

MDA : その数学的定式化

中村 正治
Masaharu NAKAMURA

1. まえがき

2014年度のFITにて、筆者はMDA, MDDの全体像を与える試みの論文を提出した。実践的な内容を報告ことが目的であり、実際、現実のプロジェクトへの応用に基づいたものでもあった。

しかしながら、モデリングアプローチは常に二つの問題を内包する。一つは、現実世界のモデルへのマッピングであり、これはモデリングの目的に応じた、恣意的なものにならざるを得ない。マッピングルールを定めたとしても、絶対的なルールなどは定めようもなく、議論を呼ぶマッピングが常に起こりうる。二つ目は、モデル自体のあいまいさである。ほとんどの文献で、モデルの定義はあいまいで厳密さを欠いており、その無矛盾性などは全く議論されていない。多くの解説書は、モデル自体の議論と現実世界とのマッピングを行きつ戻りつの記述であり、モデル自体のモデルが定義されていないという、奇妙な状態である。

多くのモデリング現場でも、権威と経験が結論を与える、というような場面が多く、およそ論理的とはいえない方法論が多くみられる。

2. 本論文の概要

本論文では、まずは、それぞれのモデルとそこに現れる概念の定義を与えることを試みた。コンピューター・サイエンスで扱うモデルは、ほとんど常に、クラス/インスタンスというメタモデル構造を持っており、それがモデル自体の定義を大変複雑にする。

定義部分だけ見れば、数学の一般的分野よりもかなり込み入っているのがわかる。いくつかの簡単な命題・補題とその証明は与えたが、今回は低利の構成に基づいた理論体系を与えるには至っていない。

ただ、このような定義体系そのものがあまりないように考えられるので、まずは、出発点として、報告することとする。

3. プロセスモデル

3.1 タスク

T : タスククラスの集合。 C : 条件クラスの集合。

任意の $t_1, t_2 \in T$ において $t_1 \neq t_2$ のとき、 T をタスククラスの正規化集合と呼ぶ。

$t_1 \neq t_2, c \in C$ のときに、 $s=[t_1, t_2, c]$ の組を順次クラスと定義する。

この時、任意の $t_1, t_2, t_3, t_4 \in T$ で、 $s_1=[t_1, t_2, c_1], s_2=[t_3, t_4, c_2]$ において、 $t_1=t_3$ ならば $c_1=c_2$ であると定義する。

$s_1=[t_1, t_2, c_1], s_2=[t_3, t_4, c_2], \dots, s_1, s_2 \in S$, かつ $(t_1 \neq t_3 \vee t_2 \neq t_4 \vee c_1 \neq c_2)$ のとき、 S を順次クラスの正規化集合と呼ぶ。

3.2 プロセス・クラス

$sp=[t, tp, cp]$ が存在して $sq=[tq, t, cq]$ なる tq, cq が存在しない、ただ一つの空のタスクを φ_s とし 始タスクと呼ぶ。

$sr=[tr, t, cr]$ が存在して $ss=[t, ts, cs]$ なる ts, cs が存在しない、ただ一つの空のタスクを φ_e とし 終タスクと呼ぶ。

c, d 成る関数を定義して、 $s=(t_1, t_2, c)$ としたとき、 $c(s)=t_1, d(s)=t_2$ と定義する。

順次クラス $\{s_1, s_2, \dots, s_n\} = p$, として、 $\forall si \in p$ で $\exists sj \mid d(s_1)=c(sj)$ ただし、 $si \neq \varphi_e$ および $sj \neq \varphi_s$ ならば、 p をプロセス・クラスと呼ぶ。

$sp=[t, tp, cp]$ が存在して $sq=[t, tq, cq]$ なる tq, cq が存在せず、しかも $t \neq \varphi_s$ の場合、 t を始端が開のタスクとよぶ。

$sr=[t, tr, cr]$ が存在して $ss=[ts, t, cs]$ なる ts, cs が存在せず、しかも $t \neq \varphi_e$ の場合、 t を終端が開のタスクとよぶ。

始端が開のタスクも、終端が開のタスクも、どちらのタスクも存在しないプロセス・クラスを、完備なプロセス・クラスと呼ぶ。

任意の $p_1, p_2 \in P$ について、 $p_1 \neq p_2$ ならば、 P をプロセス・クラスの正規化集合と呼ぶ。

3.3 ネストと規約タスク

単射 $n: \exists p \rightarrow t$ かつ $n(-1)$ が定義できて、 $n(-1): t \rightarrow p$ が単射の時、 n をネスティング、 t を p のネストプロセスと呼ぶ。

ある t において ネスティング(とその逆)が存在しない時、 t を既約タスクという。

3.4 タスクと条件のインスタンス

$T = \{t_1, t_2, \dots\}$ として $\forall t \in T \exists te \in T$ に単射 $m: t \rightarrow t$ を定義できるとき、 t は T のインスタンスと呼ぶ。

$C = \{c_1, c_2, \dots\}$ として $\forall c \in C \exists ce \in C$ に単射 $n: c \rightarrow c$ を定義できるとき、 c は C のインスタンスと呼ぶ。

3.5 フロークラスとフロー

$p = \{s_1, s_2, s_3, \dots, s_n\}$ なる p をプロセス・クラスとする。

順次クラス $x_1, x_2, \dots, x_m \in p, 1 \leq m \leq n$ として、 $f = \{x_1, x_2, \dots, x_m\}$ する。

このとき、 $\forall xi \in f$ で $\exists xj \mid d(x_1)=c(xj)$ ただし、 $xi \neq \varphi_e$ および $xj \neq \varphi_s$ なる $f \subset p$ をフローと呼ぶ。

$f_1, f_2, \dots, f_n \subset p$ なる $F = \{f_1, f_2, \dots, f_n\}$ をフロークラスと呼ぶ。

$f = \{x_1, x_2, x_3, \dots, x_n\}$ なるフロークラス f において、順次クラス $x_1, x_2, \dots, x_m \in p, 1 \leq m \leq n$ として、 $f = \{x_1, x_2, \dots, x_m\}$ する。

このとき、 $\forall x_i \in f$ で $\exists x_j \mid d(x_i)=c(x_j)$ ただし、 $x_i \neq \varphi_e$ および $x_j \neq \varphi_s$ なる x_j がただ一つしか存在しない場合、 f を STP フローと呼ぶ。

STP は Straight Through Process の略である。

命題

f が STP の場合、同時並行タスクは存在しない。

$\therefore$ 同時並行タスクとは $x_p=(t_1, t_2, c)$, $x_q=(t_1, t_3, c)$ のことであり、 $c(x_p)=c(x_q)$ となり、STP の定義に反する。 ■

命題

f が STP 場合、ループは存在しない。

$\therefore x_1=(t_1, t_2, c_1)$, $x_2=(t_2, t_3, c_2)$, $\dots$, $x_i=(t_i, t_{i+1}, c_i)$, $\dots$, $x_n=(t_n, t_1, c_n)$, $x_{n+1}=(t_1, t_{i+1}) \dots$ となり $c(x_{n+1})=t_i$, $c(x_i)=t_i$ となり、一つしかないとする定義に反する。 ■

3.6 プロセス・インスタンスとプロセス・ステート

p をプロセス・クラス、 $t_1, t_2, t_3 \dots t_i, t_j, \dots$ を p を構成するタスククラスとする。

ある、 t_i, t_j について、 $c(\dots c(o(t_j) \dots))=t_i$ のとき、 $t_i < t_j$ と記述する。

命題

この時 $d(\dots d(t_i) \dots)=t_j$ が成立する。これを $t_j > t_i$ と記述する。逆も真なので、

即ち、 $t_i < t_j$ と $t_j > t_i$ は同等である。

$t_i < t_j$ も $t_i > t_j$ が成立しない場合、 $t_i < > t_j$ と記述する。

証明

$c(\dots c(o(t_j) \dots))=t_i$ の両辺に繰り返し d を作用させる。■

t をタスククラス、 t のインスタンスを $\{t_1, t_2, \dots t_i \dots\}$ とする。

$Tt : t \rightarrow t_j \in \{t_1, t_2, \dots t_i \dots\}$ をインスタンス化と呼ぶ。

プロセス・クラス p を構成するタスククラスを、 $t_1, t_2, t_3, \dots t_n$ とする。

命名規則として、 $i < j$ のとき、 $t_i < t_j$ ないしは $t_i < > t_j$ とする。

$1 \leq j \leq n$ のある j にたいして、 $Tt_j : t_j \rightarrow t_{jp}$ なるインスタンス化があった時に、

$t_i < t_j$ を満たすすべての t_i には $Tt_i : t_i \rightarrow t_{iq}$ なるインスタンス化が存在する場合、

$t_1, t_2, t_3, \dots t_j$ を含むフローを $f_1, f_2, \dots f_x$ とすると、

$[f_1, t_1, t_2, t_3, \dots t_j]$, $[f_2, t_1, t_2, t_3, \dots t_j]$, $\dots$, $[f_x, t_1, t_2, t_3, \dots t_j]$ をプロセス・クラス p プロセス・インスタンスと呼ぶ。

この時、 $[t_1, t_2, t_3, \dots t_j]$ をプロセス・インスタンスのプロセス・ステートと呼ぶ。

クラス p のあるフロークラス f_y の全てのタスクにインスタンスが定義されていれば、「 p のフローは f_y で決定された」と表現する。

4. データモデル

4.1 エンティティ

$\mathbf{A}$: アトリビュート・クラスの集合。

$\mathbf{E}$: エンティティ・クラスの集合。

任意の $a_1, a_2 \in \mathbf{A}$, において $a_1 \neq a_2$ のとき、 $\mathbf{A}$ をアトリビュート・クラスの正規化集合と呼ぶ。

$\mathbf{A} = \{a_1, a_2, \dots\}$, この時、 $\forall a \in \mathbf{A} \exists a \in \mathbf{A}$ に単射 $m : a \rightarrow \mathbf{a}$ を定義できるとき、 $\mathbf{a}$ は $\mathbf{a}$ のインスタンスと呼ぶ。

任意の $e_1, e_2 \in \mathbf{E}$, において $e_1 \neq e_2$ のとき、 $\mathbf{E}$ をエンティティ・クラスの正規化集合と呼ぶ。

$\mathbf{E} = \{e_1, e_2, \dots\}$, この時、 $\forall e \in \mathbf{E} \exists e \in \mathbf{E}$ に単射 $n : e \rightarrow \mathbf{e}$ を定義できるとき、 $\mathbf{e}$ は $\mathbf{e}$ のインスタンスと呼ぶ。

$\mathbf{A1}$ をあるアトリビュート・クラスの正規化集合、 $e_1 \in \mathbf{E}$ をあるエンティティとすると、 $[e_1, \mathbf{A1}]$ をアトリビュートが定義されたエンティティと呼ぶ。

$\mathbf{E} = \{e_1, e_2, \dots e_n\}$ をエンティティ・クラスの正規化集合、文字列を $s_1, s_2, \dots s_n$ として、

$i \neq j$ ならば、 $s_i \neq s_j$ で、 (e_i, s_i) の組が定義できるならば、 s_i を e_i の「意味づけられた表現」と呼ぶ。

$\mathbf{A} = \{a_1, a_2, \dots a_n\}$ をアトリビュート・クラスの正規化集合、文字列を $z_1, z_2, \dots z_n$ として、 $i \neq j$ ならば、 $z_i \neq z_j$ で、 (a_i, z_i) の組が定義できるならば、 z_i を a_i の「意味づけられた表現」と呼ぶ。

4.2 正規化エンティティ

エンティティ・クラス $\mathbf{e} = \{e_1, e_2, \dots e_n\}$ のとき、定義により、可附番なので順番に自然数を与えることができる。

この時、 $\forall i, j \ 1 \leq i, j \leq n$ で $e_i \neq e_j$ ならば、 $\mathbf{e}$ は正規化エンティティである、という。

この正規化エンティティ $\mathbf{e}$ で自然数 $i : 1 < i \leq n$ と自然数 k を与え、全単射 $X : i \rightarrow k$ が定義されるとき、この k をエンティティ・インスタンス e_i のインデックスと呼ぶ。

定義から明らかのように、 $e_n \neq e_m$ ならば $n \neq m$ 。逆も正しい。

$\mathbf{A}$ をアトリビュート・クラスの集合で、 $\mathbf{A} = \{a_1, a_2, a_3, \dots\}$

$[e, \mathbf{A}]$ が定義できる時、これを正規化エンティティ $\mathbf{e}$ のアトリビュートリストと呼ぶ。

エンティティ・インスタンス $e_p \in \mathbf{e}$ が存在して、 $\mathbf{A} | e$ で、 $\exists a_1x \in a_1, \exists a_2y \in a_2, \dots$ のとき、

これらを、 e_p の値と呼び、 $[e_p | a_1x, a_2y, \dots]$ と表記し、 e_p の値と呼ぶ。

エンティティとアトリビュートに関して、次の公理を与える。

公理 : 非区別性公理

$\exists e_1, e_2 \in \mathbf{e}$ で $e_1 \neq e_2$ のとき、 $[e, \mathbf{A}]$ において、 $\exists a_1x \in a_1, \exists a_2y \in a_2, \dots$ を適切に選んで、

$[e_1 | a_1x, a_2y, \dots]$, $[e_2 | a_1x, a_2y, \dots]$ を定義できる。

補題 : 区別性定理

$\exists e_1, e_2 \in \mathbf{e}$ で $e_1 \neq e_2$ のとき、 $[e, \mathbf{Ap}]$ において、 $\exists a_1x \in a_1, \exists a_2y \in a_2, \dots$ を適切に選んで、

$[e_1 | a_1x, a_2y, \dots]$, $[e_2 | a_1x, a_2y, \dots]$ を定義したとき、

$\mathbf{Ap} \subset \mathbf{Aq}$ なるアトリビュート・クラス集合で $\mathbf{aq} \in \mathbf{Aq} \wedge \mathbf{aq} \notin \mathbf{Ap}$ で $\exists \mathbf{aq1}, \mathbf{aq2} \in \mathbf{aq} \wedge \mathbf{aq1} \neq \mathbf{aq2}$ を適切に選んで $[\mathbf{e}, \mathbf{Aq}]$ を定義し、

$[\mathbf{e1} \mid \mathbf{a1x}, \mathbf{a2y}, \dots, \mathbf{aq1}], [\mathbf{e2} \mid \mathbf{a1x}, \mathbf{a2y}, \dots, \mathbf{aq2}]$ を定義できる。

証明

$\mathbf{N} \subset$ 自然数 なる $\mathbf{N}$ を選ばば、定義により、 $\mathbf{n1}, \mathbf{n2} \in \mathbf{N} \wedge \mathbf{n1} \neq \mathbf{n2}$ をえらんで、 $\mathbf{aq1} = \mathbf{n1}, \mathbf{aq2} = \mathbf{n2}$ とすることが出来る。 ■

これらを合わせて、存在整合性とよぶ。

4.3 エンティティとインスタンスの表現

存在整合性を利用すると、エンティティとインスタンスを、視覚的に表現する根拠が得られる。

インスタンスを「一行」で表現し、その行に一から始まる連続した整数 (したがって、ユニークな数) である「行番号」を割り当てれば、この番号をインスタンスの識別子として利用できる。一から始まることや、連続していることは要件ではないので、行数から離れて、ユニークな任意の数を用いてもよい。

これらの数は、この複数行によるインスタンスの存在の整合性を担保するものと解釈できる。本論文では、必要があればこの数をインデックスと呼ぶ。

全てのインスタンスには、アトリビュート・クラスで定義されたアトリビュート・インスタンスが定義されている。

アトリビュート・クラスを「列」とすれば、インスタンスの集合は、個々のインスタンスを表現する列と、アトリビュート・クラスとそのインスタンスを表現する列のよって、「表」として表現することもできる。

この表現の正当性は自明であるので、特に証明は付さない。

4.4 正規化データモデル

エンティティ・クラス $\mathbf{ex}, \mathbf{ey} \in \mathbf{E}$ において $\forall \mathbf{ey} \in \mathbf{ey} \mid \exists$ 単射 $\mathbf{r}: \mathbf{ey} \rightarrow \mathbf{ex}, \mathbf{ex} \in \mathbf{ex}$ のとき、 $\mathbf{ey}$ は $\mathbf{ex}$ に従属的という。

任意の $\mathbf{ey}, \mathbf{ex} \in \mathbf{E}$ で $\mathbf{E}$ が正規化集合の場合、 $\mathbf{ey}$ が $\mathbf{ex}$ に従属的で、かつ $\mathbf{ex}$ は $\mathbf{ey}$ に従属的でなければ、 $\mathbf{E}$ は第三正規化集合であるという。

現実世界の実体と対応させて、アトリビュート $\mathbf{a1}, \mathbf{a2}, \dots$ 、第三正規化エンティティ $\mathbf{e1}, \mathbf{e2}, \dots$ が定義できて、これらが、クラス、インスタンスの定義と矛盾しない場合、この対応を正規化データモデルと呼ぶ。

エンティティ・クラス $\mathbf{ex}, \mathbf{ey}, \mathbf{ez} \in \mathbf{E}$ において

$(\forall \mathbf{ey} \in \mathbf{ey} \mid \exists$ 単射 $\mathbf{r}: \mathbf{ey} \rightarrow \mathbf{ex}, \mathbf{ex} \in \mathbf{ex}) \wedge (\forall \mathbf{ez} \in \mathbf{ez} \mid \exists$ 単射 $\mathbf{p}: \mathbf{ez} \rightarrow \mathbf{ex}, \mathbf{ex} \in \mathbf{ex})$ のとき、 $\mathbf{ey}, \mathbf{ez}$ は $\mathbf{ex}$ に従属的という。

この場合も、定義により、 $\mathbf{E}$ は第三正規化集合である。

エンティティ・クラス $\mathbf{eq}, \mathbf{ex}, \mathbf{ey} \in \mathbf{E}$ において

$(\forall \mathbf{ey} \in \mathbf{ey} \mid \exists$ 単射 $\mathbf{r}: \mathbf{ey} \rightarrow \mathbf{ex}, \mathbf{ex} \in \mathbf{ex}) \wedge (\forall \mathbf{ey} \in \mathbf{ey} \mid \exists$ 単射 $\mathbf{p}: \mathbf{ey} \rightarrow \mathbf{eq}, \mathbf{eq} \in \mathbf{eq})$ のとき、 $\mathbf{ey}$ は $\mathbf{ex}, \mathbf{eq}$ に従属的という。

同様、この場合も、定義により、 $\mathbf{E}$ は第三正規化集合である。

5. オブジェクトモデル

5.1 オブジェクト・クラスとその継承

$\mathbf{D}$ を任意のアトリビュート・クラスの集合、 $\mathbf{M}$ をプログラムモジュールの集合とする。

$\mathbf{m} \in \mathbf{M}$ が $\mathbf{p1}, \mathbf{p2}, \mathbf{p3} \dots \mathbf{pm}, \mathbf{q1}, \mathbf{q2}, \mathbf{q3} \dots \mathbf{qn} \in \mathbf{D} \mid \mathbf{m}: (\mathbf{p1}, \mathbf{p2}, \mathbf{p3} \dots \mathbf{pm}) \rightarrow (\mathbf{q1}, \mathbf{q2}, \mathbf{q3} \dots \mathbf{qn})$ のとき、 $\mathbf{m}$ をメソッドと呼ぶ。

$\mathbf{m}$ をメソッドの集合とする。

$\mathbf{p1}, \mathbf{p2}, \mathbf{p3} \dots \mathbf{pm} \in \mathbf{D}, \mathbf{m1}, \mathbf{m2}, \mathbf{m3} \dots \mathbf{mn} \in \mathbf{m}$ のとき、

$\{\{\mathbf{p1}, \mathbf{p2}, \mathbf{p3} \dots \mathbf{pm}\}, \{\mathbf{m1}, \mathbf{m2}, \mathbf{m3} \dots \mathbf{mn}\}\}$ なる集合を、オブジェクト・クラスと呼ぶ。

あるオブジェクト・クラスを構成するアトリビュートを、このオブジェクト・クラスのプロパティと呼ぶ。

オブジェクト・クラス $\mathbf{o1}, \mathbf{o2}$ を構成するプロパティとメソッドがすべて等しいとき、

$\mathbf{o1} = \mathbf{o2}$ と記述する。

$\mathbf{o1}, \mathbf{o2}$ をオブジェクト・クラス、 $\mathbf{D1}, \mathbf{D2}$ をアトリビュート・クラスの集合、 $\mathbf{m1}, \mathbf{m2}$ をメソッドの集合としたとき、

$\mathbf{o1} \neq \mathbf{o2} \wedge \mathbf{D1} \subseteq \mathbf{D2} \wedge \mathbf{m1} \subseteq \mathbf{m2}$,

このとき、 $\mathbf{o2}$ を $\mathbf{o1}$ の継承 (Inheritance) と呼ぶ。

$\mathbf{o1}$ を $\mathbf{o2}$ の上位クラス (Superclass)、同様に、 $\mathbf{o2}$ を $\mathbf{o1}$ の下位クラス (Subclass) と呼ぶ。

$\mathbf{o1} = \{\{\mathbf{p1}, \mathbf{p2}, \dots, \mathbf{pi}\}, \{\mathbf{m1}, \mathbf{m2}, \dots, \mathbf{mj}\}\},$

$\mathbf{o2} = \{\{\mathbf{p1}, \mathbf{p2}, \dots, \mathbf{pi+1}, \mathbf{pi+2}, \dots, \mathbf{px}\}, \{\mathbf{m1}, \mathbf{m2}, \dots, \mathbf{mj+1}, \mathbf{mj+2}, \dots, \mathbf{my}\}\}$ の時、

$\mathbf{o2} = \{\{\mathbf{o1}\}, \{\mathbf{pi+1}, \mathbf{pi+2}, \dots, \mathbf{px}\}, \{\mathbf{mj+1}, \mathbf{mj+2}, \dots, \mathbf{my}\}\}$ と記す。

この記法の類推で、

$\mathbf{cx} = \{\{\mathbf{o1}, \mathbf{o2}, \mathbf{o3} \dots\}, \{\mathbf{p} \dots\}, \{\mathbf{m} \dots\}\}$ と記した時、 $\mathbf{cx}$ を $\mathbf{o1}, \mathbf{o2}, \mathbf{o3} \dots$ の多重継承 (Multi-Inheritance) と呼ぶ。

5.2 オブジェクトのインスタンス

$\mathbf{p1}, \mathbf{p2}, \mathbf{p3} \dots \mathbf{pm} \in \mathbf{D}, \mathbf{m1}, \mathbf{m2}, \mathbf{m3} \dots \mathbf{mn} \in \mathbf{m}$ として、 $\{\{\mathbf{p1}, \mathbf{p2}, \mathbf{p3} \dots \mathbf{pm}\}, \{\mathbf{m1}, \mathbf{m2}, \mathbf{m3} \dots \mathbf{mn}\}\}$ なるオブジェクト・クラス $\mathbf{o}$ において、

$\mathbf{p1}, \mathbf{p2}, \mathbf{p3} \dots, \mathbf{pm}$ をそれぞれ $\mathbf{p1}, \mathbf{p2}, \mathbf{p3} \dots \mathbf{pm}$ のそれぞれのインスタンスとする。

このとき、 $\mathbf{o} = \{\{\mathbf{(p1, p1)}, \mathbf{(p2, p2)}, \mathbf{(p3, p3)} \dots \mathbf{(pm, pm)}\}, \{\mathbf{m1}, \mathbf{m2}, \mathbf{m3} \dots \mathbf{mn}\}\}$ と書いて、

$\mathbf{o}$ の $\mathbf{(p1, p1)}, \mathbf{(p2, p2)}, \mathbf{(p3, p3)} \dots \mathbf{(pm, pm)}$ の値を取るインスタンスと呼ぶ。

$\exists (\mathbf{pi}, \mathbf{pi}), (\mathbf{pj}, \mathbf{pj}) \dots \mid 1 \leq i, j \leq m$ で

$\mathbf{IO(-1)} ((\mathbf{pi}, \mathbf{pi}), (\mathbf{pj}, \mathbf{pj}), \dots)$:

$\mathbf{o} = \{\{\mathbf{(p1, p1)}, \mathbf{(p2, p2)}, \mathbf{(p3, p3)} \dots \mathbf{(pm, pm)}\} \rightarrow \mathbf{IO}((\mathbf{pi}, \mathbf{pi}), (\mathbf{pj}, \mathbf{pj}), \dots)$

が単射の時、 $\mathbf{IO}$ をオブジェクト・クラスのインスタンス化と呼び、 $(\mathbf{pi}, \mathbf{pi}), (\mathbf{pj}, \mathbf{pj}) \dots$ を $\mathbf{o}$ の初期状態とよぶ。

当然ながら、

$\exists (\mathbf{pi}, \mathbf{pxi}), (\mathbf{pj}, \mathbf{pyj}) \dots \mid 1 \leq i, j \leq m$ で

$\mathbf{IO}((\mathbf{pi}, \mathbf{pxi}), (\mathbf{pj}, \mathbf{pyj}), \dots): \mathbf{o} = \{\{\mathbf{p1}, \mathbf{p2}, \dots \mathbf{pm}\}, \{\mathbf{m1}, \mathbf{m2}, \dots \mathbf{mn}\}\} \rightarrow \phi$

即ち、インスタンス化が不可能なアトリビュート・インスタンスの組を定義することができる。

5.3 オブジェクト・インスタンスの発展

$\exists (\mathbf{pi}, \text{pi}), (\mathbf{pj}, \text{pj}) \cdots |1 \leq i, j \leq m$ で

$\mathbf{IO}((\mathbf{pi}, \text{pi}), (\mathbf{pj}, \text{pj}), \cdots) : \mathbf{o} = \{ \mathbf{p1}, \mathbf{p2}, \cdots, \mathbf{pm} \}, \{ m1, m2, \cdots, mn \} \rightarrow \mathbf{o} = \{ (\mathbf{p1}, \text{p1}), (\mathbf{p2}, \text{p2}), (\mathbf{p3}, \text{p3}) \cdots (\mathbf{pm}, \text{pm}) \}$

を なる初期状態を持つオブジェクト・インスタンスとにおいて、

$\exists \text{pix, piy}$ で $(\mathbf{pi}, \text{pix}), (\mathbf{pi}, \text{piy}) \cdots |1 \leq i \leq m$ のとき、

$D : \mathbf{o} \rightarrow D(\mathbf{o})$ なる写像が存在し、

$\mathbf{o} = \{ (\mathbf{p1}, \text{p1}), \cdots (\mathbf{pi}, \text{pix}) \cdots (\mathbf{pm}, \text{pm}) \}$

$D(\mathbf{o}) = \{ (\mathbf{p1}, \text{p1}), \cdots, (\mathbf{pi}, \text{piy}) \cdots (\mathbf{pm}, \text{pm}) \}$ のとき、定義から明らかなように、

$\mathbf{IO}(-1) (D(\mathbf{o})) = \mathbf{o}$

つまり、 $D(\mathbf{o})$ も、オブジェクト・クラス $\mathbf{o}$ のオブジェクト・インスタンスの一つである。

この D をアトリビュート $\mathbf{pi}$ についての単独の発展 (Development) と呼ぶ

$D<-1> : D(\mathbf{o}) \rightarrow \mathbf{o}$ は、自明で定義できる。

$D<0> : \mathbf{o} \rightarrow \mathbf{o}$

を定義した上で、

$Dd : \mathbf{o} \rightarrow Dd(\mathbf{o})$ なる写像 $Dd(\mathbf{o}) = \{ (\mathbf{p1}, \text{p1}), \cdots, (\mathbf{pi}, \text{pid}) \cdots (\mathbf{pm}, \text{pm}) \}$ 全体の集合に $D<-1>$ 、 $D<0>$ を追加したものを $\mathbf{D}$ とし

$Dp, Dq \in \mathbf{D}$ のとき 演算 $*$ を $Dp * Dq(\mathbf{o}) = Dp(Dq(\mathbf{o}))$ と定義すれば、

$Dp, Dq, Dr \in \mathbf{D}$ のとき $Dp(Dq(Dr(\mathbf{o}))) = Dp(Dq * Dr(\mathbf{o})) = Dp * Dq(Dr(\mathbf{o}))$ なので、結合則が成り立って、演算 $*$ によって、 $\mathbf{o}$ 上に群が定義される。

しかも定義から明らかなように、 $Dp * Dq = Dq * Dp$ なので可換群である。

さらに一般化して、

$D : \mathbf{o} \rightarrow D(\mathbf{o})$ なる写像が存在し、

$\mathbf{o} = \{ (\mathbf{p1}, \text{p1}), \cdots (\mathbf{pi}, \text{pix}), \cdots (\mathbf{pj}, \text{pjx}) \cdots (\mathbf{pm}, \text{pm}) \}$

$D(\mathbf{o}) = \{ (\mathbf{p1}, \text{p1}), \cdots, (\mathbf{pi}, \text{piy}) \cdots, (\mathbf{pj}, \text{pijy}) \cdots (\mathbf{pm}, \text{pm}) \}$ のとき、

同じく

$D<-1> : D(\mathbf{o}) \rightarrow \mathbf{o}$

$D<0> : \mathbf{o} \rightarrow \mathbf{o}$

を定義した上で、

$Dd : \mathbf{o} \rightarrow Dd(\mathbf{o})$ なる写像 $Dd(\mathbf{o}) = \{ (\mathbf{p1}, \text{p1}), \cdots, (\mathbf{pi}, \text{pid}) \cdots (\mathbf{pj}, \text{pid}) \cdots (\mathbf{pm}, \text{pm}) \}$ 全体の集合に $D<-1>$ 、 $D<0>$ を追加したものを $\mathbf{D}$ とし

$Dp, Dq \in \mathbf{D}$ のとき 演算 $*$ を $Dp * Dq(\mathbf{o}) = Dp(Dq(\mathbf{o}))$ と定義すれば、同様に、結合則は成立し、演算 $*$ によって、 $\mathbf{o}$ 上に群が定義される。

しかしながら、定義上、 $Dp * Dq = Dq * Dp$ を保証するものではなく、非可換群と考えるのが一般的である。

発展が 3 個以上のアトリビュート・クラスについてのものである場合に一般化すれば、演算の順序が重要な意味を持つことがわかる。

一般的に、

$\mathbf{IO}(-1) : \mathbf{o1} \rightarrow \mathbf{o}$

$\mathbf{IO}(-1) : \mathbf{o2} \rightarrow \mathbf{o}$

$D : \mathbf{o1} \rightarrow \mathbf{o2}$

$\mathbf{o1} = \{ (\mathbf{p1}, \text{p11}), \cdots (\mathbf{pi}, \text{pi1}), \cdots (\mathbf{pj}, \text{pj1}) \cdots (\mathbf{pm}, \text{pm1}) \}$

$\mathbf{o2} = \{ (\mathbf{p1}, \text{p12}), \cdots (\mathbf{pi}, \text{pi2}), \cdots (\mathbf{pj}, \text{pj2}) \cdots (\mathbf{pm}, \text{pm2}) \}$

とすると、 $\mathbf{o}$ は定義の中に $\mathbf{D}$ を含んでいない。

すなわち、インスタンス化は、写像に与える変数のみで決定されるので、オブジェクト・クラスはステートレスである、と表現する。