ソフトウェア工学 83－13
（１９９２．２．７）

集合に基づく形式的言語を使ったソフトウェア仕様の記述形態について

来間 啓伸　　大槻 繁

（株）日立製作所　システム開発研究所

ソフトウェアの開発に関わる人々は，仕様を通じて意図の明確化と伝達を行なう．ところが，仕様の記述形態は，対象のどのような性質に注目し，それをどのように表現するかによって異なり，記述者固有の暗黙の前提に依存する．ここでは，対象を把握するための構成概念として「情報」「ふるまい」「個体」の３つの概念をとりあげ，これらの視点から，形式的言語を使った仕様の記述形態を特徴付ける．記述言語にはＺを用い，その言語要素を構成概念の点からとらえ直す．単純化した仕様化概念の下に記述形態を位置付けることで，仕様の結合，組み替え技術を構築する基盤を与えることができる．

A Classification of Software Specification Styles for Set Based Language

Hironobu Kuruma, Shigeru Otsuki

Systems Development Laboratory, Hitachi, Ltd.
1099 Ohzenji Asao-ku Kawasaki-shi, 215 Japan

In the process of software development, developers at each stage communicate with each other by specifications. Today, specification languages with rigidly defined syntax and semantics are proposed to establish precise communication. However, one's specification style depends on the properties he notices and the way to represent them. In this paper, we classify the specification styles on the basis of three components, "information", "behaviour" and "individual". Three specification examples corresponding to each component are shown in Z, a formal language based on set theory and first-order logic. By classifying specification styles under these simple concepts, the basic framework for constructing and restructuring specifications could be provided.

1．はじめに

ソフトウェアを開発する際に，開発しつつあるシステムを明確にし，他の部分と整合していることを確認する手段として，仕様は重要な役割を果たす．ところが，同一の対象を同一の言語を使って仕様化しても，対象のどのような性質に注目し，それをどのように表現するかによって，仕様の記述形態は異なる．

形式的言語を使った仕様記述に関してもこれは同様であり，構文規則と意味規則が厳密に定まった記述言語を用いても，記述形態までもが一意に定まるわけではない．記述形態の混乱は，仕様の理解性を妨げるだけでなく，仕様の結合や組み替えを困難にし，仕様の一般性を低下させるが，各種の言語提案はなされている半面，その記述方法に関する提案はあまりなされていない．

本稿では，人間が対象を把握する際の認識単位として「情報」「ふるまい」「個体」の３つの概念をとりあげ，これらの視点から，仕様の記述形態を特徴付ける．各概念に関連する記述形態の比較を，形式的仕様の上で行なう．記述言語には形式言語Ｚを用いた．

２節では仕様記述モデルについて述べ，単純化した仕様記述モデルから構成概念を導出する．３節では，文献［１，６，７］に基づいて，形式仕様記述言語Ｚの構文と意味を簡単に紹介する．４節では，Ｚの言語要素に２節の構成概念に従った認識的意味を与え，それぞれに異なった表記を与える．５節では，簡単な問題について，各構成概念に関連する記述形態を，４節の表記を使って例示する．

2．仕様化概念

2．1　仕様記述モデル

ソフトウェアの開発は，計算機によって解決したい問題を抽出し，それを解決するために計算機が提供しなければならない機能を明確にし，機能を実現する方法をプログラムに表現する過程である．多くの場合，問題は漠然としており，機能は抽象的である．これに対し，プログラムは計算機の動作を具体的に指示するものでなければならない．この意味では，プログラム開発は，対象世界の中で意味を持つ漠然とした意図を，計算機の制約された形式の下で具体的に表現する過程であると言うことができる．

ソフトウェアに対する要求，ソフトウェアが提供すべき機能，それを実現するための方法などは，仕様の記述を通じて明確にされてゆく．これらの各段階に関与する人々は，仕様を解釈するための概念を共有し，その下で仕様を記述，理解することにより，意図の表現の形式化と意図の伝達を行なうことができる．仕様に解釈を与える共通概念を，ここでは仕様記述モデルと呼ぶことにする．

この意味で，ソフトウェア開発技法やプログラミング技法など，ソフトウェアを対象との関係において把握するための技法は，仕様記述モデルを与える．仕様記述モデルは，人間の対象認識と自然な対応をなしていることが求められる．

2．2　構成概念

ソフトウェアは，それが動作する対象世界において認識され，意味づけられる．仕様記述モデルは，ソフトウェアとそれを取り巻く対象世界の関係のとらえ方を表現するモデルである．そこで，極めて単純なモデルに基づき，それを構成する基本概念を考える．

計算機システムを，図２．１のように，システム外部の世界から情報を受け取り，操作し，システム外部の世界に出力するものと捉える。

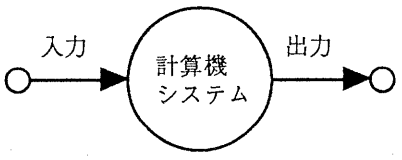


図２．１　計算機システムの認識モデル

このモデルから，人間の対象世界の認識単位として「情報」「ふるまい」「個体」の３つの概念をとりあげる．

2．2．1　情報概念

計算機システムの操作対象を規定する概念を，情報概念とする．すなわち，情報概念は，時間的要素を含まない，「データ」や「関数」などを規定する．

計算機システムにおいて，ある時点で直接あるいは間接に観測することのできる値の集合を状態とすると，情報概念はシステムの取り得る状態空間を規定する．

2．2．2　ふるまい概念

情報の入出力とその操作には，時間的順序関係が伴う．ふるまい概念は，情報の操作順序を規定する概念であり，計算機システムの時間的な変化を表現する．

情報概念が状態空間を規定するのに対し，ふるまい概念は状態遷移を規定する．

2．2．3　個体概念

個体概念は，計算機システムの境界およびその構成要素の境界を規定する概念である．したがって，「オブジェクト」や「エンティティ」などは個体概念によって規定される．

情報概念が状態空間を，ふるまい概念が情報遷移を規定するのに対し，個体概念は状態空間の部分空間への分割を規定する．

3．形式仕様記述言語Z

3．1　Zの基本概念

Zは一階の論理と集合論に基づく仕様記述言語であり，ソフトウェアの仕様化と設計を数学的正確さの下で行なうための記法の提供を目的とする［1，6，7］．

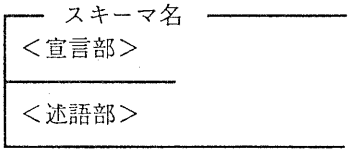
Zでは，システムが取り得る状態の状態空間の構造と，対象が起こし得る状態遷移を与えることによって，仕様を記述する．ここで，状態は値の組として表現され，状態空間の構造は，値の間の制約として与えられる．この制約は，恒等関係（invariant relation）と呼ばれる．状態遷移は，遷移前の値の組と遷移後の値の組を示すことによって表現される．Zは，状態空間の記述に集合を用い，制約の記述に一階論理を用いる．

3．2　スキーマ

Zでは，スキーマと呼ばれる言語要素を組み合わせて仕様を表現する．図3.1は，スキーマの図式表現の構文規則の概略を示す．ここで，変数宣言は変数名とそのデータ型の宣言である．変数宣言は，入出力に関係する変数の宣言を含み，入力変数は変数名に？を後置，出力変数は変数名に！を後置して表わす．述語は，宣言部に宣言された変数に対する制約を規定する．また，宣言部では他のスキーマを参照することができ，参照するスキーマの値を継承する．

システムの静的，動的側面は，ともにスキーマによって表現される．スキーマが表現する静的側

＜スキーマ＞：：＝



＜宣言部＞：：＝

　　＜宣言＞；　…　；＜宣言＞

＜宣言＞：：＝

　　＜変数宣言＞　または
　　＜スキーマ参照＞

＜変数宣言＞：：＝

　　変数名：型

＜スキーマ参照＞：：＝

　　スキーマ名

＜述語部＞：：＝

　　述語；　…　；述語

図3．1　スキーマの構文

面は，システムが占めることのできる状態空間の部分空間と，システムの状態が変わる時に保持される恒等関係であり，動的側面は，可能な演算と，入出力の関係，状態の変化である．静的側面を表現するスキーマでは，述語は各変数間の恒等関係を表わす．この恒等関係は状態遷移の前後でも不変に保たれ，データの性質を表わすものとも言える．動的側面を表現するスキーマでは，状態遷移を，宣言部に遷移前のスキーマと遷移後のスキーマを並べて入れ子にすることによって表現する．述語は，遷移が起こるための前提条件と，遷移後に各変数が持つ値を示す．

3．3　仕様記述の例

図3．2，図3．3は，簡単な住所データベースの仕様記述の例の一部である．このデータベースは，人の名前と，各々の名前に対応する住所を格納する．データベースに対する演算は，新しい人に対する住所の登録である．ここで，登録される可能性のある名前の集合NAMEと，登録される可能性のある住所の集合ADDRESSはあらかじめ与えられているものとする．

図3．2のスキーマAddressDBは，このデータベースシステムの状態空間の構造を表わす．状態空間は，変数personとrecordによって規定される．ここで，personはNAMEの部分集合であり，recordはNAMEからADDRESSへの関数の部分集合である．恒等関係は，関数recordの定義域がpersonと一致することである．

図3．3のスキーマAddPersonは，新しい人と住所の登録による状態遷移を表わす．この演算により，AddressDBの状態は，AddressDB'の状態に変わる．personとrecordは遷移前の状態変数を，

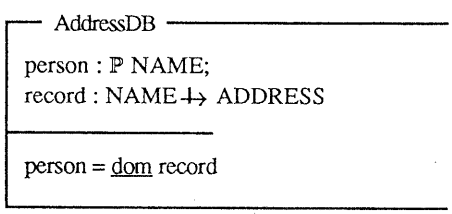


図3．2　状態空間を表わすスキーマ

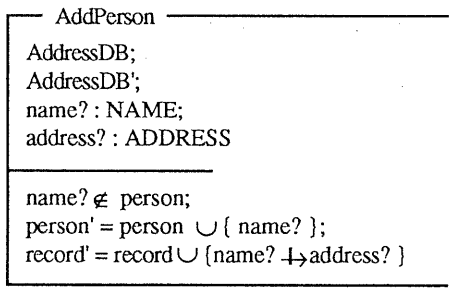


図3．3　状態遷移を表わすスキーマ

person'とrecord'は遷移後の状態変数を表わす．演算の入力は，新しい人の名前name?とその住所address?である．述語部には，この遷移が起こるための前提条件と，遷移前後の状態変数の値の関係を与える．ここでは，新しい人の名前が既に登録されたものでないことを前提条件とし，遷移後は記録が付け加えられることを表わす．

4．記述の制約

4．1　スキーマの分類

Zは，システムが取り得る状態の状態空間の構造と，対象が起こし得る状態遷移を与えることによって，仕様を記述する．Zでは，このようなシステムの静的側面も動的側面も，ともにスキーマを使って表現する．

しかし，記述者がシステムをどう認識し表現したかを明確にするという点からは，各々のスキーマの構文要素と意味要素を制約し，これらを区別するほうが望ましいと思われる．ここでは，システムの静的側面を表わすスキーマをエンティティ，動的側面を表わすスキーマをイベントと呼び，それぞれに異なった表記を与える．イベントはエンティティ間の相互作用を表わす．以下では，イベントとエンティティの構文と意味および，2節の構成概念にしたがった認識的意味について述べる．

4．2　エンティティの構文

エンティティは，状態空間の部分空間の構造を

表わす．2節の構成概念との対応では，エンティティは情報概念と個体概念によって規定されることになる．

図4．1は，エンティティの構文規則である．スキーマの構文をほぼ受け継ぐが，述語部は恒等関係とし，時間的変化の記述は許さないこととする．言い換えれば，エンティティが表現するのは，広い意味での「データ型」のみに制限する．

4．3　イベントの構文

イベントは，状態遷移を表わす．2節の構成概念との対応では，イベントはふるまい概念によって規定されることになる．

図4．2は，イベントの構文規則である．イベントは，イベント名と遷移式またはイベント式の組によって記述する．遷移式は，遷移前の状態と遷移後の状態の組である．イベント式はイベントの積または和であり，積はそれらのイベントの同時発生を，和はそれらのイベントの少なくとも1つの発生を表わす．

なお，スキーマとの対応では，状態遷移を起こすエンティティの宣言が必要であるが，本稿では省略する．

<エンティティ>：：=

エンティティ名
<宣言部>
<恒等関係>

<宣言部>：：=
　<宣言>；　…　；<宣言>

<宣言>：：=
　<状態変数宣言>
　または
　<エンティティ参照>

<状態変数宣言>：：=
　変数名：型

<エンティティ参照>：：=
　エンティティ名

<恒等関係>：：=
　述語；　…　；述語

図4．1　エンティティの構文

<イベント>：：=
　イベント名==<イベント本体>

<イベント本体>：：=
　<遷移式>
　または
　<イベント式>

<遷移式>：：=
　<プリコンディション>
　ー>　<ポストコンディション>

<プリコンディション>：：=
　述語，　…　，述語

<ポストコンディション>：：=
　述語，　…　，述語

<イベント式>：：=
　<イベント式> and <イベント式aux>
　または
　<イベント式> or <イベント式aux>
　または
　<イベント式aux>

<イベント式aux>：：=
　イベント名
　または
　（<イベント式>）

図4．2　イベントの構文

5．仕様記述形態

5．1　多様性の要因

構成概念レベルに分解した場合，記述形態の多様性の大きな要因として以下の点を挙げることができる．

（1）個体の任意性．

（2）情報とふるまいの二重性.

前者は，対象世界のどのような性質を個体と見なしてカプセル化するか，また，どのような性質を個体間の相互作用と見なすか，に関連する選択の任意性であり，後者は，どのような性質を静的な情報として記述するか，どのような性質を動的なふるまいとして記述するか，に関連する任意性である．これらの任意性がソフトウェアの開発過程で取捨選択され，一つの，あるいはレベルが異なる複数の，記述形態へと収束することになる．

以下では，同じ問題を３つの形態で記述した仕様を例示する．問題として，自動車のワイパーシステムの仕様化を考える．このワイパーシステムは，以下のような非常に簡単な機能を持つ．

（1）オン／オフ型の切り替えスイッチによって，ワイパーの動作／停止を行なう．

（2）ボタン型のスイッチを押すとウォッシャーとワイパーが動作する．

（3）ボタン型のスイッチを放すと，ウオッシャーが停止する．

（4）ボタン型のスイッチを放したとき，切り替えスイッチがオフになっていればワイパーも停止する．

```
WiperSystem
switch-state : {on, off};
button-state : {on, off};
wiper-state : {on, off};
washer-state : {on, off}
-----------------------------
(switch-state=off ∧ button-state=off
∧ wiper-state=off ∧ washer-state=off) ∨
(switch-state=on ∧ button-state=off
∧ wiper-state=on ∧ washer-state=off) ∨
(switch-state=off ∧ button-state=on
∧ wiper-state=on ∧ washer-state=on) ∨
(switch-state=on ∧ button-state=on
∧ wiper-state=on ∧ washer-state=on)
```

図５．１　情報主導型の仕様

５．２　情報主導型

情報主導型の記述スタイルでは，システムが取り得る状態を規定し，状態から状態への遷移や，それを引き起こすイベントの時間的順序関係は規定しない．また，システムへの入出力にも直接には関わらない．これらは，暗黙のうちに示されることになる．

図５．１は，ワイパーシステムを情報主導スタイルで記述した例である．

ここでは，状態変数の値に対する制約によって状態を表現している．状態変数は，switch-state, button-state, wiper-state, washer-stateの４つで，各々の変数はonかoffのいずれかの値を取る．このような変数の組み合わせは2^4通りあるが，恒等関係により，４つの組み合わせのみが許される．すなわち，WiperSystemは，４つの状態を取り得る．

５．３　ふるまい主導型

```
WiperSystem
state : { init, wiping,
          washig-while-wiping, washing }
```

```
SwitchOn.Wipe ==
   state = init  ->  state = wiping;
SwitchOff.Stay ==
   state = wiping  ->  state = init;
ButtonOn.Wash.Wipe ==
   state = init  ->  state = washing;
ButtonOff.Stop.Stay ==
   state = washing  ->  state = init;
ButtonOn.Wash ==
   state = wiping
   ->  state = washing-while-wiping;
ButtonOff.Stop ==
   state = washing-while-wiping
   ->  state = wiping;
SwitchOn ==
   state = washing
   ->  state = washing-while-wiping;
SwitchOff ==
   state = washing-while-wiping
   ->  state = washing
```

図５．２　ふるまい主導型の仕様

ふるまい主導の記述スタイルでは，システムに状態変化を引き起こすイベントと，その時間順序を中心に，システムを規定する．

図５．２はワイパーシステムをふるまい主導スタイルで記述した例である．ワイパーシステムは，それを構成する各部分の動作と，その時間順序によって特徴付けられる．ここでは，各動作を次のように表わした．

・スイッチのオン　SwitchOn
・スイッチのオフ　SwitchOff
・ボタンのオン　ButtonOn
・ボタンのオフ　ButtonOff
・ワイパーの動作開始　Wipe
・ワイパーの動作停止　Stay
・ウォッシャーの動作開始　Wash
・ウォッシャーの動作停止　Stop

これらの動作は，イベントを構成する．ここで，複数の動作が同時に起こると見なしうる場合，それらをまとめて一つのイベントとし，関連する動作を"."でつないだものをイベント名とした．例えば，SwitchOn.Wipeはスイッチのオンとワイパーの動作開始が同時に起こるイベントである．

イベントの発生は，それ以前に起こったイベントの順序に依存する．ここでは，このようなイベントの発生条件を，イベントの順序を陽に示すのではなく，状態変数stateを用いて表わしている．

５．４　個体主導型

個体主導の記述スタイルでは，システムを構成

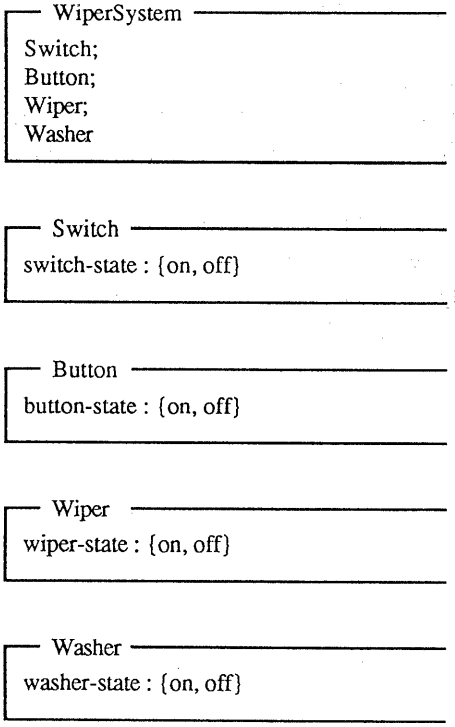


図５．３．１　個体主導型の仕様（１）

```
SwitchOn ==
   switch-state = off
   -> switch-state = on;
SwitchOff ==
   switch-state = on
   -> switch-state = off;

ButtonOn ==
   button-state = off
   -> button-state = on;
ButtonOff ==
   button-state = on
   -> button-state = off;

Wash ==
   button-state = on, washer-state = off
   -> washer-state = on;
Stop ==
   button-state = off, washer-state = on
   -> washer-state = off;

Wipe1 ==
   switch-state = on, wiper-state = off
   -> wiper-state = on;
Stay1 ==
   switch-state = off, wiper-state = on
   -> wiper-state = off;

Wipe2 ==
   button-state = on, wiper-state = off
   -> wiper-state = on;
Stay2 ==
   button-state = off, wiper-state = on
   -> wiper-state = off;

Wipe == wipe1 or Wipe2;
Stay == Stay1 and Stay2
```

図５．３．２　個体主導型の仕様（２）

する個体と，個体間の相互作用を中心にシステムを表現する．各々の個体は，情報主導型，ふるまい主導型スタイルによって記述される．あるいは，個体主導スタイルによってさらに分割される．この結果，個体主導スタイルでは，情報主導スタイルやふるまい主導スタイルの特徴が混在し得る．このスタイルでは，個体の切り分け方と相互作用の表現が仕様を特徴付ける．

図５．３．１と図５．３．２は，ワイパーシステムを個体主導スタイルで記述した例である．ここでは，物理的対象であり，システムの特性に本質的影響を及ぼすと思われる，スイッチ，ボタン，ワイパー，ウォッシャーを個体としている．すなわち，WiperSystemは，Switch，Button，Wiper，Washerから構成される．各々の個体は状態空間の部分空間を構成し，イベントの発生によって状態遷移を起こす．５．３項の記述とは異なり，ここでは各動作にイベントを割り付けている．ただし，Wipe1とStay1はスイッチのオン／オフによるワイパーの動作開始と動作停止，Wipe2とStay2はボタンのオン／オフによるワイパーの動作開始と動作停止イベントである．イベントWipeは，Wipe1とWipe2のプリコンディションのいずれかが満たされたときに発生し，イベントStayは，Stay1とStay2のプリコンディションの両方が満たされたときに発生する．

６．おわりに

形式的仕様は，構文規則と意味規則が厳密に定義された記述言語と，仕様を操作する形式的な方法の下に，記述者固有の暗黙の前提をできるだけ排除した，より客観的な意味を持つ仕様である．しかし，個々の問題について見れば，形式的仕様でも記述形態は様々であり，仕様間の結合や組み替えは必ずしも容易ではない．

本稿では，仕様の記述形態を対象を把握する抽象化概念に基づいて特徴付け，各々の概念に関連する仕様記述形態を例示した．また，記述言語に形式言語Ｚを用いて，各形態を同じ枠組みの下で表現することにより，それらの相関を探った．

ここで述べた分類はまだ不十分なものであるが，単純化した仕様化概念に基づいて記述形態を位置付けることにより，仕様の結合，組み替えを行なう一定の基盤を提供できるものと考える．特に個体概念は，ソフトウェア開発過程における記述形態の転換に関して，重要な役割を果たすものと思われる．個体概念によって規定される性質と仕様化モデルとの関連づけは，今後の課題である．

最後に，本研究の機会を与えていただいた(株)日立製作所システム開発研究所の堂免信義所長に感謝いたします．

参考文献

[１]Diller, A. , Z - An Introduction to Formal Methods, John Wiley & Sons Ltd. (1990)

[２]二木，ＩＳＯにおける形式記述技法の標準化動向，情報処理，31，1，pp. 3-10 (1990)

[３]Milner, R. , Communication and Concurrency, Prentice Hall (1989)

[４]Narayana, K. T. and S. Dharap, Invariant Properties in a Dialog System, Proceedings of the ACM SIGSOFT Internal Workshop on Formal Methods in Software Development, pp. 67-79 (1990)

[５]大槻，ソフトウェア仕様記述モデルのための形態学，情報処理学会研究報告，71，2，pp. 9-16 (1990)

[６]Spivey, J. M. , Understanding Z, Cambridge University Press (1988)

[７]Spivey, J. M. , The Z Notation - A Reference Manual, Prentice Hall (1989)