．ASTER はソフトウェア開発技術者と研究者が集い，テスト技術に関する技術交流を行っている．テスト設計コンテストでは，実用的な観点から評価を行っているので，ここで公開されている仕様書を実用的な仕様書とみなした．

2. モデル規範型形式仕様言語の概要

機能仕様を記述するための実用的な形式仕様言語として，モデル規範型形式仕様言語の概要を説明する．

モデル規範型形式仕様言語は一般的に集合と第 1 階述語論理に基づいた言語で，対象の状態を集合の概念を用いて表現し，その機能を状態変化として論理式を用いて表現する．対象の状態は型付きの変数の集まりとしてモデル化される．対象を数学の集合や関数や列などを用いてモデル化し，機能を前状態と後状態の関係として定義する．仕様記述言語の特徴として，事前条件や状態不変条件の記述があげられる．

実用的なモデル規範型形式仕様言語として，ツールが整備され参考書が提供されいてる Z[4], B メソッド [5], VDM[6], [7] があげられる．Z は上記の中で数学の記法を素朴に利用する言語である．計算機による実行を想定していないので，簡潔な記述が可能となるが，その記述は数学的な記述能力に依存する．B メソッド は Z を発展させて，リファイメントに基づいた仕様からプログラムまでの検証を含意した形式手法である．VDM は VDM-SL と VDM++ と VDM-RT の言語を持ち，仕様からプログラムまでの多彩な言語を提供している．

3. 形式仕様記述の記述スタイル

本稿では, 機能仕様を記述するための言語として VDM-SL を用いる．VDM-SL を用いる理由は，

- 日本語の識別子が利用可能
- 実績のあるフリーのツールの存在
- 日本語を含めたマニュアルと参考文献の存在

があげられる．既存の自然言語の仕様書から，VDM-SL を用いて機能と振る舞いを記述するスタイルを提示する．

3.1 機能の記述スタイル

対象の機能の記述を

- ものと
- 述語と
- 操作

に分類して VDM-SL による記述のスタイルを示す．記述者は仕様書から「もの」と「述語」と「操作」の概念を見つけて，以下で提示されるテンプレートを用いて, その概念の名前と意味を記述する．

3.2 ものの定義

対象を記述するために必要な「もの」の構造を

- モジュール（module）と
- 型（type）

を用いて記述する．

3.2.1 モジュールによる定義

モジュールの記述には import と export の機能があるので，モジュールの内部構造が変わっても上位のモジュールに影響されないように，ものを抽象化する場合に用いる．

モジュール定義のひな形を以下に示す．

```
module モジュール名
imports from モジュール名 1 ...,
        ...
        from モジュール名 n ...
exports ...
definitions

  定義

end モジュール名
```

モジュールで利用するモジュールを `imports` に記述し，モジュールの外に公開する型や関数を `exports` に記述する．

3.2.2 型による定義

ものには全て型付けを行い，ものの構造は「複合型」を用いて記述する．

型定義のひな形を以下に示す．

```
types
  型名 = 型;

  複合型名::
    メンバー名 1: 型 1
    ...
    メンバー名 n: 型 n;
```

VDM-SL の基本型に用意されていない型は，利用者が値を列挙して新しい型を定義する．

列挙による定義のひな形を以下に示す．

```
types
  型名 = <値 1> | <値 2> | ... | <値 n>
```

仕様書で型が明示されない値があれば，安易に自然数や文字列を用いずに，列挙による型を用いて適切な概念に名前をつける．

3.3 述語と計算式の定義

述語は真偽値を返す関数として定義する．

述語定義のひな形を以下に示す．

```
functions
  述語名:型 1 * 型 2 * ... * 型 n -> bool
  述語名 (arg1, arg2, ... , argn) ==
    述語を表す条件
```

ものや，ものとものの間の性質や条件に名前をつけて述語を定義する．逆に真偽値を返す関数は述語とみなす．

計算式のひな形を以下に示す．

```
functions
  計算式名:型1 * 型2 * ... * 型n ->  型x
  計算式名(arg1, arg2, ... , argn) ==
    計算式の値
```

計算式にはその式が表す目的があるので，それに名前をつけて関数として定義する．

述語と計算式に適切な名前をつけることにより，仕様書の可読性が向上する．

3.4 操作の定義

ものの操作を関数を用いて定義する．

操作定義のひな形を以下に示す．

```
functions
  操作名:型1 * 型2 * ... * 型n -> 型n
  操作名(arg1, arg2, ... , 操作前状態) ==
    操作後の状態
```

関数を用いて操作を記述するためには，引数で操作前の状態をもらい，操作後の状態を返すことで定義する．

4. 事例：自動販売機システム

ソフトウェアテスト技術振興協会（ASTER）が開催したテスト設計コンテスト'15 [*2] で用いられた自動販売機の仕様書を対象として，前節の記述スタイルを適用した結果を示す．

4.1 ASTER 仕様書の概要

自動販売機の仕様書は，

- ASTER 自動販売機ハードウェア構成および販売者用機能仕様 [3]

を対象とする．文献 [3] には，自動販売機の機器の観点から，全体の構造から具体的な数値に至るまで，その機能が詳細に記述されている．

4.2 VDM による機能記述

自動販売機の構造を機能の観点からモジュールに分割して，自動販売機の機能をモジュールで定義されている機能を用いて定義した．

4.2.1 ハードウェア構成のモジュール化

自動販売機の構成を図 1 に示すように機能の観点から 7 つのモジュールに分割した．以降では，自動販売機の主要な構成機器である

- 商品選択機器
- 商品処理機器
- 金銭処理機器

のモジュールの記述例を示し，自動販売機の機能を上記のモジュールを用いて記述する．

4.2.2 商品選択機器

商品選択機器は販売ボタンで構成されており，「商品選択機器型」を「販売ボタン型」を用いて定義する．

```
types
  商品選択機器型::
    販売ボタン map: 販売ボタン map 型;

  ID 型 = token;
  販売ボタン map 型 = map ID 型 to 販売ボタン型;
```

販売ボタンは複数の販売ボタンで構成されているので，写像（map）を用いて表現するのが適切である．販売ボタンを識別するために「ID 型」を定義している．ID 型はトークン型（token）として定義している．トークン型は具体的な値を規定していない型である．トークン型は後の工程で具体化されるもので，仕様記述の段階では詳細と捉え議論すべきでない場合に用いる．

販売ボタンは販売可能ランプと準備中ランプと売切れランプから構成されているので，「販売可能ランプ型」と「準備中ランプ型」と「売切れランプ型」を用いて定義する．

```
types
  販売ボタン型::
    販売可能ランプ: 販売可能ランプ型
    準備中ランプ: 準備中ランプ型
    売切れランプ: 売切れランプ型;

  販売可能ランプ型::
    ランプ: ランプ型;
  準備中ランプ型::
    ランプ: ランプ型
  inv l == l.ランプ.文字色 = <緑>;
  売切れランプ型::
    ランプ: ランプ型
  inv l == l.ランプ.文字色 = <赤>;
```

仕様書にはランプの文字色の記述があったので，文字色を型の不変条件（inv）として記述している．販売可能ランプには文字色の指定がないので，仕様書に文字色に関する記述がなかったことがわかる．型の不変条件を用いて，ものの局所的な性質を統一的に記述することができる．これは，仕様書の読みやすさや保守の観点から有用である．

「ランプ型」の定義を以下に示す．

```
types
 ランプ型::
   状態: オンオフ型
   文字色: 色型;

 オンオフ型 = <オン> | <オフ>;
 色型 = <緑> | <赤>;
```

[*2] http://aster.or.jp/business/contest/contest2015.html

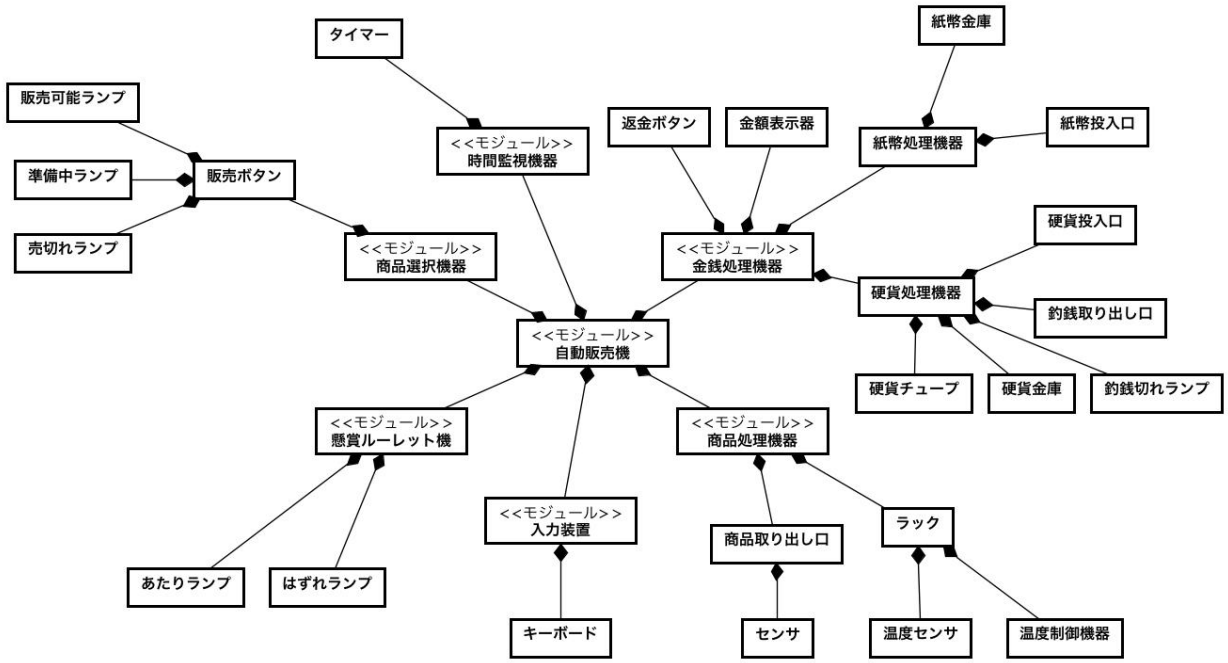

図 1 自動販売機のハードウェア構成
Fig. 1 The hardware configuration of the vending machine

仕様書にはランプに関して，ランプが点灯する記述があったので，メンバーとして，上記の文字色に加えて状態を加えている．ランプの状態は「点灯」「消灯」「ついている」「消えている」などいろいろな表現が想定されるので，ここでは抽象的にオンとオフを用いて表現した．

文字色に関する記述が詳細な記述と判断すれば，ランプ型のメンバーから文字色を削除し，文字色に関する記述を削除すれば良い．上記のような削除に関する修正の漏れがないかは，VDM-SL の型検査を用いて調べることができる．このような型検査器による型の安全性検査は，仕様書の品質として重要である．

4.2.3 商品処理機器

商品処理機器はラックと商品取り出し口で構成されており，「ラック型」と「商品取り出し口型」を用いて定義する．

```
types
  商品処理機器型::
    ラック map: ラック map 型
    商品取り出し口: 商品取り出し口型;

  ラック ID 型 = token;
  ラック map 型 = map ラック ID 型 to ラック型;
```

ラックは前節の販売ボタンと同じ構造である．商品取り出し口型の定義を以下に示す．

```
types
  商品取り出し口型::
    センサ: センサ型;
  検知未検知型 = <検知> | <未検知>;
```

商品取り出し口では，商品があるかないかを検知する機能がある．

ラックの定義の前に，商品処理機器の操作の例として「商品排出」の定義を示す．商品排出はラックを指定して商品処理機器の状態を変化させるので，関数の引数にはラック ID と商品処理機器の操作の前状態を取る．

```
functions
  商品排出:ラック ID 型 * 商品処理機器型 -> 商品処理機器型
  商品排出 (id, 機器) == mu(機器,
   ラック map |->
    機器. ラック map ++ {id |->
     在庫数を減らす (機器. ラック map(id))});
```

上記の定義では，mu 演算子を用いてラック map の id に対応する在庫数を更新している．在庫数の更新は，以下のラックの定義の操作「在庫数を減らす」で定義される．

```
functions
 在庫数を減らす:ラック型 -> ラック型
 在庫数を減らす (ラック) ==
   mu (ラック, 在庫数 |-> ラック. 在庫数 - 1)
 pre 在庫あり (ラック);

 在庫あり:ラック型 -> bool
 在庫あり (ラック) == ラック. 在庫数 > 0;
```

このように形式仕様言語を用いて，対象システムで用いる操作の名前（用語）と意味を関数定義で与えることにより，記述の統一性と可読の容易性を向上させている．

「ラック型」の定義ではハードウェアの「温度センサー型」と「温度制御機器型」に加えて，

- ラックには温商品と冷商品のモードがあり，
- それぞれ適温がある

ことを定義する．

```
types
 ラック型::
   モード: ラックモード型
   収納数上限: nat
   在庫数: nat
   温度センサ: 温度型
   温商品温度範囲: 上限下限型
   冷商品温度範囲: 上限下限型
   温度制御機器: 温度制御機器型
 inv r == r.在庫数 <= r.収納数上限;

 ラックモード型 = <温商品> | <冷商品>;
 温度型 = int;
 上限下限型::
   上限: nat
   下限: nat;
```

モードは「ラックモード型」で表現され，適温は以下の述語で定義する．

```
functions
 適温:ラック型 -> bool
 適温(ラック) ==
   適温温商品(ラック) and 適温冷商品(ラック);

 適温温商品:ラック型 -> bool
 適温温商品(ラック) ==
   ラック.モード = <温商品> =>
   上限下限内(ラック.温商品温度範囲，ラック.温度センサ);

 適温冷商品:ラック型 -> bool
 適温冷商品(ラック) ==
   ラック.モード = <冷商品> =>
   上限下限内(ラック.冷商品温度範囲，ラック.温度センサ);
```

仕様書には，適温の具体的な数字が書かれていたが，具体的な数値よりも，

- 適温という概念があり，
- 適温は上限値と下限値で定義される

ことを表現することが重要である．具体的な数値は型の不変条件で表現することができる．概念を明確にすると，「適温」「上限」「下限」という用語を用いて，統一性の高い記述が可能になる．

4.2.4 金銭処理機器

金銭処理機器は図 1 より「返金ボタン型」と「金額表示器型」と「紙幣処理機器型」と「硬貨処理機器型」を用いて定義する．

```
types
  金銭処理機器型::
    返金ボタン: 返金ボタン型
    金額表示器: 金額表示器型
    紙幣処理機器: 紙幣処理機器型
    硬貨処理機器: 硬貨処理機器型
```

金銭処理機器の操作として，「紙幣投入」と「硬貨投入」と「返金」と「格納」の定義を以下に示す．

```
functions
  紙幣投入:金銭処理機器型 -> 金銭処理機器型
  紙幣投入(金銭処理機器) ==
    mu(金銭処理機器,
       紙幣処理機器 |-> 紙幣処理機器紙幣投入(金銭処理機器.紙幣処理機器));

  硬貨投入:金額型 * 金銭処理機器型 -> 金銭処理機器型
  硬貨投入(金額，金銭処理機器) ==
    mu(金銭処理機器,
      硬貨処理機器 |-> 硬貨処理機器硬貨投入(金額，金銭処理機器.硬貨処理機器));

  返金:金銭処理機器型 -> 金銭処理機器型
  返金(金銭処理機器) ==
    mu(金銭処理機器,
      紙幣処理機器 |-> 紙幣処理機器返金(金銭処理機器.紙幣処理機器),
      硬貨処理機器 |-> 硬貨処理機器返金(金銭処理機器.硬貨処理機器));

  格納:金銭処理機器型 -> 金銭処理機器型
  格納(金銭処理機器) ==
    mu(金銭処理機器,
      紙幣処理機器 |-> 紙幣処理機器格納(金銭処理機器.紙幣処理機器),
      硬貨処理機器 |-> 硬貨処理機器格納(金銭処理機器.硬貨処理機器));
```

上記の操作の詳細は，金銭処理機器を構成する各機器の操作を用いて定義されている．以下に，上記で用いらている操作のうち，ここでは「紙幣処理機器格納」と「硬貨処理機器格納」を取り上げる．紙幣処理機器格納の定義を以下に示す．

```
types
  紙幣処理機器型::
    紙幣投入口: 紙幣投入口型
    紙幣金庫: 紙幣金庫型;

functions
  紙幣処理機器格納:紙幣処理機器型 -> 紙幣処理機器型
  紙幣処理機器格納(mk_紙幣処理機器型(紙幣投入口，紙
```

```
    幣金庫)) ==
        if 紙幣あり(紙幣投入口)
        then
          let
            x = mu(紙幣投入口, 紙幣 |-> <なし>),
            y = mu(紙幣金庫, 格納枚数 |-> 紙幣金庫.格納枚数 + 1)
          in
            mk_紙幣処理機器型(x,y)
        else mk_紙幣処理機器型(紙幣投入口, 紙幣金庫);
```

上記の定義では，紙幣投入口に紙幣がない場合は状態は変化せず，紙幣がある場合は，

- 紙幣の状態を「なし」にして
- 紙幣を金庫に格納する

ことを表現している．

硬貨処理機器格納は，紙幣の処理と同様に硬貨を金庫に格納するのであるが，硬貨の場合は

- 「釣銭」用に硬貨チューブに格納して，
- 残りを金庫に格納する

操作となる．この定義を以下に示す．

```
types
  硬貨処理機器型::
    硬貨投入口: 硬貨投入口型
    硬貨チューブ: 硬貨チューブ型
    硬貨金庫: 硬貨金庫型
    釣銭取り出し口: 釣銭取り出し口型
    釣銭切れランプ: 釣銭切れランプ型

functions
  硬貨処理機器格納:硬貨処理機器型 -> 硬貨処理機器型
  硬貨処理機器格納(硬貨処理機器) ==
    let
      硬貨チューブへ格納する硬貨 =
        硬貨チューブに格納可能な枚数(硬貨処理機器.硬貨投入口.投入枚数, 硬貨処理機器.硬貨チューブ),
      硬貨金庫へ格納する硬貨 =
        硬貨map差(硬貨処理機器.硬貨投入口.投入枚数, 硬貨チューブへ格納する硬貨)
    in
      mu(硬貨処理機器,
        硬貨投入口 |-> 硬貨投入口クリア(硬貨処理機器.硬貨投入口),
          硬貨チューブ |-> 硬貨チューブに格納(硬貨チューブへ格納する硬貨, 硬貨処理機器.硬貨チューブ),
          硬貨金庫 |-> 硬貨金庫に格納(硬貨金庫へ格納する硬貨, 硬貨処理機器.硬貨金庫));
```

上記の定義では，let を用いて

- 硬貨チューブへ格納する硬貨と
- 硬貨金庫へ格納する硬貨

の局所変数を定義することで明示されている．実際の計算式は

- 硬貨チューブに格納可能な枚数 () と
- 硬貨 map 差 ()

に定義されている．

硬貨 map 差 () の定義を以下に示す．

```
    硬貨map差:硬貨map型 * 硬貨map型 -> 硬貨map型
    硬貨map差(x, y) ==
      if (x = {|->})
      then {|->}
      else let i in set dom(x)
            in {i |-> x(i) - y(i)} munion 硬貨map差({i} <-: x, {i} <-: y)
      pre dom(x) = dom(y);
```

関数の硬貨 map 差 () は，再帰的に定義される．

4.2.5 自動販売機

自動販売機の仕様を「商品処理機械」と「金銭処理機器」と「商品選択機器」のモジュールをインポートして定義する．

```
module 自動販売機
imports
  from 商品処理機器 all,
  from 金銭処理機器 all,
  from 商品選択機器 all
exports all
definitions
...
end 自動販売機
```

自動販売機は上記の機器に加えて

- 販売ボタンとラックの関係および
- 価格

の情報が必要となる．販売ボタンとラックの関係は「販売ボタンとラックの関係型」，価格は「価格型」として自動販売機型のメンバーに追記する．

```
types
  自動販売機型::
    商品処理機器: 商品処理機器`商品処理機器型
    金銭処理機器: 金銭処理機器`金銭処理機器型
    商品選択機器: 商品選択機器`商品選択機器型
    販売ボタンとラックの関係: 販売ボタンとラックの関係型
    価格: 価格型;
```

販売ボタンとラックの関係型について考える．仕様書には「販売ボタンとラックの間は多対多の関係である」と記述されていた．これは

- 複数のボタンが同じ商品である
- 複数のラックが同じ商品である

などを含意していると想定される．したがって，観点によって販売ボタンとラックの関係の捉え方が変わる．商品

販売の観点から見ると

- 販売ボタンに対して複数のラックが対応している

という捉え方が適切で，仕様書ではこれを「当該ラック」と呼んでいた．

「販売ボタンとラックの関係型」は観点によって捉え方が違うので，観点に応じた適切な型を複合型で表現する．販売ボタンとラックの関係型の定義を以下に示す．

```
types
  販売ボタンとラックの関係型::
    当該ラック: 当該ラック型;

  当該ラック型 = map 販売ボタン ID 型 to set of ラック ID 型;
  価格型 = map 販売ボタン ID 型 to 金額型;
```

販売ボタンとラックの関係型に「当該ラック型」のメンバーを定義している．当該ラック型は「販売ボタン ID 型からラック ID 型の集合への写像」として定義した．これとは逆に，販売ボタンとラックの関係型を「ラック ID 型から販売ボタン ID 型の集合への写像」と見ることができる．新しい観点の型が必要になれば，新しい型に名前をつけて複合型のメンバーとして追加すれば良い．

自動販売機の購入の操作の定義を以下に示す．

```
  購入:販売ボタン ID 型 * 自動販売機型 -> 自動販売機型
  購入 (販売ボタン ID, 機器) ==
    let ラック id in set 機器. 販売ボタンとラックの関係. 当該ラック (販売ボタン ID) in
      mu(機器,
        商品処理機器 |->
          商品処理機器‘商品排出 (ラック id, 機器. 商品処理機器),
          金銭処理機器 |-> 金銭処理機器‘格納 (機器. 金銭処理機器));
```

購入は，販売ボタン ID と自動販売機の操作前の状態を引数にとって，自動販売機の操作後の状態を返す関数として定義する．上の定義では

- 当該ラックのラックを 1 つ選ぶ
- 選んだラックの商品を排出する
- 金銭を格納する

ことを表現している．「当該ラックのラックを 1 つ選ぶ」は `let` を用いて定義されている．上記の `let` の定義では，「ラック id は販売ボタン ID で指定された当該ラック（ラックの集合）から 1 つのラックが非決定的に選択される」ことを記述している．

プログラミングでは，この非決定的な記述は決定的な記述に詳細化する必要がある．仕様の段階では「集合のうちの任意の 1 つ」ということが重要である．この非決定性を形式仕様言語を用いて上記のように陽に記述することができる．非決定的な記述は，簡略な記述を仕様記述者に提供するとともに，後の工程で詳細化すべきことを明確にする効果がある．

5. 考察

形式仕様言語を用いてソフトウェアの仕様記述を分析・記述した過程を振り返って，「形式化による記述の統一性」と「計算機による仕様記述支援」と「形式的な記述のための準備」の観点から考察する．

5.1 形式化による記述の統一性

本事例では，VDM-SL の関数スタイルによる機能の記述にテンプレートを与えて，記述の統一性を図った．このテンプレートは，記述者に「型」と「述語と計算式」と「操作」を形式的に定義することを強制する．仕様を形式的に定義するためには，「名前」と「意味」を記述する必要がある．形式的な定義は対象領域の「辞書」の作成に他ならない．

仕様記述の本質は，対象領域の語彙に名前をつけてその意味を定義することである．形式的な仕様記述を見れば，その記述者が対象をどのような概念で捉えているかが明示される．仕様のレビューでは，名前を通して，その概念や名前の適切さが議論される．

形式的な定義がわかりにくい場合の原因は

- 名前が不適切と
- 概念が未整理

の場合が考えられる．「型」の名前はものを表すので名詞となり，「述語」の名前はその性質を表す語句で，「操作」は動詞となる．名前が適切であれば，仕様書は自然に読みやすいものになる．これは，プログラムにおいて変数名と関数名が適切でなければ，そのプログラムの可読性は低くなるのと同じである．仕様書の段階で語彙が適切に定義されていれば，可読性が高い設計書やプログラムの作成に繋がることが期待できる．

VDM-SL では「未定義」の場合には，`undefined` と記述することができる．また，データ型を決めない場合には `token` 型を利用することができる．これは考慮していないことを陽に表すものである．自然言語の仕様書では，記述の漏れや間違いなのか，あえて記述していないのか判断することができない．形式仕様言語を用いることによって，「書いてあること」と「書いてないこと」が明確になることから，仕様に対する本質的な議論が促進される．

5.2 計算機による仕様記述支援

仕様記述における形式仕様言語の導入効果として，計算機による仕様記述の支援があげられる．計算機による支援として

- 型検査による語彙の一貫性保証
- プロトタイピングによる妥当性確認

がある．

仕様記述言語の型検査では，プログラミング言語の型検査と同じ効用が期待できる．型検査は，変数名や関数名を変更した際に，その修正の構文的な誤りがないことを検査することができる．仕様書では特に，「行なう」「行う」などの送り仮名の違いの検出に有効である．

本稿で用いた VDM-SL の関数は実行可能な関数（陽関数）なので，プロトイピングとして計算機上で仕様を実行して，仕様の妥当性を確認することができる．標準化したテストケースを用いることにより，テストケースの自動実行も容易に実現できる．ここで作成したテストケースは，プログラミングによる詳細化と対応づけることにより，プログラムのブラックボックステストケース生成支援に応用することが期待できる．

5.3 形式的な記述のための準備

形式仕様言語を導入するための最も大きな障壁は

- 形式化の意義と
- 対象を表現するための道具としての数学の理解

であると考えられる．

形式化の意義を理解するためには，形式言語における「構文と意味」の分離を理解することが鍵となる．これは，命題論理と第 1 階述語論理の教科書に必ず書かれている．形式化は「形」に着目することであるが，それは意味を排除するのではなく，形による意味が定義されていることが重要である．型検査が計算機で実行できるのは，その背後に型が正しいことを計算するための形式的な定義があるからである．この形式的な定義の存在が型検査の信頼性を保証している．

モデル規範型形式仕様言語を利用するための数学的な素養として

- 集合と関係と第 1 階述語論理

があげられる．特に数学における関数は「集合間の要素の対応」であることを理解することが重要である．C 言語に代表される手続き型プログラミング言語で用いられる関数と，数学における関数の違いを理解せずに，モデル規範型形式仕様記述言語を利用することは難しいかも知れない．数学の関数を理解を促進する手段として，関数プログラミングの習得は有効であろう．

6. おわりに

本稿では，ソフトウェアの機能仕様を分析・記述するための VDM-SL を用いた記述法を提示して，既存の公開されている実用的な仕様書に対して，その記述法を適用して得られた形式仕様記述を通してその有用性と課題を議論した．今後，既存の仕様書と形式仕様記述の間の対応関係を分析し，全ての形式仕様記述を公開する予定である．

謝辞 本研究の一部は，JSPS 科研費 24500049, 24220001，2015 年度南山大学パッヘ研究奨励金 I-A-1 の助成を受けて実施した．

参考文献

[1] Hall, J.A.: Seven Myths of Formal Methods, IEEE Software, 7(5): 11-19 (1990).
[2] 栗田太郎：携帯電話組込み用モバイル FeliCa IC チップ開発における形式仕様記述手法の適用，情報処理，Vol. 49，No. 5，pp. 506.513 (2008).
[3] ASTER 自動販売機ハードウェア構成および販売者用機能仕様，http://aster.or.jp/business/contest/doc/2015tdc-v1_1.zip.
[4] Spivey, J.M.: The Z Notation, Prentice Hall (1992), http://spivey.oriel.ox.ac.uk/ mike/zrm/zrm.pdf.
[5] 来馬啓伸：B メソッドによる形式仕様記述，近代科学社 (2007).
[6] Fitzgerald, J. and Larse, P.G.: Modelling Systems, Cambridge, (2009).
[7] 荒木啓二郎，張漢明：プログラム仕様記述論，オーム社 (2002).