ソフトウェア工学　81－3
（1991．10．4）

CASE指向抽象ソフトウェアモデルIII
― オブジェクト指向との結び目 ―

松本憲幸　岡山敬　湯原義彦

(株)東芝

プログラミング言語により具現化される以前の抽象段階において、ソフトウェアの構造や振る舞いを統一的に記述することを想定した記号モデルについて説明する。このモデルは、ソフトウェア定義に関する人の思考過程とコンピュータ処理を媒介することを想定したもので、このモデル化によってソフトウェア工学の定義範囲を越える仕様記述を回避すると同時に、数学的なモデルとの対応の明確化およびコンピュータによる静的・動的解析の強化を狙っている。

本論文では、リアルタイム構造化分析情報に基づく従来のモデルを現実的なシステムの設計に適用する場合のいくつかの問題点を解決するためのオブジェクト指向的な概念の導入と、改造されたモデルとオブジェクト指向分析手法との対応関係について説明する。

Abstract Software Model for CASE (III)
― Object Oriented Extension ―

Noriyoshi Matsumoto　Takashi Okayama　Yoshihiko Yuhara

TOSHIBA Corporation

1 Toshiba-cho, Fuchu-Shi, Tokyo 183, Japan

An abstract software model is introduced to describe the primary definition of the software in Upper CASE domain. The model of our early work synthesized the specifications of the software in the framework of the real-time structured analysis and described the information in the framework of the finite automata theory. But it was hard to apply the model to design the large system, for it lacks the rule for the scope of the definition and the classification method, both of them are the specific method of the object-oriented methodology.

In this paper the extension of our old symbolic model to involve the object-oriented specifications of the software and the association with the object-oriented software design methodology are presented.

1. はじめに

プログラムとして実体化される以前の抽象段階におけるソフトウェアの定義過程をコンピュータサポートするための基礎研究として、設計方法論に従った各種設計図によって定義されるソフトウェア情報を統合したモデルの検討を行っている[1],[2]。

このモデルは、ソフトウェア定義に関する人の思考過程とコンピュータ処理を媒介することを想定したもので、このモデル化によってソフトウェア工学の定義範囲を越える仕様記述を回避すると同時に、数学的なモデルとの対応の明確化および定義域を限定した強力なコンピュータサポートを狙っている。

モデル化は、各種ソフトウェア設計方法論によって決定される情報とオートマトン定義とを対応付けることから始め、既にリアルタイム構造化分析手法[3]、ジャクソン法およびワーニエ-オー法により決定される定義情報との対応を可能としている。

本論文では、従来のモデルを現実的なシステムの設計に適用する場合のいくつかの問題点を解決するためのオブジェクト指向的な概念の導入と、改造されたモデルとOOA(オブジェクト指向分析)手法[4]との対応関係について説明する。

2. 構造化モデル適用上の問題点

(1) 状態遷移過程の構造化の問題

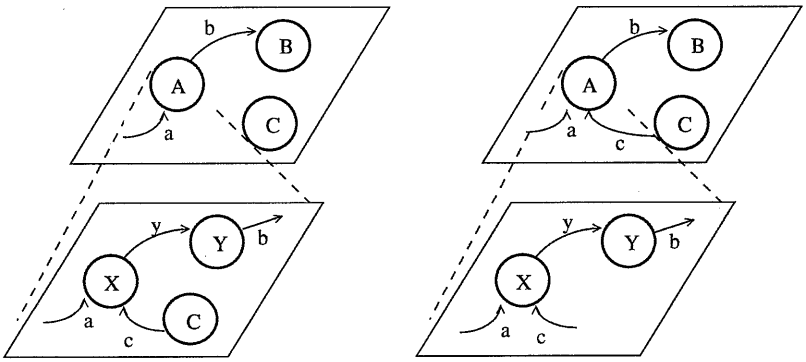
従来のモデルは、非決定的有限オートマトン(NDFA)定義に相当する記述形式によって、リアルタイム構造化分析情報を統合している。NDFAは、自らの内部状態変化によって要求仕様を実現するような1つの平坦な(階層化されない)マシンを表現する。これに対して、DeMarcoの手法[5]に従って処理過程を構造化した我々のモデルは「異なる特性をもったNDFAの組み合わせによって記述される別のNDFA」を定義している。この拡張は、次の問題を含む。

- NDFA定義に対して、詳細化により追加される構造化定義が与える影響。
- 構造化された遷移過程実行中のマシンの状態の解釈。

(2) スコープの問題

従来のモデルでは、設計要素定義の有効範囲(スコープ)の規約がなく、記述された要素がシステムの内部要素か外部要素か判別できない。

これは、図1に示される不都合を生じる。例(1-a)において、ある階層で要素Aと外場との



(1-a) スコープ規則なし　　(1-b) スコープ規則あり

例1. スコープを持たない構造化とスコープを設定した構造化

関係を記述し、Aの構造化定義の中で異なった要素Cとの関係を記述した場合、CがAの内部構造に属する要素でないとすると、上位のレベルで記述された関係と食い違い、仕様上の矛盾を生じる。論理的な因果関係を正確に記述しながら構造化を行った場合はこのような事態は有り得ないが、上位の要求仕様が必要最低限度しか記述されずこれを詳細化して行くような解析パターンではこのような事態が発生する。このような矛盾を含む場合、ある階層で記述した要素間の相互関係が情報の因果関係の意味を成さないことになる。

(3) 類似要素の設計効率の問題

従来のモデルは、個々の設計要素ごとに定義を行う形式をとっており、人が大規模の情報を取り扱う場合に行う情報の分類の手段を与えていない。このため、大規模なシステムを記述する場合に、多くの要素の特性をすべてバラバラに定義/理解することになり、各要素間の類似性が読み取れないことや、類似した要素を設計する場合の記述効率が悪いなどの問題をもつ。

3. モデル改造の基本概念

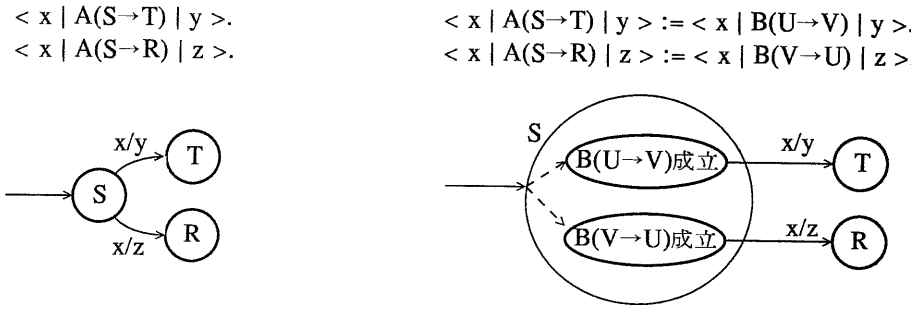
ここでは、従来のモデルに対する修正として、構造化定義の解釈、スコープの設定およびクラス/インスタンス定義の導入について説明する。

3.1 構造化定義の解釈

構造化された処理(例2-b)の左辺(NDFA仕様)と右辺(構造化された過程)の本質的な違いは、抽象度の違いである。通常NDFAによるソフトウェア定義は、階層構造なしで厳密な定義を1平面上に表現する。一方、設計方法論に従って構造化された定義は、初期段階の「構造化されないNDFA定義」がシステムの微細な構造や状態に関する記述を無視している点で完全ではない可能性を示唆し、右辺の詳細な構造定義は左辺を修飾する仕様として位置付けられる。

構造化された右辺が左辺を修飾する「システムの微細構造に関する補足仕様」であるとすると、両辺の仕様が充足される場合にのみこの遷移過程が起こることになる。ところが、構造化された右辺の仕様は動的な状態変化を伴うため、その過程を実行することなく仕様の成否を確認することができない。このため、右辺により記述される仕様の修飾は、右辺の仕様を動的に確認できた時点で左辺仕様の遷移が起こるような後ろ向き的な修飾となる。

この解釈の下では、構造化と遷移過程定義との対応関係は例2のようになる。(2-b)において、破線の状態遷移が左辺の定義からは判定できず、右辺による動的な判定となる部分である。この現象は、左辺で記述される遷移定義が不完全なために起こるもので、決定論的に解釈すれば右辺の成否を支配する条件が左辺に記述されていないことに起因する不確定性である。



(2-a) 構造化されない遷移　　(2-b) 構造化された遷移

例2. 構造化による状態遷移図の変化

この解釈に基づいた場合、次のようなモデルの解釈が出来上がる。

・NDFA仕様の構造化は、抽象段階で無視されていたシステムの微細構造に関する仕様を表面化するものであり、NDFA仕様を後ろ向きに修飾する。
・構造化された遷移過程の実行中(即ち微細な仕様の検証中)は、マシン(NDFA)の状態は変化せず、右辺のすべての仕様が成立した時点で状態遷移が起こる。

また、このモデル解釈は、現実的なシステムの設計に対しては、次の意味をもつ。

・処理構造の詳細化とともに表面化する処理条件の追加を、上位レベルで記述したNDFAの定義を書換えることなく行うことができる。
・処理の条件の列挙によって仕様を記述することが出来る。
・あるマシンが正常に動作するための内部的な条件をそのまま記述することができる。

3.2 スコープの設定

要素定義の有効範囲としてのスコープの設定の主な目的は以下の2つである。

・2-(2)で述べた階層化された仕様に生じる矛盾を回避する。
・設計方法論で提唱されるカプセル化による内部定義情報の隠蔽への対応。

ここで、第一の要求を充足するためには、次の規則を導入しなければならない。

(1) ある構造化定義の右辺に現れるすべての要素は、左辺が定義するマシンの内部要素でなければならない。
(2) ある要素と外的な要素との相互作用は、この両者を内部要素としてもつような上位システムの構造化定義の右辺でのみ定義される。

この規則によって、構造化定義の右辺に現れるある要素が、左辺で定義されるシステムの外部要素がもつ情報を必要とするような場合、そのシステムと外部要素の両者を内部要素としてもつような上位システムの構造化定義の右辺でシステム間の情報交換が行われ、この情報は上位のシステムからその情報を必要とする内部システムへデータフローとして渡されるように仕様記述が行われなければならないことになる(例1-b)。これは、データフロー図のプロセスバブルを情報隠蔽のカプセル境界として選んだことに相当する。

3.3 クラス定義とインスタンス定義

3.3.1 クラス-インスタンスの判別

単純に3.2のカプセル化に相当する情報のスコープを設定した場合、グローバルな定義をもつ数学関数などが必要な演算位置に記述できないような仕様記述の不自然さなどが考えられる。この不自然さは、従来のモデルがクラスとインスタンスの判別の手段を与えていないために起こる概念の混乱に起因している。従来のモデルでは、設計者がある要素の定義を記述する場合に、一般的な概念としての要素のクラスの振る舞いを定義しているのか、あるシステムの中での個々の要素の振る舞いを定義しているのかを意識することが出来ない。例えば、信号機を設計する場合、ランプは信号機の内部構造に含まれるが、ランプの標準的な定義は信号機の内部に存在するものではなく普遍的なものである。一方で、信号機システムに組み込まれた1つの部品としてのランプの振る舞いの定義は、ランプの標準的な定義の範囲内で信号機システム内部における役割のみを定義するもので、システムに含まれるそのランプの固有の定義である。つまり、あるマシンのクラス定義はシステム定義のカプセルの外部に存在するとしても、そのインスタンスであるマシンは、システムの内部構造を構成する部品として存在し得る。この混乱を回避するためにモデルにクラス-インスタンス定義の記述上の区別を導入する。

3.3.2 クラスの定義

情報のクラス化は、概念的には情報の全体集合から部分集合を作成することと同等の意味を持つ。但し、オブジェクト指向の概念に基づくクラス定義は、さらに以下の情報を含まなければならない。

・クラスの要素に与えられるメソッド定義。
・クラスの要素に与えられる属性定義。
・メソッドおよび属性の抽象化を実現するためのクラス階層の定義。

この内、インスタンスに与えられるメソッドおよび属性の定義は従来のモデルで記述可能であり、必要なのはクラスと従来のモデル定義を結び付ける規則と、クラスの階層を定義する規則である。ここでは、クラスをサブクラスやインスタンスの集合として表現する方法をとり、次の規則でクラスとインスタンスの定義を行う。

・クラスを非終端記号、インスタンスを終端記号で表し識別可能とする。
  クラス記号・・<SIN関数>, <電子>, 等.
  インスタンス記号・・SIN関数, 電子, 等.

・クラスは、その要素としてサブクラスを表す非終端記号またはインスタンスを表す終端記号を含む。
  <人工衛星>={<観測衛星>,<放送衛星>}.
  <観測衛星>={ 白鳥, 天馬, ひまわり }. 等

・クラスの要素に与えられるメソッドは、クラス名称に対応する素過程定義で表現する。
  <放送衛星> = { ゆり2号, ゆり3号 }.
  <太陽光 | 放送衛星(放電→充電) |>.
  <電波 | 放送衛星(充電→放電) | 増幅波>. 等

・クラスの要素に与えられる属性は、クラスに対して設定された条件で表現する。
  <運転手>={18以上|<熟練者>,<初心者>}.
  <初心者>={取得1年未満 | 太郎, 花子}. 等

ここで、クラス定義に付加される条件(HalfSpin,Unchargedなど)は、クラスの任意の要素に対して成立する述語名を表すことにして、述語をBNFで定義するものとする。これによって、素過程で定義されるメソッドで動的特性、BNFで定義される述語によって静的特性を表すことが可能となる。

3.3.3 インヘリタンスによる定義の変化

本モデルでは、クラスの任意の要素(サブクラスまたはインスタンス)はクラスに定義された素過程定義(メソッド)を動的特性の一部としてもち、クラスに定義された述語による属性判定を満足する。これにより起こる動的特性/静的特性の継承は、次のようなものとなる。

・メソッドの継承
継承の単位は素過程(つまり1状態遷移過程)であり、継承が起こるごとにNDFA定義の状態遷移図上では遷移線が増えることに対応する。これは、入出力アルファベット、状態および遷移関数の定義すべてに変化を与え得る。

・属性の継承
継承の単位は述語(BNFパーサ)であり、継承が起こるごとに要素が充足すべき条件が増加し、構造がより限定的になる。

4. OOAとの対応関係

ここで作成したモデルとOOA手法は例3に示すように、本質的な部分では良い対応関係を持つが、細部において以下の相違を生じる。

4.1 属性定義

OOAではクラス要素に与えられる属性が単に変数名で表されるのに対して、本モデルでは任意のクラス要素が充足すべき条件(述語名)で表している(例4)。両者は、モデルの述語が「要素に変数名のスロットがあることを確認する述語」である場合に意味的に一致する。

モデルにおけるこの拡張は、集合定義の手法に基づいて行われたものであるが、これにより現実の世界で行われているクラス化のうちOOAで表せない「goto文を含まない構造化プログラム」や「スピンが半整数であるフェルミ粒子」といったクラス分類の定義を可能としている。

4.2 メソッド定義

OOAでは、Gen-Spec図の表記上は単一メソッド(single-method)形式で、かつメソッド結合を想定していないサービス定義形式をとっている。これに対して、本モデルでは遷移過程定義をメソッドとして採用しているため、メソッド選択は多重メソッド(multiple-method)型となり、継承によりメソッドは単に多重化されるためメソッド結合[6]が起こる。但し、継承によりメソッドは単に加算されるという単純な結合規則に基づくものとなる。(例5)

4.3 Whole-Part構造のレンジ

本モデルにおいてBNFで定義される情報の構造は、OOAのWhole-Part構造に該当する。しかし、BNF定義はトップ-ダウンの階層構造を記述する能力しかもたないため、Whole-Part構造定義において上位要素が持ち得る下位要素のレンジは記述できても、下位要素が持ち得る上位要素のレンジは記述できない(例6)。

4.4 インスタンス結合

本モデルでは、独立したクラス間あるいはインスタンス間に存在し得る一般的な関係を記述するための特別な記述形式を用意していない。

現在のモデルでこれを表現しようとするならば、インスタンスの内部に「特定の関係を確立した他のインスタンスの名称を記録するスロット」を用意して、この値の対応関係で表すことになる。

< Particle > = { < Boson > , < Fermion > }.

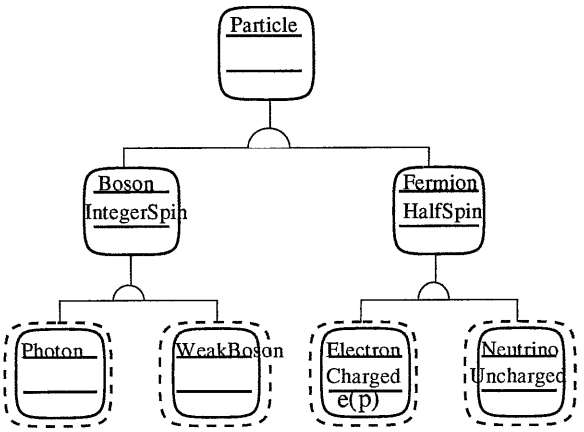
< Boson > = { IntegerSpin | < Photon > , <WeakBoson> }.

< Fermion > = { HalfSpin | < Electron > , < Neutrino > }.

< Neutrino > = { Uncharged | ν 1, ν 2 }.

< Electron > = { Charged | e1, e2 }.

<e(p) | Electron(q→ q+t) |e(p-t) >.



例3. Gen-Spec構造の記述

< Particle > ::= (<mass>, <charge>, <four-momentum> , < spin>).

<Uncharged> ::= (<mass> , <zero> , < four-momentum >, <spin>).

<zero> ::= 0 .

<Charged> ::= (<mass> , <non-zero> , < four-momentum >, <spin>).

<non-zero> ::= -1 | 1 .

例4. 例3に対する属性定義の記述

< Lepton > = { <Charged-Particle>, <Uncharged-Particle> }.

< ν (p) | Lepton(q→q+t) | ν (p-t)>.

< Charged-Particle > = { < Electron> }.

< e(p) | Charged-Particle(q→ q+t) | e(p-t)>.

< Electron > = { e1, e2 }.　　クラス定義

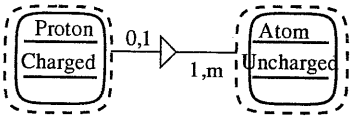
インスタンス(e1)に与えられる特性 ↓

< ν (p) | e1(q→q+t) | ν (p-t)>.

< e(p) | e1(q→ q+t) | e(p-t)>.

例5. メソッド定義と継承

<Atom> ::= <Proton>$^{(1,m)}$ <Neutron>$^{(0,m)}$ <Electron>$^{(1,m)}$.



例6. Whole-Part構造のレンジ記述

<Atom> = { H-1 , He-1 }. ・・(a)

<Atom> ::= (<Proton>$^{(1,m)}$, <Neutron>$^{(0,m)}$, <Electron>$^{(1,m)}$). ・・(b)

<Proton> = { p1 , p2, p3 }. ・・(c)

<Neutron> = { n1 , n2 }.

<Electron> = { e1 , e2 , e3 }.

<H-1> ::= (p1, φ , e1). ・・(d)

<He-1> ::= (p2 p3 , n1 n2 , e2 e3).

例7. 個々のインスタンス定義記述

4.5 個々のインスタンス定義

OOAで記述されるWhole-Part構造やインスタンス結合とは、Class&Objectの概念で表現される一般的な群と群との間に存在し得る関係を記述するもので、個々のインスタンス間に実際に存在する構造や関係は記述はできない。

本モデルで個々のインスタンスを定義する場合、例7に示す形式となる。例7に示されるように、While-Part構造定義(b,d)はクラス/インスタンスに関する形式上の相違がなく、クラス定義(a)からAtomはクラス、H-1 および He-1 が Atom のインスタンスであることが示される。この時、(d)は(a)のクラス定義に従って実態化された個々のシステムの構造を表す。

5. おわりに

リアルタイム構造化分析情報定義とOOA情報定義の両者の視点を与えるソフトウェアの記号モデルについて説明した。

ここで示したように、オブジェクト指向概念の導入は、リアルタイム構造化モデルに対して、スコープの設定やクラス-インスタンスの識別など、定義記述上のいくつかの規約として影響を与える。

作成したモデルとOOAの間には、クラス化の概念やインスタンス定義の記述レベルにおける差があり、また両者とプログラミング言語レベルで実際に使用されているオブジェクト指向の概念の間にもメソッド結合などの動的な定義解釈の点で差がある。これは、各形式が想定している記述レベルの差によるものであり、現実的なシステムの設計への適用性・有効性の面からさらに評価・改良を行っていく必要がある。

[参考文献]

[1] 松本, CASE指向抽象ソフトウェアモデル, 情報処理学会 ソフトウェア工学研究会報告 71−9, 1990.

[2] 松本, 湯原, CASE指向抽象ソフトウェアモデル(II), 情報処理学会 ソフトウェア工学研究会報告 77−10, 1991.

[3] Derek J. Hatley, Strategies for Real-Time System Specification, Dorset House Publishing Co., Inc., 1986.

[4] Peter Coad and Edward Yourdon, Object-Oriented Analysis, Prentice-Hall, Inc.,, 1991.

[5] Tom DeMarco, Structured Analysis and System Specification, YOURDON, Inc.,1979.

[6] 井田, 元吉, 大久保 編, Common Lisp オブジェクトシステム-CLOSとその周辺-, 共立出版, 1988.