Vol.2019-ARC-240 No.43
Vol.2019-SLDM-191 No.43
Vol.2019-EMB-53 No.43
2019/2/28

MVCに基づいた組み込みソフトウェアの形式仕様メタモデルに関する考察

張 漢明[1]　野呂 昌満[1]　沢田 篤史[1]

概要：本研究の目的は，組み込みソフトウェアにおける形式仕様のメタモデルを提示することである．本研究の基本的なアイデアは，対象に対する機能と振る舞いの 2 つの視点から，MVC アーキテクチャに基づいて仕様の構造を整理することである．形式仕様のメタモデルは，仕様のメタモデルにおける表現（ビュー）の構造に特徴づけられる．提案するメタモデルは，視点に応じた仕様記述のための，形式的な表現と非形式的な表現が混在する記法定義の基盤となる．

Consideration on formal specification meta-model for embedded software by appling MVC architecture

1. はじめに

組み込みソフトウェア開発において，開発文書の品質を向上するために形式仕様の導入は必要不可欠である．組み込みシステムは，一般的に，複数の対象が並列に動作するので，ソフトウェア制御において並行性を考慮する必要がある．並行性は自然に非決定性を含むので，並行性に関係する性質を，プログラムの段階でテストを用いて検証することは困難である．並行性に関する性質の検証は，設計段階の文書に対して行う必要がある．コンピュータを用いて検証を自動化するためには，文書の形式化は避けられない．

開発文書に形式仕様を導入するためには，対象に対する視点に応じた仕様の書き方の提示が必要である．数学的な概念と記法に慣れていない人にとって，形式手法は難しいと感じられる．形式的な仕様は，数学的な記法だけでしか表現されるわけではない．UML 記法の OCL は，記号化された図式表現を前提とした形式的な表現である．UML 記法は，並行性や時間に関する性質を検証するための言語として十分ではない．対象領域の記述には，その領域の特性に応じた馴染みのある記法が求められる．数学的な表現を部分的に非数学的な表現で代替するには，形式仕様の構造，すなわち形式仕様のメタモデルの分析が必要である．

本研究の目的は，組み込みソフトウェアにおける形式仕様のメタモデルを提示することである．形式仕様には形式言語毎にそれぞれ特性があり，記述したい視点に応じて適切な形式言語を選択する必要がある．組み込みシステムでは，タイミング図を用いた記述が利用される．タイミング図を用いて適切な仕様を記述もしくは抽出することができれば，慣れた記述法を用いて仕様を記述することが可能になる．適切な記法を提供するためには，形式仕様のメタモデルが必要である．

本研究の基本的なアイデアは，対象に対する視点として「機能」と「振る舞い」に着目し，MVC アーキテクチャに基づいて仕様の構造を整理することである．機能と振る舞いの間には相互に依存関係がある．機能の集合を定義すると，事象列の集合すなわち振る舞いが自然に決まる．逆に，振る舞いの集合を定義すると，振る舞いと整合性を保つように機能を定義しなければならない．本研究では，仕様に対する関心事は，仕様の構造，仕様の表現，仕様記述のプロセスである．仕様の表現に依存しない本質的な構造をモデル，仕様の表現をビュー，仕様記述のプロセスをコントローラとみなして，MVC アーキテクチャを用いて分析することは自然である．

形式仕様のメタモデルは，仕様のメタモデルにおける表現（ビュー）の構造に特徴づけられる．形式仕様は，仕様を表現するための手段の 1 つである．形式仕様は，仕様を記述するための自然言語表現，図式表現，表形式表現の代替表現とみなすことができる．形式仕様の表現として，代

[1] 南山大学
Nanzan University, Showa, Nagoya 466–8673, Japan

Vol.2019-ARC-240 No.43
Vol.2019-SLDM-191 No.43
Vol.2019-EMB-53 No.43
2019/2/28

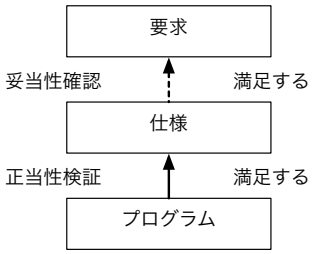

図 1 要求・仕様・プログラム間の関係

数仕様，状態規範型仕様，プロセス代数仕様を分析の対象とした．代数仕様は，内部構造に依存しない機能の関係，すなわち抽象データ型の役割を果たす．状態規範型は機能を対象の状態，入力，出力の間の関係を用いて表現する．プロセス代数は，並行性をインターリーブを用いた事象列とみなして表現する．ソフトウェアに対する時間・空間の制約は機能の表現に付随する．

提案するメタモデルは，視点に応じた仕様記述のための，形式的な表現と非形式的な表現が混在する記法定義の基盤となる．本稿では，実用的な形式仕様言語である Z 言語 [12]，VDM-SL[2], [5] に対して，メタモデルを用いた仕様構造の概略を示す．

2 節では，ソフトウェア開発における仕様と形式仕様の役割について説明する．3 節において MVC に基づいた仕様の構造を示し，4 節で形式仕様の構造について述べる．5 節において，既存の形式仕様言語に対する形式仕様メタモデルの適用ついて考察する．

2. 仕様と形式仕様の役割

ソフトウェア開発文書のおける仕様の役割と形式仕様の役割について述べる．ソフトウェア開発では，解決すべき問題を記述する「要求」，要求を解決するために作成する対象の「仕様」，および，仕様を実現する「プログラム」の記述がある．仕様は，ソフトウェア開発の中心的な役割を果たす．

2.1 仕様の役割

仕様は，

- プログラムに対する正しさの基準と
- 要求を解決するための開発対象の記述

としての役割がある．ここでは，「要求・仕様・プログラムの関係」と，仕様間の関係である「詳細化関係」について説明する．

2.1.1 要求・仕様・プログラム間の関係

要求，仕様，プログラムの間には，図 1 に示すように，「仕様が要求を満足する」と「プログラムが仕様を満足する」という関係がある．本稿では，前者の関係を調べるプロセスを「妥当性確認」，後者の関係を調べるプロセスを「正当性検証」と呼ぶ．妥当性確認は，

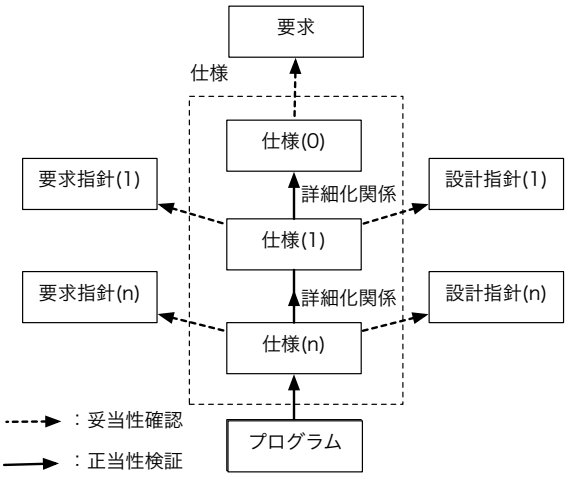

図 2 仕様間の関係

- 要求を満足する正しいソフトウェアを作っているか，

正当性検証は，

- 仕様を満足して正しくソフトウェアを作っているか，

を調べるプロセスとみなされる．妥当性確認では客観的な証拠を提示することは困難であるが，正当性検証では客観的な証拠を提示することが求められる．要求に対する仕様の妥当性が確認されて，仕様に対するプログラムの正当性が検証されれば，プログラムの妥当性が確認できたことになる．プログラムの正しさの検証は，要求に対してではなく，仕様に対するプロセスである．

2.1.2 詳細化関係

仕様には，図 2 に示すように，上位仕様と下位仕様の間に詳細化関係という構造があり，各詳細化の仕様に対して指針がある．この指針には，目的を達成するための指針と，ソフトウェアの再利用などを目的とした設計の指針がある．ここでは，前者を要求指針，後者を設計指針と呼ぶ．仕様は，最も上位の仕様 (0) からプログラムと同じ詳細な仕様 (n) までの仕様の列として表される．詳細化された仕様 (n) に対して，プログラムの正当性を検証する．

詳細化関係が要求指針によるものか設計指針のよるものかを記録することは，ソフトウェアの保守の観点から重要である．要求指針に対応した詳細化関係を変更することは困難であるが，設計指針に対応した詳細化関係は変更可能である．また，詳細化の妥当性を確認するためには指針が必要である．詳細化関係は，（上位仕様，下位仕様，仕様間の関係） の 3 項組で表される．

2.2 形式仕様の役割

ソフトウェア開発における形式仕様の役割について，

- コミュニケーション支援，
- 形式仕様で用いる数学的な概念

の観点から説明する．

ソフトウェアを開発するさいには，開発者の間で概念を

Vol.2019-ARC-240 No.43
Vol.2019-SLDM-191 No.43
Vol.2019-EMB-53 No.43
2019/2/28

共有することが最も大切である．概念を共有するためにはコミュニケーションが必要で，形式化は概念共有のための手段として用いられる．

本稿では，形式仕様とは「形式仕様言語で記述されたもの」と定義する．形式仕様言語とは，「仕様を記述するための構文と意味が厳密に定義された言語」である．プログラミング言語も形式仕様言語とみなす．形式仕様言語は，実行可能な言語とは限らない．実行可能な言語による記述は，計算モデルの構造に依存した記述になる．形式仕様言語には，計算モデルに依存しない抽象度の高い記述を可能にするものがある．

2.2.1 コミュニケーション支援

コミュニケーションの支援に対して形式仕様は，

- 形式化の道具と
- 前提の明示

としての役割を果たす．

共通概念を共有するためには，概念に名前をつけてその意味を共有する必要がある．形式化は，まさにこの「名前」と「意味」を定義するプロセスである．意味を記述する道具として自然言語を用いれば，意味に曖昧さが混入する恐れがある．意味を記述する道具として形式言語を用いると，言語上の曖昧さを排除することができる．

厳密な意味をもつ言語として「数学」の言語がある．形式仕様を記述するためには，数学の知識が必要である．この知識は，仕様を記述するための知識であり，自然言語の代替となるものである．概念の意味は，他の概念との関わりで表される．この関わりの表現は，記述のモデルに依存する．

形式仕様では，使用する概念に名前をつけるので，未定義の用語を用いて記述することができない．形式仕様を記述するプロセスでは，必ず前提を記述することになる．概念を名前として与えて，詳細な意味は後で定義することも，仕様記述のプロセスでは意義がある．

2.2.2 形式仕様で用いる数学的概念

形式仕様で用いる数学的な概念として，

- 述語論理
- 集合
- 時相論理
- 代数

がある．以下，それぞれのソフトウェア記述に関する役割を説明する．

述語論理

述語論理は，自然言語における文の構造の形式化である．述語論理は，自然言語の表現の代替表現である．述語論理では，対象および対象間に付随する概念に名前をつけ，その意味をより単純な概念を用いて「または」「かつ」「ならば」などを意味する演算子を用いて構成する．概念に名前をつけること（記号化）は，コミュニケーションのための抽象化の道具である．述語論理は，複雑な概念をより単純な概念を用いて表現することや，共通概念を記述するための道具である．

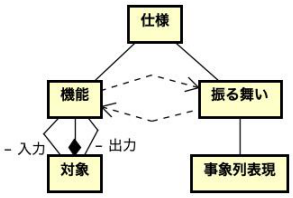

図 3 仕様に対する 2 つの視点

集合

集合は，集合とその集合に含まれるもの（要素）の関係の形式化である．集合と要素に名前をつけて，集合と要素の関係および集合間の関係を用いて概念を表現する．関係，関数，直積などは集合の概念で表現される．数学的対象は，集合の概念を用いて表現される．ソフトウェアの対象は，集合と述語論理を用いて形式的に表現される．集合や要素に名前をつけることが形式化の本質であり，述語論理や関係，関数などは表現の道具である．

時相論理

時相論理 [9] は，順番に並んだ列の性質の表現の形式化である．時相論理では，「事象 A の後に必ず事象 B がある」，「A の後は常に X という性質を満たす」などの表現を記号を用いて記述する．時系列に関する仕様は，時相論理を用いて記述することができる．

代数

代数は，等式による演算の形式化である．代数では，対象をソート（集合）とみなして，対象に付随する演算に名前をつけて等式を用いて演算の性質を表現する．代数は，ソートの要素の構造に言及しない抽象的な表現である．抽象データ型は，集合に対して演算を規定しその実現方法は自由度があることから，まさに代数仕様 [4] である．

3. 仕様の構造

対象に対する視点として「機能」と「振る舞い」がある．この 2 つの視点に着目して MVC に基づいた仕様の構造について説明する．

3.1 対象に対する 2 つの視点

仕様に対する 2 つの視点として，図 3 に示すように，「機能」と「振る舞い」があり互いに依存する関係がある．機能は，対象，入力，出力から構成され，入力と出力の関係を定義する．振る舞いは，機能を事象として抽象化して，対象を事象列として定義する．

機能の集合を定義すると，それぞれの機能が適用できる条件が決められているので，機能と事象を 1 対 1 で対応させると，実行可能な事象列が決まる．逆に，振る舞い（事象

Vol.2019-ARC-240 No.43
Vol.2019-SLDM-191 No.43
Vol.2019-EMB-53 No.43
2019/2/28

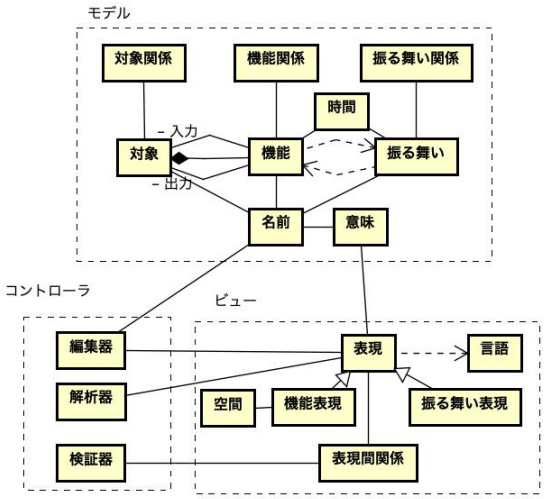

図 4　MVC に基づいた仕様構造の概要

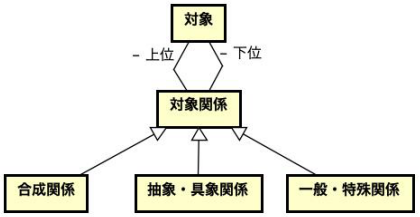

図 5　対象間の関係

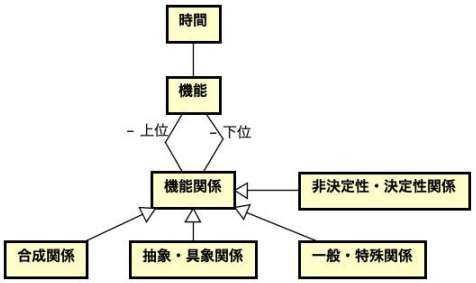

図 6　機能間の関係

列）の集合を定義すると，振る舞いと整合性をとるように機能の集合を定義する必要がある．機能と振る舞いの関係の整合性を保つように適切に表現することが，ソフトウェア開発で重要である．

3.2　MVC に基づいた仕様の構造

MVC アーキテクチャに基づいて仕様の構造を整理する．まず，仕様の構造の概要を説明した後で，モデル，ビュー，コントローラの詳細について説明する．

3.2.1　概要

MVC アーキテクチャに基づいた仕様構造の概要を図 4 に示す．モデルは，対象，機能，振る舞いで構成されていて，それぞれ名前が対応付いている．対象には複数の機能が付随している．時間は，機能と振る舞いに対する記述である．また，対象，機能，振る舞いには，それぞれ関係が存在する．具体的な関係は後述する．

ビューは，モデルの構成要素の意味を記述するための表現方法を提供する．表現は使用する言語に依存する．表現には 2 つの視点からの機能表現と振る舞い表現がある．対象，機能，振る舞いのそれぞれの間の関係を表現間関係とした．

コントローラは，編集器，解析器，検証器で構成されている．編集器は，モデルとビューに関連している．モデルへの編集は，名前および機能，対象，振る舞い間の関係に対して行われる．モデルの変更に伴い，対応する表現を更新する．ビューへの編集は，意味に変更がなく表現が変わる場合の編集である．解析器は，表現に対して，静的な意味解析および動的な意味解析を行う．静的な意味解析の例として型検査，動的な意味解析の例としてフロー制御に基づいた検査などがある．コンピュータを用いた実行（アニメーション）も動的解析とみなすことができる．

3.2.2　モデルの詳細

モデルの詳細として，対象，機能，振る舞いのそれぞれの関係について説明する．

対象間の関係

対象間の関係を図 5 に示す．対象間の関係には，合成関係，抽象・具象関係，一般・特殊関係がある．合成関係は，対象が複数の対象から構成されていることを表す．上位対象の構成要素が下位対象で構成されている．抽象・具象関係は，上位で表現されているデータ構造と下位の詳細なデータ構造との関係である．例えば，集合で表現されていた概念を配列で表現する場合の関係である．一般・特殊関係は，一般化された上位概念と特殊化された下位概念の関係である．これは，オブジェクト指向における is-a 関係に相当する．

機能間の関係

機能間の関係を図 6 に示す．合成関係は，上位の機能が下位の機能で構成されていることを表す．抽象・具象関係は，機能の入出力の型を表すシグネチャとその実現の関係である．一般・特殊関係は，入出力の型の一般化の関係である．上位の機能は，型変数を用いて表される．非決定性・決定性関係は，出力における非決定性と決定性の関係である．下位の機能では，上位の機能に比べてより決定的な出力が決まる．

振る舞い間の関係

振る舞い間の関係を図 7 に示す．合成関係は，振る舞いが複数の振る舞いから構成されていることを表す．抽象・具象関係は，事象の詳細化関係である．一般・特殊関係は，一般化した共通の振る舞いと特殊な振る舞いの関係であ

Vol.2019-ARC-240 No.43
Vol.2019-SLDM-191 No.43
Vol.2019-EMB-53 No.43
2019/2/28

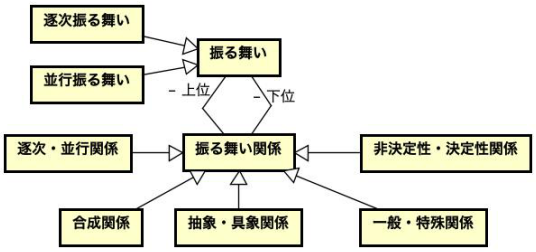

図 7　振る舞い間の関係

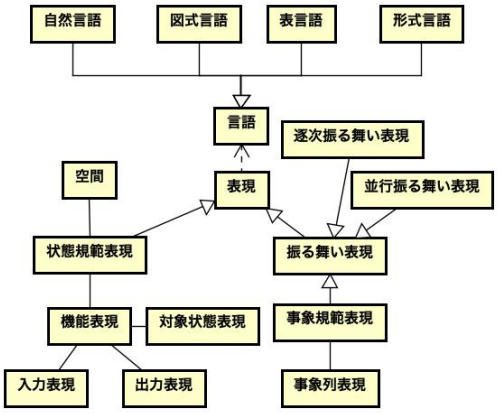

図 8　ビューの詳細構造

る．非決定性・決定性関係は，振る舞いにおける非決定性と決定性の関係である．下位の振る舞いは，上位の振る舞いに比べてより決定的な振る舞いになる．逐次・並行関係は，並行に発生する事象を逐次的に表現する関係である．

3.2.3　ビューの詳細

ビューの詳細構造を図 8 に示す．表現に用いる言語は，自然言語，図式言語，表言語，形式言語がある．表現の方法には，状態規範表現と振る舞い表現の 2 つがある．振る舞い表現は，言葉を変えると事象規範表現とみなすことができる．

状態規範表現

状態規範表現は，対象を状態として表現し，機能を，入力，出力，対象状態で表現する．手続き型プログラミング言語は状態規範表現である．

振る舞い表現

振る舞いは，事象列の集合として表現される．振る舞いには，事象が順番に起こるとみなす逐次振る舞い表現と，事象が並行に起こるとみなす並行振る舞い表現がある．

3.2.4　コントローラの詳細

コントローラの詳細構造を図 9 に示す．編集器は，モデルおよび表現の生成・消滅・変更の編集機能がある．解析器は，仕様の静的な意味に基づいて解析を行う静的解析器と，動的な意味に基づいて解析を行う動的解析器がある．検証器は，表現間の関係の正当性を検証する．下位の表現が上位の表現を満足することを検証する．検証する手段として，テスト，モデル検査，証明がある．

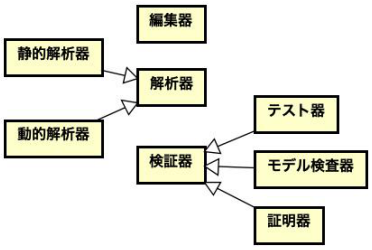

図 9　コントローラの構造

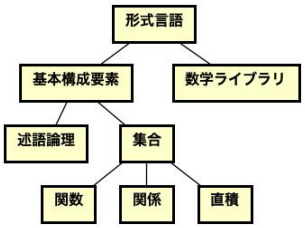

図 10　形式言語の構成要素

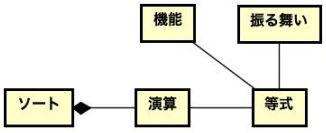

図 11　代数仕様

4.　形式仕様の構造

形式仕様の構造は，ビューの表現の構造で特徴付けられる．形式言語の構成要素を示し，代数仕様，状態規範型仕様，プロセス代数仕様について説明する．また，それぞれの仕様表現について他の仕様表現との間の関係を説明する．

4.1　形式言語の構成要素

形式言語の構成要素を図 10 に示す．形式言語は，基本構成要素と，基本構成要素を用いて構築される数学ライブラリがある．ソフトウェアの表現は，数学ライブラリの表現を用いて簡潔に表現できる可能性が高い．

基本構成要素は，述語論理と集合である．述語論理は，文の形式化の道具であり，集合は対象や機能を表現する基本道具である．直積，関係，関数は，集合の概念で捉えられる．

4.2　代数仕様

代数仕様の構造を図 11 に示す．代数仕様は，対象をソート（集合）とみなし，ソートに付随する演算間の関係を等式で表す．演算間の等式を用いて，機能の意味と振る舞いを規定する．ソートの要素の内部構造を規定しない抽象的な表現である．

Vol.2019-ARC-240 No.43
Vol.2019-SLDM-191 No.43
Vol.2019-EMB-53 No.43
2019/2/28

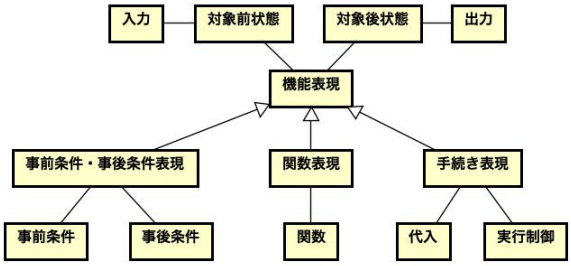

図 12　状態規範型仕様

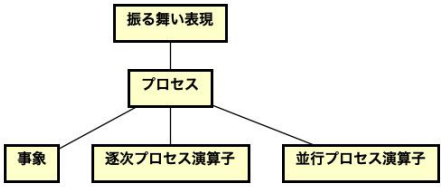

図 13　プロセス代数仕様

4.3　状態規範型仕様

状態規範型仕様の構造を図 12 に示す．機能は，対象の前状態，入力，対象の後状態，出力で構成される．機能の表現方法には，事前条件・事後条件表現，関数表現，手続き表現がある．

事前条件・事後条件表現

事前条件・事後条件表現では，事前条件と事後条件の視点から機能の意味を表現する．この表現は，入力から出力を計算するアルゴリズムに依存しない抽象的な表現で与えられる．事前条件と事後条件の関係を代入を用いずに，述語論理と集合を用いて表現する．事前条件・事後条件表現は，プログラム表現の上位表現としての役割を果たす．

関数表現

関数表現では，入力と対象前状態の組を定義域，出力と対象後状態の組を値域とした関数で，機能の意味を表現する．この表現は，事前条件・事後条件表現を関数で表現するとみなすことができる．関数表現は，手続き表現と同様にコンピュータによる実行が可能である．状態の変化を代入を用いずに関数で表現し，手続き表現の繰り返し表現を再帰を用いて表現する．

手続き表現

手続き表現では，状態の変化を代入と，条件，繰り返し，逐次合成を用いた実行制御で表現する．対象前状態は手続きを実行する直前の状態，対象後状態は手続きが終了した直後の状態に対応する．手続き型のプログラミング言語と形式仕様言語の違いは，形式仕様言語では，前述の形式言語の基本構成要素が言語のプリミティブとして用意されてることである．

4.4　プロセス代数仕様

プロセス代数の構造を図 13 に示す．プロセス代数では，機能を事象として抽象化して，事象列を逐次プロセス演算子と並行プロセス演算子で表現する．並行振る舞いは，インターリーブの概念を用いて逐次振る舞いとして表現される．

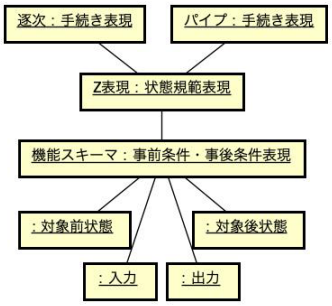

図 14　Z 言語を用いた仕様表現の構造

4.5　形式仕様間の関係

上述のそれぞれの仕様表現について，他の仕様表現との間の関係について述べる．

代数仕様

代数仕様では等式を用いて演算の意味を表現するが，演算の意味を定義するために状態の概念が必要な場合がある．例えば，バッファの仕様を表現するためには，バッファの状態を記述する必要がある．代数仕様では，状態を演算の添字として表現する．演算名に列の概念を用いてバッファとして保持すべき情報を表現する．このように，代数仕様でも状態の概念は存在する．

状態規範型仕様

状態規範型仕様では，機能の表現で，振る舞いの表現が記述できる．プロセス代数で表現される事象列は，状態遷移モデルで表現することができる．状態遷移モデルは，状態規範型表現で記述することができる．

プロセス代数仕様

プロセス代数仕様は，事象列を用いて仕様を表現するが，状態の概念が必要な場合がある．プロセス代数では，代数仕様と同様に，状態をプロセスの添字として表現する．プロセス代数でも状態の概念は存在する．

5.　既存の形式仕様言語に対する考察

既存の状態規範型の形式仕様言語である Z 言語 [12] と VDM-SL 言語 [2], [5] に対して，形式仕様メタモデルを適用する．形式仕様言語で表現される仕様の構造は，メタモデルから生成されるオブジェクトとして表される．Z 言語を用いた仕様表現の構造を図 14，VDM-SL 言語を用いた仕様表現の構造を図 15 に示す．

Z 言語では，機能の記述を事前条件・事後条件表現による機能スキーマを用いて表現する．機能スキーマは，入力，対象前状態，対象後状態，出力から構成され，これらの関

Vol.2019-ARC-240 No.43
Vol.2019-SLDM-191 No.43
Vol.2019-EMB-53 No.43
2019/2/28

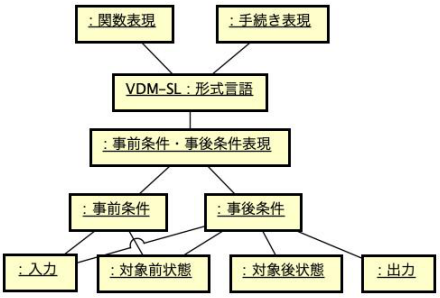

図 15 VDM-SL 言語を用いた仕様表現の構造

係を集合と述語論理を用いて記述する．Z 言語では，手続き表現として逐次とパイプを用いた記述が可能である．

VDM-SL 言語は，事前条件・事後条件表現，関数表現，手続き表現の 3 つの表現方法を提供する形式仕様言語である．仕様記述者は，目的に応じて適切な表現方法を選択することができる．Z 言語と同様に，事前条件・事後条件を用いた記述が可能である．Z 言語と VDM-SL 言語の違いは，後者は事前条件と事後条件が分かれていることである．前者は，事前条件と事後条件が分かれていないので，より簡潔な記述が可能である．また，VDM-SL 言語は，関数プログラミング言語や手続き型プログラミング言語と同様の概念を含んでいる．形式仕様言語とプログラミング言語の違いは，集合に基づいた数学ライブラリを利用することが可能なことにある．

6. おわりに

本稿では，形式仕様に基づいた形式仕様記述を支援するために，MVC アーキテクチャに基づいた形式仕様メタモデルの構築を試みた．仕様の構造を，機能と振る舞いの 2 つの視点に着目して，仕様の表現に依存しない仕様に共通な構造（モデル），仕様の表現（ビュー），仕様記述のプロセス（コントローラ）に分離して，それらの構造と関係を示した．

仕様を記述する視点として，機能に着目した状態規範型表現と，振る舞いに着目した事象規範表現がある．状態規範表現には，事前条件・事後条件表現，関数表現，手続き表現がある．状態規範型表現と事象規範表現の間には，相互に依存関係がある．これらの両方がある表現方法も存在する．

プログラミング言語 Ada[1] は，形式仕様の表現として SPARK 言語 [10], [11] を正式に取り入れている．プログラムに対して，事前条件・事後条件表現を与えることにより，SMT ソルバーを用いたプログラムの正当性検証の自動化を支援している．B-Method[8] では，証明による段階的詳細化の検証環境を整備している．今後，形式仕様の必要性が高まることが予想される．

形式表現と非形式表現を融合した仕様表現を構築するためには，提案した MVC によるメタモデルではプリミティブの構成として不十分である．今後は，アーキテクチャの観点として，国際標準である ISO/IEC/IEEE42010[6]，Kruchten の「4+1」ビューモデル [7]，Clements らのアーキテクチャの枠組み [3] を検討して，形式仕様のメタモデルの構築を目指す予定である．

謝辞 本研究の一部は，JSPS 科研費 19K11911，2019 年度南山大学パッヘ研究奨励金 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] 荒木啓二郎，張漢明：プログラム仕様記述論，オーム社 (2002).
[3] Clements, P., Bachmann, F., Bass, L., Gar- lan, D., Ivers, J., Little, R., Merson, P., Nord, R. and Stafford, J.: Documenting Software Architectures: Views and Beyond, Second Edition, Addison-Wesley (2010).
[4] Enrig, H. and Mahr, B.: Fundamentals of Algebraic Specification 1, Springer-Berlag (1985).
[5] Fitzgerald, J. and Larse, P.G.: Modelling Systems, Cambridge, (2009).
[6] ISO/IEC/IEEE 42010: Systems and Software Engineering - Architecture Description, 2011, http://www.iso-architecture.org/42010/.
[7] Kruchten, P.: Architectural Blueprints - The “4+1” View Model of Software Architecture, IEEE Software, Vol. 12, No. 6, pp. 42-50 (1995).
[8] 来馬啓伸：B メソッドによる形式仕様記述，近代科学社 (2007).
[9] Manna, Z. and Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems - Specification- , Springer-Verlag (1992).
[10] SPARK Team: SPARK 2014 Reference Manual, AdaCore, http://docs.adacore.com/spark2014-docs/html/lrm/.
[11] SPARK Team: SPARK 2014 Toolset User’s Guide, AdaCore, http://docs.adacore.com/spark2014-docs/html/ug/.
[12] Spivey, J.M.: The Z Notation, Prentice Hall (1992), http://spivey.oriel.ox.ac.uk/ mike/zrm/zrm.pdf.