

MVC アーキテクチャに基づく形式仕様と 仕様記述プロセスに関する考察

張 漢明¹ 野呂 昌満¹ 沢田 篤史¹

概要：ソフトウェア開発における形式仕様導入の障壁として、形式仕様の難解さと記述のための適切な指針がないことがあげられる。本稿では、形式仕様のモデルを MVC アーキテクチャの概念に基づいて定義し、詳細化関係を考慮した構成要素および要素間の関係を整理する。

Consideration on formal specification and the process of specification description by applying MVC architecture

HAN-MYUNG CHANG¹ MASAMI NORO¹ ATSUSHI SAWADA¹

Abstract: Lack of appropriate guidelines on describing formal specification becomes a barrier to successful introduction of formal methods to software development practices. In this paper a model of our formal specification is designed borrowing the concept of MVC architectural style. We organize based on this model the constituent elements and relationships between them taking refinement relations into account.

1. はじめに

ソフトウェア開発の仕様記述に形式仕様を導入することは、ソフトウェアの信頼性を向上させるための有望な手段である [2]。日本では、Felica IC チップの開発における、VDM[11], [12] を用いた形式仕様技術の適用事例が有名である [5]。しかし、実際の一般的なソフトウェア開発において、形式仕様は実用化されていない。形式仕様の実用化を阻害する要因として、

(1) 記述モデルの多様性と、

(2) 形式仕様の記述プロセスがないこと

があげられる。形式仕様言語を用いて仕様を記述するためには、記述する対象の性質に応じて、適切な記述モデルおよび表現方法を選択する必要がある。仕様を検証するためには、検証に適した表現があり、仕様の翻訳が必要な場合がある。また、対象の性質や検証したい性質に応じて、形式仕様を記述するための明確なプロセスは存在しない。

本研究の目的は、形式仕様の多様な記述モデルを整理し

た形式仕様モデルを定義し、形式仕様モデルに基づいた仕様記述のプロセスを提示することである。形式仕様言語には記述モデルが存在して、形式仕様言語はそれらの記述モデルに対応づけられる。形式仕様に基づいた仕様の検証は、記述モデルに依存する。仕様には、記述に適した表現や検証に適した表現などがあり、それぞれ適した記述モデルがあると考えられる。記述モデル間の互換性を明らかにして、同じ仕様に対して適切な表現を提供することができれば、形式仕様の実用化が促進されることが期待できる。

本研究では、形式仕様における記述モデルの多様性に着目し、形式仕様モデルを MVC アーキテクチャに対応づけて、記述モデルおよび表現方法と、形式仕様を構成する本質的な概念を分離する。MVC アーキテクチャは、対象の表現方法（ビュー）と対象への制御（コントローラ）に着目して、ビューとコントローラに依存しない抽象化した概念（モデル）を分離した構造である。形式仕様の表現方法を抽象化した記述モデルがビューに対応し、抽象化された仕様記述プロセスの要素技術がコントローラに対応する。形式仕様モデルに基づいて、特定の形式仕様言語に依存しない概念で、仕様記述のプロセスを定義することにより、

¹ 南山大学
Nanzan University, Showa, Nagoya 466-8673, Japan

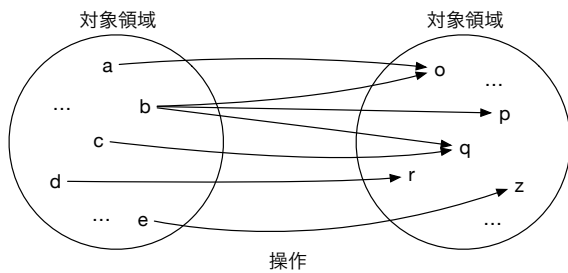


図 1 仕様概念

形式仕様の実用化をはかる。

本稿では、MVC アーキテクチャに基づいた形式仕様モデルについて説明する。モデルでは、形式仕様の抽象的な概念を整理した。ビューでは、状態に着目した表現方法（状態規範）と操作に着目した表現方法（操作規範）の構成要素を示し、状態規範と操作規範の関係、および、仕様間の詳細化関係を整理した。コントローラでは、仕様の工程を仕様記述、詳細化関係記述、正当性検証記述として整理した。形式仕様のモデルに基づいて、形式仕様導入の効果と難しさについて議論する。

以降、2 節では、ソフトウェア開発における仕様とその役割について述べ、本研究で想定する仕様の概念を説明する。3 節では、形式仕様を規定する形式仕様モデルを提示して、形式仕様導入の効果と難しさについて 4 節で議論する。

2. 仕様の定義とその役割

本節では、本研究で想定する仕様について説明する。まず、抽象的な仕様の概念を定義して、要求、仕様、プログラム間の関係を明らかにする。次に、仕様を分類して、仕様間の関係と仕様の例示について述べる。

2.1 仕様の定義

ソフトウェアにおける「仕様」とは、「対象」と対象に対する「操作」を規定するものと捉える。対象を規定することは、対象の集合すなわち「対象領域」を定義することである。操作を規定することは、対象間の「関係」を定義することである。仕様を以下のように定義する。

仕様 = (対象領域, 操作)

対象領域 = 対象の集合

操作 = Rel(対象, 対象)

上記で、(X,Y) は X と Y の 2 項組、Rel(X,Y) は X と Y の関係を表す。

仕様が対象領域と操作の 2 項組で規定されること概念を図 1 に示す。対象領域である対象の集合は丸で、対象と対象の関係である操作は矢印 (→) の集合で表されている。矢印は対象と対象の 2 項組であり、図 1 では操作が 2 項組の集合

{(a,o), (b,o), (b,p), (b,q), (c,q), (d,r), (e,z),

...}

として表現されている。

操作は、対応が一意に定まる関数ではなく、関係として表現されている。これは、対象が関係で定まる複数の対象のうち、どれか 1 つに対応することを表している。図 1 では、対象 b は、操作によって対象 o,p,q のどれかに対応する仕様であることを表している。仕様では、このような非決定的な記述が許されており、また、非決定的であることを仕様として記述することが重要である。

仕様を記述するためには、対象領域と操作を何らかの方法で表現する必要がある。仕様は、自然言語、UML 図、状態遷移表、形式仕様言語などの図や表などを含めた「言語」を用いて表現される。一般的には、仕様は複数の言語で記述される。また、この仕様の定義では、プログラミング言語で記述されたプログラムも仕様とみなすことができる。

仕様を記述するためには、対象領域で用いられる「概念」を明らかにして、複数の開発者間で概念を「共有」することが重要である。対象領域の概念は、概念に名前をつけて、その意味を定義することによって規定される。この意味は、何らかの言語を用いて表現される。概念は必ずしも名前がつけられるわけではないが、概念を共有するためには、

- 概念に適切な名前をつけることと、

- その意味を厳密に定義すること

が重要である。仕様を記述することは、

- 対象領域の「辞書」を作成すること

とみなすことができる。辞書は、対象領域の用語とその意味を定義したものである。

2.2 要求・仕様・プログラムの関係

ソフトウェア開発における、要求、仕様およびプログラムの関係を定義する。A が B に依存していることを $A \rightarrow B$ と表記すれば、これらの関係は

- 仕様 \rightarrow 要求
- プログラム \rightarrow 仕様

となる。 $A \rightarrow B$ は、A は B を満たす、A は B を含意するともいう。依存関係 (\rightarrow) において推移性が成り立つとすれば、「プログラム \rightarrow 要求」という関係が成り立つ。要求と仕様は全て記述されているとは限らないが、「概念的には」存在する。極端な場合、要求と仕様の記述はないが、プログラムだけが存在する場合が考えられる。この場合でも、要求と仕様は記述されていないが、概念的には存在しているとして、プログラム \rightarrow 仕様、仕様 \rightarrow 要求が成り立つと考えられる。

2.2.1 妥当性確認と正当性検証

仕様は、要求およびプログラムに対して正しさの基準としての役割を果たす。それぞれの依存関係を調べることを、妥当性確認および正当性検証という。

妥当性確認 「仕様 \rightarrow 要求」が成り立つことを調べる。

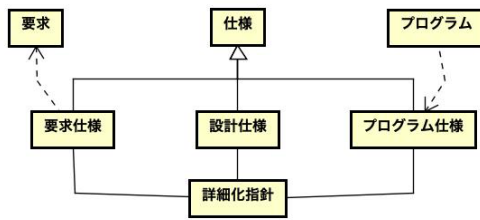


図2 仕様の種類

正当性検証 「プログラム→仕様」が成り立つことを調べる。

妥当性確認では、仕様は十分に記述されているが、「要求は十分に記述されていない」ことを前提とする。「十分に」という表現が曖昧である。「十分な記述」とは、依存関係が満たされていることを客観的に判定することが可能な記述とする。要求では、「計算時間が速い」は許されるが、仕様としては十分な記述ではない。「計算時間が1秒以内」は仕様として十分な記述である。一方、正当性検証では、プログラムと仕様はどちらも十分な記述であることを前提とする。以降、プログラムと仕様の記述は十分であるとする。

妥当性確認では、仕様が要求を満たすことを

- 仕様と要求のレビュー、および
- 仕様もしくはプログラムの実行

によって調べる。正当性検証では、プログラムが仕様を満たすことを

- プログラムと仕様のレビュー、
- プログラムの実行によるテスト、
- プログラムの証明

によって調べる。妥当性の確認は、要求が十分に記述されていないので、証明することができない。

2.3 仕様の分類と仕様間の関係

仕様は、要求を分析して、抽出もしくは導出することから獲得する。要求の記述で、正当性検証が可能な要求の記述は、仕様の記述となる。仕様記述の作業は、様々な要求から、客観的な記述を仕様としてまとめる作業である。プログラムが要求を満たすことは、仕様を媒介して調べられる。要求を満たすプログラムを作成するためには、いかにして仕様をまとめ、構成、表現することが重要である。要求、仕様、プログラムは、それぞれの作成を繰り返すことにより、要求、仕様、プログラムの概念が明確になり、徐々に適切な表現で記述される。

仕様を構成する上で、仕様の分類を図2のクラス図に示す。仕様とプログラムの間にはギャップがあり、仕様から段階的に詳細化されることによりプログラムと同等な仕様が形成される。この仕様の形成の過程で仕様は、

- 要求仕様、
- 設計仕様、

● プログラム仕様

に分類される。それぞれの仕様と仕様間の関連および要求との関連を以下で説明する。

要求仕様 要求仕様は、要求から獲得される仕様である。

要求仕様は要求に依存し、要求仕様の正しさは、要求に対する妥当性確認によって得られる。

設計仕様 設計仕様は、要求仕様の詳細化、もしくは、設計仕様の詳細化として得られる。要求仕様と設計仕様の間には、詳細化指針を用いて、

設計仕様と詳細化指針→要求仕様

という関係がある。つまり設計仕様は、詳細化指針に基づいて要求仕様を満たしていることを検証することができる。また、設計は段階的に行われることが想定されるので、

設計仕様と詳細化指針→設計仕様

という関係がある。

プログラム仕様 プログラム仕様は、設計仕様（もしくは要求仕様）の詳細化として得られる。プログラム仕様と設計仕様の間には、詳細化指針を用いて、

プログラム仕様と詳細化指針→設計仕様

という関係がある。プログラムはプログラム仕様に依存し、プログラムの正しさは、プログラム仕様に対する正当性検証によって得られる。

設計仕様とプログラム仕様では、詳細化指針を加えて「仕様→仕様」の関係であるから、設計仕様とプログラムの仕様の正しさは、正当性検証によって得られる。ここで、仕様間の関係に「詳細化指針」が存在することが重要である。詳細化指針には、例えば、

- 集合で表現したものを、リストで表現

といったものがある。集合をリストで表現する場合、リストで表現される要素の順番や重複した要素は、集合では無関係である。このような関係は、仕様だけでは推測しかできない。リストで表現されたものが集合を表していることは、詳細化関係の記述が無ければわからない。

2.4 仕様の例示

仕様は、全ての対象を対象領域として、また、全ての操作を対象の関係として表現する必要がある。しかし、全ての対象と全ての操作を列挙することは、現実的には不可能である。仕様を何らかの方法で、言語を使って一般化して表現する必要がある。一般化した仕様を記述することや読むことは、容易であるとは限らない。

仕様を具体的なもので表すことを、「仕様の例示」ということにする。例示された仕様は、具体的な記述なので、記述の内容はわかり易いと考えられる。仕様の例示は、仕様の一部とみなすことができる。仕様の例示はあくまでも仕様の一部であると認識して、一般的な表現で仕様を記述する努力が求められる。

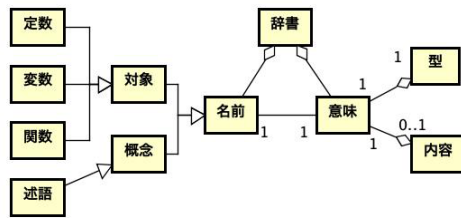


図3 モデルの構成要素

3. 形式仕様モデル

形式仕様の概念を整理するために、MVC アーキテクチャの概念を形式仕様に応用して、形式仕様の特性を規定する形式仕様のモデルを提示する。形式仕様は、形式仕様言語を用いて記述された仕様である。仕様は、対象領域と操作を規定するが、仕様の表現方法は、形式仕様言語によって違う。仕様の表現は、MVC のビュー (V) に相当し、表現に依存しない概念をモデル (M) と分離して整理する。コントローラ (C) は、仕様の工程 (プロセス) に対応する。形式仕様を

モデル 形式仕様の表現に依存しない共通の概念

ビュー 形式仕様の表現

コントローラ 形式仕様のプロセス

の観点から整理する。

3.1 形式仕様の定義

形式仕様を以下のように定義する。

形式仕様 形式言語を用いて表現したもの

形式言語 「構文」と「意味」が規定されたもの

プログラムは実行可能な形式言語で記述された表現なので、プログラムを実行可能な形式仕様とみなす。

3.2 モデル

形式仕様の表現に依存しない構成要素と仕様間の関係を提示する。

3.2.1 構成要素

形式仕様の共通の概念を、「仕様」「辞書」「形式言語の構成要素」の観点から分析する。仕様は、対象と対象領域の概念を規定するものである。形式仕様では、対象や概念に名前をつけて、対象領域を厳密に規定する。形式言語には共通の構成要素として、定数、変数、関数、述語、型があるとみなした。モデルは、これらの関連を整理してまとめたものである。

モデルの構成要素を、クラス図を用いて図3に示す。辞書は、名前と意味から構成されている。名前には、対象を表すものと、概念を表すものがある。これらはそれぞれ、形式言語の述語論理で一般的に規定されている、定数、変数、関数、および、述語に対応する。

意味は、型と内容から構成されている。型は、対象や概



図4 仕様間の関係

念の種類を規定する。例えば、ある概念 P が集合 A の要素と集合 B の要素で表現される場合は、P の型は、A と B の直積から真理値への関数 ($A \times B \rightarrow \text{Bool}$) と表すことができる。内容は、名前づけられた対象や概念を具体的に定義する。

形式言語では、型と内容を分離して、内容が未定義を許すものがある。仕様では、この未定義を記述することが重要な役割を果たす。仕様の段階では、詳細な構造や、概念が未整理で厳密な定義が与えられない場合がある。また、詳細な構造は無視して、抽象化された構造で性質を記述することが有効な場合もある。仕様を記述する場合、対象や概念の名前と型を決めておけば、それらを使って仕様を記述することが可能となる。ここで、詳細化されていない仕様の段階では、この「内容は未定義である」という情報が重要である。

3.2.2 仕様間の関係

仕様間の関係には、含意 (\rightarrow) の関係がある。仕様 A が仕様 B を含意することを

仕様 A \rightarrow 仕様 B

と記述する。含意の関係は、「仕様 A が仕様 B を満たす」や「仕様 A ならば仕様 B」という場合もある。

仕様間の関係を、クラス図を用いて図4に示す。含意関係は、前件と後件から構成されている。「仕様 A \rightarrow 仕様 B」の仕様 A が前件、仕様 B が後件である。仕様では、対象領域の特性を性質として記述する。性質には、公理と定理がある。公理は、対象領域の性質を決定づける最小の性質とみなすことができる。定理は、公理と定理から含意関係を用いて導出することができる性質である。仕様では、論理の言葉では、理論、つまり無矛盾な公理集合を定義することが求められる。

3.3 ビュー

形式仕様の表現には、状態規範と操作規範の2種類の考え方があることを示し、仕様間の関係について整理する。

3.4 構成要素

状態規範と操作規範の形式仕様に関する概念を整理して、構成要素として提示する。ビューの構成要素を、クラス図を用いて図5に示す。

3.4.1 状態基盤の形式仕様

状態基盤の形式仕様は、対象の状態に着目して内部状態の構造を表現し、操作を事前状態と事後状態の関係として

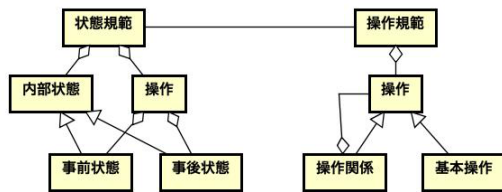


図 5 ビューの構成要素

表現する。スタック (Stack) を例として状態基盤の考え方を以下に示す。

- 状態領域：要素，Stack
- 内部構造
 - － 要素：内部構造を規定しない
 - － Stack：順番を保持するデータ構造で表現
- 操作：pop
 - － 事前条件：空 Stack ではない
 - － 事後条件：Stack の最後の要素を取り除く

Stack の状態領域としては，要素と Stack が必要である。Stack の内部構造として，配列やリストなどの要素の順番を保持するデータ構造を用意する。pop は空 Stack に対して使うことができない。これが，事前条件として表現される。事後条件は pop の効果を事後条件として表現する。最後の要素は，要素の順番の保持の仕方に依存する。

3.4.2 操作基盤の形式仕様

操作基盤の形式仕様は，対象領域で分割することができない操作を基本操作として表現して，操作間の関係に着目して操作を表現する。

- 状態領域：要素，Stack
- 操作：push, pop, top
- 操作の関係

```

Stack = Stack(0)
Stack(0) = push → Stack(1)
Stack(n) = push → Stack(n+1) []
          pop → Stack(n-1) [] top → Stack(n)
          ただし n > 0
    
```

Stack の操作に着目し，Stack の操作である push, pop, top の順番関係を表現している。操作の順番を表現するために，保持している要素の数を Stack のパラメータとして保持している。

3.4.3 状態基盤と操作基盤の関係

仕様を対象領域と操作の 2 項組とみなした場合，状態基盤でも操作基盤でも，原理的には同じ仕様を表現することが可能であると推察できる。上記の Stack の例では，状態基盤の仕様で，事前条件と事後条件を完全に表現すれば，操作基盤の仕様を記述することができる。逆に，操作基盤の仕様で，Stack のパラメータとして要素を順番に保持すれば，状態基盤の仕様を記述することができる。

ただし，操作基盤の仕様で全てを記述しようとするれば，

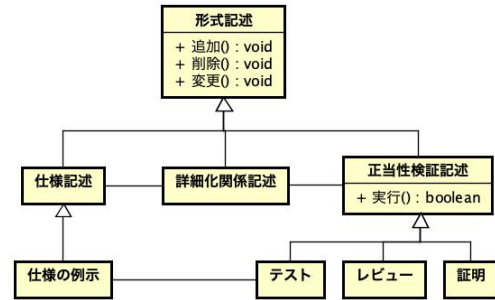


図 6 コントローラの対象

Stack の例でもわかるとおり，Stack のパラメータに状態を保持する必要がある。操作に全てを集約すると，操作に状態を内包することになる。仕様記述では，表現したい内容を適切な言語で記述することが重要である。この時，必要のない概念を捨象して抽象化することが必要である。操作基盤の Stack の例では，Stack の内部状態を捨象することにより，Stack 操作の順番に関して明確な仕様となっている。

3.4.4 詳細化関係

モデルにおける仕様間における詳細化関係について，状態基盤と操作基盤について考える。操作基盤の詳細化関係は，内部状態の詳細化とみなし，操作基盤の詳細化関係は，操作もしくは操作関係の詳細化とみなすことができる。

状態基盤における内部状態の詳細化は，内部構造の定義を詳細化することである。Stack の例では，要素の内部データを詳細化することに相当する。詳細化には，情報の具体化，追加，細分化などが考えられる。操作基盤における操作もしくは操作関係の詳細化では，操作の追加や細分化などが考えられる。

さらに，非決定的な仕様を決定的にする決定化関係がある。事後条件が複数の可能性が許されている場合，そのうちのどれかを選択することは決定化の詳細化関係となる。また，操作の順番として複数の可能性が許されている場合，そのうちの順番を決定することも決定化の詳細化関係となる。

3.5 コントローラ

コントローラは，形式仕様の作成工程とみなすことができる。ビューの構成要素に対する操作を，クラス図を用いて図 6 に示す。操作の対象は形式記述であり，操作として一般的な編集機能「追加」「削除」「変更」などが考えられる。形式仕様の作成工程として

- 仕様記述
 - 詳細化関係記述
 - 正当性検証記述
- がある。

仕様記述が正しいことは、仕様記述の正しさの基準として、詳細化関係記述が概念的に存在する。詳細化関係が満たされていることを調べるための記述として、正当性検証記述がある。正当性検証記述を実行すると、その真偽がわかる。

正当性検証には、「テスト」「レビュー」「証明」がある。テストは、仕様の例示を用いて正当性の検証を実行する。レビューと証明は、仕様記述と仕様の例示に対しても行うことができる。ここで重要なことは、正当性検証を行うためには、詳細化関係記述が必要なことである。詳細化関係を厳密に記述することは困難であるが、形式仕様を用いることによりその役割が明確になる。

4. 考察

これまでに提示した形式仕様のモデルを基に、

- 期待される形式仕様導入の効果
- 形式仕様の難しさ

の観点から考察する。

4.1 期待される形式仕様導入の効果

4.1.1 対象領域の概念形成支援

仕様の最終成果物を「辞書」とみなせば、開発者間で利用する「用語」を決定する作業が概念形成に相当する。対象領域を表現するための「名前」とその「意味」を定義する作業である。形式言語を用いると、名前づけと意味定義の作業が分離されて、仕様記述の作業が明確になる。

名前をつける場合には、図3に示されている「対象」なのか「概念」なのかを考える必要がある。さらに、対象を「定数」「変数」「関数」で表現することにより、対象が明確になる。変数は、対象領域にある対象を一般的に表現することが可能になる。変数の導入による対象の一般化は、記号化による最も大きな効果である。

形式仕様は、状態の内部構造を無視したり、意味を未定義にするという意味合いで、未完成の記述が可能である。未定義を導入することにより、表現されているものと、表現されていないものが、明確になる。記述されたことが明確なので、記述した人の考えが形式仕様を通じて明確になる。

形式言語を用いて、名前と意味を記述する過程で、同じ構造を発見することにより、概念が浮かび上がることがある。意味の記述が複雑であれば、より簡単な構造を模索するきっかけになる。本質的に複雑な構造をしているのか、より簡単な概念が存在するのかを吟味する作業が生じる。

このようにして、形式仕様を作成する過程を通して、対象領域に対する共通概念が形成される。形式仕様の最終的な意義は、意味の記述ではなく、名前かも知れない。開発者間で意思疎通ができる概念を共有することにより、顧客への説明も明確になることが期待できる。厳密な意味定義

は、正当性検証を行うために必要な記述である。

4.1.2 コンピュータによる作業の自動化支援

仕様を形式的に記述すると、コンピュータによる作業の自動化が期待できる。コンピュータによる自動化として、

- 仕様記述の型検査
- 正当性検証の自動化
- テストケース生成およびテストケースの自動実行

があげられる。実際の開発で使用されている形式仕様言語として、VDM[11], [12], B メソッド [10], SPARK Ada[1], [7], [8] などがあげられる。仕様記述の型検査は、どの形式言語でも、通常のプログラミング言語と同様な静的な検査を行うことができる。

VDM は、手続き指向、関数指向、論理指向、オブジェクト指向などの多様な表現が可能である。言語のサブセットとして、実行可能な仕様記述がある。仕様のテストケースを作成することにより、テストケースを自動で実行することが可能である。また、仕様記述の静的な意味を保証するための、検証条件を自動生成することができる。Felica IC チップの開発における VDM を用いた形式仕様の適用事例では、実現に依存しない抽象化した厳密な形式仕様を記述し、テストを用いて仕様に対する高い信頼性を得ることが、最終成果物であるプログラムコードの品質向上に寄与することが報告されている [5]。

B メソッドは、抽象的な仕様から、段階的な詳細化の過程を全て厳密に記述して、図6における、証明を用いた正当性検証を行う環境を提供している。B メソッドは、記述法として開発された、集合と述語論理に基づいた形式仕様表記法 Z[9] を発展させた、証明を前提とした開発手法である。

SPARK Ada は、主に組み込みシステムで利用されているプログラミング言語に、図5における、状態規範の形式仕様を記述することができる。Ada 言語のサブセットである。SPARK Ada では、事前条件と事後条件を仕様として記述し、プログラムの正しさを保証するための、検証条件を自動生成する機能がある。検証条件は、SMT ソルバーを用いて自動検証を試みる。Spark Ada は実行可能なプログラミング言語なので、プログラム作成と並行して利用が可能である。

4.2 形式仕様の難しさ

一般のソフトウェア開発で、形式仕様を導入するさいの難しさについて

- 形式言語の習得
- 抽象化と一般化の能力

の観点から議論する。

形式仕様言語を習得するさいの困難さの要因として、

- 新しい記法習得の困難さと、
- 数学の述語論理と集合を用いた記述法の問題理解の困

難さ

の2つが考えられる。形式仕様言語の記法習得の困難さは、新しいプログラミング言語の記法習得の困難さと同じ程度と考えられる。必要になれば新しい記法の習得は問題ない。問題は、述語論理と集合の概念理解である。述語論理と集合の記法は単純である。難しいのは、対象を数学の独特な表現で記述することである。しかし、例えばZ表記法のリファレンスマニュアル[9]には、よく使われる便利な数学ライブラリが提供されている。ライブラリを理解することが仕様記述言語を習得する鍵となる。

形式仕様記述の難しさは、対象を抽象化および一般化して、記号を用いて漏れなく厳密に記述することである。表記法の難しさではなく、対象の性質の本質を抽出して、関係のないものを捨象する能力である。形式仕様言語を用いて仕様を記述すると、対象をどのように捉えるかが明らかになる。また、言語が考え方に影響を与える。したがって、記述の性質に対して適切な表現方法を選択することが求められる。本研究では、このような仕様記述の支援を目指す。

5. おわりに

本稿では、形式仕様の概念とその役割を明確にするために、MVCアーキテクチャの概念を形式仕様の分析に適用し、形式仕様のモデルを提示した。

- モデルでは、仕様は辞書とみなして、形式仕様に通ずる概念を辞書と対応づけた。
- ビューでは、状態規範と操作規範の2つの表現を提示して、それらの構成要素と、状態規範と操作規範の関係を示した。
- コントローラでは、仕様の工程を仕様記述、詳細化関係記述、正当性検証記述として、それらの位置付けを示した。

形式仕様を導入したさいに期待される効果として、以下があげられる。

- 仕様を名前と意味に分離することで、記述が明確になり、厳密な記述を基にして、開発者間の対象領域に対する理解が深まる。
- 仕様の段階で、型検査やデータフローなどの静的な検査や、テストケースの自動実行などが可能になる。

逆に、困難な点として、

- 形式仕様記述の難しさ

がある。これは、慣れの問題で、プログラムを記述する能力が高い人は、仕様を記述する能力も高いと思われる。

本稿で提示した仕様モデルは十分ではなく、仕様記述のプロセスはまだ提示されていない。今後の課題として、

- 仕様モデルの詳細化と洗練
- 仕様記述プロセスの提示

があげられる。また、自動販売機の仕様[6]などの公開されている仕様に対して形式仕様を適用することにより、形

式仕様のモデルの妥当性を検証する。

謝辞 本研究の一部は、JSPS 科研費 16K00110、2018年度南山大学パツへ研究奨励金 I-A の助成を受けて実施した。

参考文献

- [1] Ada Conformity Assessment Authority: Ada Reference Manual, ISO/IEC 8652:2012(E). 3rd edn, 2012.
- [2] Hall, J.A.: Seven Myths of Formal Methods, IEEE Software, 7(5): 11-19 (1990).
- [3] IEEE Standard Board: IEEE Recommended Practice for Software Requirements Specifications, Std 830-1998 (1998).
- [4] Jackson, M.: Software Requirements and Specifications, ACM Press (1995).
- [5] 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).
- [6] 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>.
- [7] SPARK Team: SPARK 2014 Reference Manual, AdaCore, <http://docs.adacore.com/spark2014-docs/html/lrm/>.
- [8] SPARK Team: SPARK 2014 Toolset User's Guide, AdaCore, <http://docs.adacore.com/spark2014-docs/html/ug/>.
- [9] Spivey, J.M.: The Z Notation, Prentice Hall (1992), <http://spivey.oriel.ox.ac.uk/mike/zrm/zrm.pdf>.
- [10] 来馬啓伸: B メソッドによる形式仕様記述, 近代科学社 (2007).
- [11] Fitzgerald, J. and Larse, P.G.: Modelling Systems, Cambridge, (2009).
- [12] 荒木啓二郎, 張漢明: プログラム仕様記述論, オーム社 (2002).