モデル理論アプローチによるシミュレーション

― 自律分散システムの開発手順 ―

Model Theory Approach for Simulation
: Development Procedure of Autonomous Distributed Systems

旭　　貴朗

１．はじめに
２．自律分散系の構造
３．単数要素のモデル
４．複数要素のモデル
５．考察

１．はじめに

本稿の目的は，簡単な自律分散システムのシミュレーション開発を例にして，モデル理論アプローチにおけるシミュレーション開発手順の特徴を抽出することである．

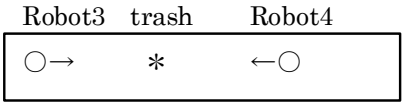
対象とする自律分散システムとは，複数の意思決定要素（学習機能を含む）が制約のもとに行動しつつ，相互作用をおこなっている複合体のことである．本稿では，ひとつの部屋のなかで複数の掃除ロボットがゴミを拾ってまわるような状況を想定している．掃除ロボットは自律的に判断し行動することができるが，壁や衝立てなどの障害物による行動制約があり，さらに他のロボットと接触すると進行方向を変化させるなど物理的な相互作用もある．このような複雑システムのシミュレーションを行なうには，その状況（要素の動作や環境の変化）を表わすモデルが必要である．同じ対象に対して複数のモデルが考えられるが，本稿では標準的な階層構造（学習，選択，実行）のモデルを考える．

モデル理論アプローチでは，対象に対して数理モデルを構築し，シミュレーション開発を行なうことが特徴である．しかしながら，その数理モデルが現実のプログラムとかけ離れたものならば，シミュレーション開発者にとって役に立つものにはならない．本稿では情報システムに対するモデル理論アプローチ[1]の立場から議論を進める．モデル理論アプローチが提案する CAST というモデル記述言語[2]では，数理モデルがほとんどそのまま CAST によるモデルとなり，開発実行環境 Simcast [3]を用いてシミュレーションを実行することができる．そのため容易にモデルの整合性を実際の動作でチェックすることができる．

以下，まず第２節で自律分散システムに対する数理モデルの構造を提案する．第３節と４節では，簡単な例として，細長い部屋を２つのロボットが掃除するシステムの具体的なモデルを示す．第５節でシミュレーション開発の特徴抽出を行い，自律分散システムのための若干の示唆を述べる．

２．自律分散システムの構造

自律分散システムの簡単な例として，自律走行する複数のロボットをひとつの部屋に配置して掃除を行うような状況を考える．図表１は細長い部屋の平面図である．

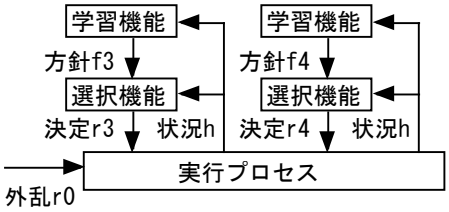


図表１　自律分散システムの例

掃除ロボットの本体底面に吸引装置があり，ロボットがゴミの位置を通過すれば，ゴミは吸い取られるものとする．また，ロボットの前面にはセンサーが取り付けられており，障害物（壁や衝立）を検知することができるものとする．またこの部屋は細長いので，２つのロボットがすれ違うことができないものとする．

さて，効率的に掃除をするには，どのような走行方式（行動ルール）をロボットに与えればよいだろうか．単純に直進し，障害物があれば方向転換するだけでよいのだろうか．ゴミを吸引し通り過ぎたあとにもゴミが発生するので，適度に直進し適度に方向を変える必要があるかもしれない．ではどのような行動ルールにもとづいて走行すれば，最も多くゴミを収集できるであろうか．最適な行動ルールを求めることがシミュレーションの目的となる．

本稿では，システム理論の考え方にもとづき，学習や選択の機能を実行プロセスと区別することにする[4，5]．すなわち，図表２のように，認識や学習や代替案の選択を行なう決定プロセスと決定にもとづく実行プロセスを切り離してモデル化する．これらを区別する理由は，要素の行動が互いの相互作用を受けるので，実行前の予想と実行後の結果が異なることがあるからである．



図表２　自律分散システムの構造

図表２は自律分散システムを表わすブロック線図である．図中の「実行プロセス」には，ロボット３とロボット４の行動，およびそれらの相互作用や（外乱としての）ゴミの発生が含まれており，その状況の変化をモデル化することになる．例えば２つのロボットが同じ位置に移動しようとしてもすれ違うことができないので，モデルはそのような制約を満足する必要がある．ここでは実行プロセスをオートマトン（正確には「状態遷移システム」）としてモデル化する．また，ゴミの発生は状態に変化を与える「外乱」であり入力の一種であるが，人間が手入力することは面倒なので，本稿では乱数を使って自動的にゴミを発生させる予定であり，モデルとしては実行プロセスに内包させるので，ここでは変数 r0を明示化しない．

各ロボットの「選択機能」は，実行プロセスの状況 h を認識し，次の行動のための実行方針 r3，r4を決定する過程である．選択は，上位の学習機能から得られた選択方針 f3，f4にもとづいて行なう．ここでは選択機能を「静的システム（状態を持たないシステム）」として定義する．ただし各ロボットは目前だけしか知覚できないとするので，ロボットが認識する内容は状況 h 全体ではなく部分的なものである．不完全な認識にもとづく意思決定となる．

次に，各ロボットの「学習機能」は，選択機能と同じく，実行プロセスの状況 h を（部分的かも知れないが）認識し，次の選択のための方針 f3，f4を決定する過程である．ここでは学習機能を「状態遷移システム」として定式化する．学習のためには，過去の結果と現在の方針を含む記憶 fw が必要である．過去の結果を現在の結果と比較して，新しい選択方針 f3と再び記憶すべき内容を決定することになる．また，下位の選択機能に新しい選択方針を伝達する．

以上，学習機能と実行プロセスは状態遷移システムである．選択機能は静的システムである．全体システムはそれらが図表２のように結合した複合システムである．したがって，通常のオートマトン理論により，全体システムは状態遷移システムとなる．モデル理論アプローチでは，モデル理論の意味のモデルを構成し，シミュレーション開発を行なうことが特徴である．

３．単数要素のモデル

まずは，ひとつのロボットが掃除をしているシステムのモデルを作成し，実行してみたい．本節では，図表２のそれぞれの機能の具体的な定義を行なっていく．

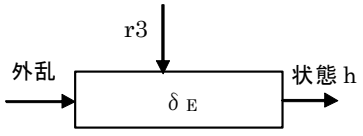
(1) 実行プロセスのモデル

最初に，物理的状況（図表１）の変化を表わす「実行プロセス」を状態遷移システムとして定式化する．部屋の長さを30とし，ロボットやゴミの位置を表わすベクトル変数を，

e = (e1, e2, e3, ..., e30)

とする．各項 ei (i=1,2,...,30) は数値0, 1, 2, 3, 4のいずれかを値としてとるものとする．数値の意味は，

ei = 0 —— 空所
ei = 1 —— ゴミ
ei = 2 —— 壁
ei = 3 —— ロボット3
ei = 4 —— ロボット4



図表３　実行プロセス

であり，各数値を識別番号と呼ぶことにする．

例えば，

e = (2, 0, 0, 3, 0, 0, 0, 0, 1, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2)

は，両サイドが壁であり，位置4にロボット３がおり，位置９にゴミがあり，位置12にロボット４がいることを表わしている．

そのほか各ロボットは移動しつつゴミを収集しているのであるから，変数 c3 = (3, u3, w3) を導入する．それぞれ，３はロボットの識別番号，変数 u3は移動方向（1または -1）を，w3はすでに吸引したゴミの量を表わす．したがって全体の状況は，次のように表わせる．

h = (e, c3) = (e, (3,u3,w3))

変数 h はオートマトン理論では「状態」と呼ばれる．ここで 3つの集合を定義する．

N4 = {0,1,2,3}, N30 = N4×N4×...×N4, E = { e | e∈N30, (∃! *i*)(e*i* = 3) }.

ただし，記号×は集合の直積である．すると，とり得る状態の範囲である「状態集合」H は次のように定義できる．

H = E×({3}×{1, -1}×非負整数集合)

ここに，記号 ヨ！は唯一存在を表わす限定子である．

さて，「入力」r3はロボットが決定した移動方針であり，方向の継続性を表わすものである．機械的には現在の移動方向をそのまま続けるのか，逆方向に向かうのかを，駆動部に伝える制御信号に相当する．ここでは，次の時刻での移動方向が現在の方向と同じなら+1，逆方向に転換するなら-1の値をとるものとする．したがって「入力集合」は {1, -1}である．

また，状況変化のダイナミクス（状態遷移関数）δ_E : H×{1,-1}→H を定式化する．つまり，現在の「状態」が h = (e, (u3, w3))であるとき，r3が入力された場合，次の時刻における状態

δ_E ((e, (u3,w3)), r3) = (e', (3, u3',w3'))

を定義する．まず，ゴミは30回に 1 回空所にランダムに最大 3 個発生するものとする．したがって，もしも吸引しない場合は空所がなくなり，最大28個発生するだけである（吸引されれば補充される）．

e1 = (e の空所にランダムに1を記入する)

次の時刻での方向は,

u3' = r3*u3

となる．またロボット n の現在の位置が x = projection^{-1}(e1,n)であるとき，次の位置は x' = x+u3'となるはずである．ここに，projection(e1,i)はベクトル e の中から第 i 項 ei の値を取り出す関数であり，projection^{-1}(e1,n)はその逆関数である．しかしながら，そこに障害物（壁２や他のロボット４）があるときには移動することができないので，x3' = x3となる．ロボット3の前方の識別番号は,

p = projection(e1, x + u3')

であり，前方が p=0（空所）または1（ゴミ）ならば，そこに移動することができるので,

$$x' = \begin{cases} x+u3' & \text{if} \quad p < 2 \\ x & \text{otherwise} \end{cases}$$

となる．新しい位置 x'が確定すれば，状況 e'は e1の第 i 項を n (= 3)に置き換えればよい．これは制約と相互作用を定式化している．さらにその位置 x' にゴミがあれば，吸引量は 1 増加する．

$$w' = \begin{cases} w+1 & \text{if} \quad \text{projection}(e1,x') = 1 \\ w & \text{otherwise} \end{cases}$$

以上の定式化を，あえて等号を含む述語論理学における論理式で記述するならば，次のようになる．

δ_E ((e, (n, u, w)), r3) = (e', (n, u', w')) ⇔

e1 = (e の空所にランダムに1を記入する),
u' = r3*u,
x = projejection^{-1}(e1, n),
p = projection(e1, x + u'),
(p < 2) → (x' = x + u') otherwise (x' = x),
(p = 1) → (w' = w+1) otherwise (w' = w),
e' = (e1の位置 x'に3を記入する).

ここに，記号 ⇔ は必要十分条件を表わす論理記号であり，直感的には右辺で左辺を定義していると考えればよい.

以上により，実行プロセスのモデル <{1, -1}, H,δ_E > が定義された．この数理モデルを言語 CAST で記述したものが付録１の中の状態遷移関数 deltaE である．言語 CAST がモデルを記述する言語であることから，数理モデルとシミュレーションモデルがほとんど同形である．以下，付録の中の各述語の定義は文献[2, 3]を参照されたい.

モデル理論アプローチにおけるモデルは，論理学のモデル理論におけるモデルと同じ意味であって，集合や関係（関数）の組のことである．シミュレーションへの適用では，主にオートマトンモデルを扱う．一般にオートマトンのモデルは5項組〈入力集合，出力集合，状態集合，状態遷移関数，出力関数〉で表現する．項の数は場合によって変化する. 例えば, 実行プロセスでは出力を考えず状態遷移だけを考えるので, 上記のような3項組になっている．また各集合と各関係（関数）が具体的に定義されたときモデルと呼ぶことにする.

(2) 選択機能のモデル

次にロボットの行動選択の機能 decide3の定式化を行う．選択機能は，実行プロセスの状況 h を認識し，次の行動のための方針 r3を決定する過程である．ロボットは前面にセンサーを備えているので面前の障害物を発見することができる．実行プロセスの状況が h=(e, (3,u3,w3))であるとき，ロボット3の前方の識別番号は,

p = projection(e, projection^{-1}(e,3)+u3)

である．前方が p=0（空所）または１（ゴミ）ならば前進することができる．選択のために使用できるのは変数 p だけである．全状況 h をそのまま認識しているわけではない.

さてシミュレーションの目的はゴミの収集量を増やすことにある．そのため本稿で注目するのは前進の継続性である．ゴミを吸引し通り過ぎたあとにもゴミが発生するので，適度に直進し適度に方向転換する必要があろう．ここでは，確率 f3 (0≦f3≦1) で前進を継続するものとしたい．実際には，一様乱数 t (0 < t < 1)を発生させ，前方に障害物がなく(p < 2) かつ乱数 t が数値 f3以下なら前進を継続することとしたい．そこで，選択機能を表わす関数 decide3(h,f3) =を次のように定義する.

$$r3=\begin{cases} 1 & \text{if}\quad p<2\text{かつ}\ t<f3 \\ -1 & \text{otherwise} \end{cases}$$

したがって，集合 F を0以上1以下の実数の集合とすると，「入力集合」は H×F であり，「出力集合」は{1, -1}である．「(状態のない) 状態遷移関数」の定義域と値域は decide3: H×F→{1, -1} と書け，その内容を論理式で表現するなら，

decide3([e,(3,u3,w3)],f3) = r3 ⇔

p = projection(e, projection^{-1}(e,3)+u3),

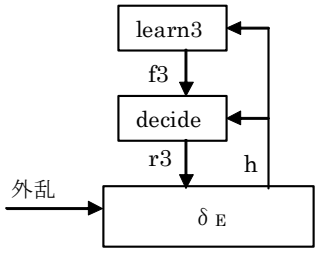
t = 乱数,

(p < 2かつ t < f3) → (r3 = 1) otherwise (r3 = -1).

と書くことができる．以上により，選択機能のオートマトンモデル < H×F, {1, -1}, decide3 > が定義された．以下では，r3 = 1のとき前進を継続するので，確率 f3を「前進確率」と呼ぶことにする．

(3) 学習機能のモデル

一方，前進確率 f3を定める学習機能について考える．学習には記憶が必要である．ここでは，学習機能は状況 h=(e,(3,u3,w3))の中から，1000シミュレーションサイクル (すなわち移動回数1000歩) ごとに，ゴミ収集量 w3のみを観測しているとし，過去の実績と現在の収集量 w3を比較し，収集量の増加を目指して前進確率 f3を変化させるものとする．過去の実績を記憶しておく必要から，学習機能を「状態遷移システム」として定式化することになる．



図表４　学習を含むシステム

具体的には，学習機能の「状態」fws は現在の前進確率 f3および過去（1001回前）の前進確率 f0，過去（1001回前）のゴミ収集量 w0の組とする．fws = (f3, f0, w0)．したがって，「状態集合」は，Fws = F×F×非負整数集合 である．また，「入力」は実質的には1000サイクル毎のゴミ収集量 w3であるが，表面上は h である．したがって，「状態遷移関数」の定義域と値域は learn3: Fws×H→Fws となる．

learn3((f3, f0, w0),h) = (f3',f0',w0')

は，「1000回に１回状況 h の中の w3を観測し，もしもゴミ収集量 w3が前回 w0から１割以上減少したら，以前の確率 f0に小さな乱数（−1以上1以下）を加算して次回の確率 f3'とする．そうでない場合には確率は変化させない」ように定式化する．

$$f3'=\begin{cases} f3 & \text{if}\quad w3 \geqq 0.9\cdot w0 \\ f0 + \text{小さな乱数} & \text{if}\quad w3 < 0.9\cdot w0 \end{cases}$$

これを論理式で表現すると，次のようになる．

learn3((f3, f0, w0), (e, (n,u,w))) = c' ⇔
(cycle%1000 = 1) → ((w≧0.9*w0) → (c' = (f3, f3, w)),
(w < 0.9*w0) → (tt = f0 + 小さな乱数,
ff3 = max(0, min(1, tt)),
c' = (ff3, f3, w)))
otherwise (c' = (f3, f0, w0)).

以上により，学習機能のモデル < H, Fws, learn3 > が定義された．また，学習機能から選択機能への指示は，関数 out : Fws → F　s.t.　out(f3, f0, w0) = f3 として定義する．

(4) 全体システムのモデル

以上，学習機能と実行プロセスは状態遷移システムである．選択機能は静的システムである．全体システムはそれらが図表4のように結合した複合システムである．したがって，通常のオートマトン理論により，全体システムは状態遷移システムとなる．

その「状態」は c = (h,fws) であり，「状態集合」は H×Fws である．入力は考えないので「状態遷移関数」は δ: H×Fws → H×Fws となり，それは次のように定式化できる．

δ([h,fws]) = c' ⇔
fws' = learn3(fws, h),
f3' = out(fws'),
r3' = decide3(f3', h),
h' =deltaE(h, r3'),
c' = (h', fws'). ----- (1)

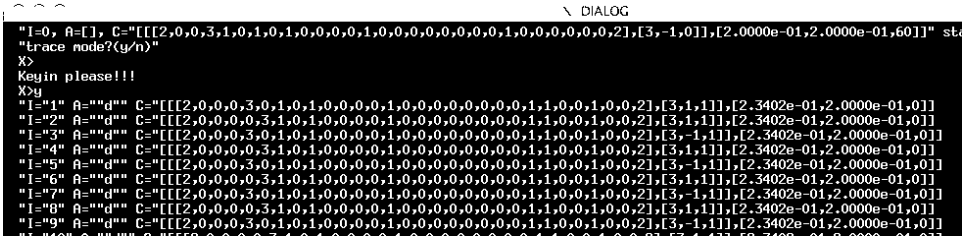
よって，全体システムはオートマトン <H×Fws, δ> としてモデル化できた．

(5) シミュレーションの実行

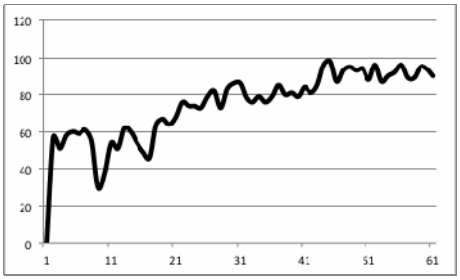
以上により，学習機能をもつ要素から成る単体の自律システム（図表４）のモデル <H×Fw3,δ> が定義された．これにより，シミュレーションモデルを開発し，実行することができる．付録1は学習を含む場合の言語 CAST によるシミュレーションモデル（ファイル名 walker21.set）である．言語 CAST が数理モデルを記述する言語であることから，数理モデルとシミュレーションモデルがほとんど同形であることが特徴である．ただし，Simcast の仕様により，時間を進ませるために，入力変数 a があるが，これはダミーである．

付録１をシミュレーション開発実行環境 Simcast で実行することができる．Simcast は，オートマトンモデルと問題解決モデルをコンパイルして実行するシミュレーション環境であるが，ここではオートマトンモデルだけを使用する．図表５は実行中の様子である．入力変数 a にダミーとして記号 "d" を強制的に300万回入力している．

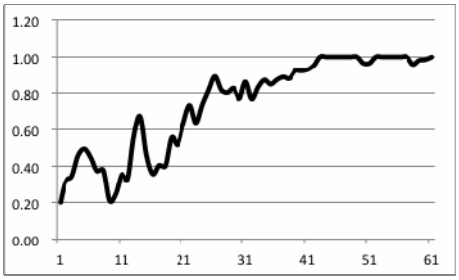
図表５には，サイクル回数 I，全体システムへの入力 a，全体システムの状態 c が



図表５　単数要素のシミュレーション



図表６　前進確率 f3の変化



図表７　吸引量 w3の変化

並んでいる．最初は前進確率が小さいので，ロボット３が左右に動いているだけであるが，時間が経過するにつれ変動幅が大きくなる様子を見ることができる．図表６，７にシミュレーションサイクル300万回の実行のうち５万回毎に記録した実績を示す．

一つのロボットが部屋にいて掃除する場合は，図表６をみると，前進を継続しているほうが多数のゴミを収集できるので，前進確率が高くなっていく．わずかに上下する他は，ほとんど正確に確率1.00を維持することになる．前進確率が高くなるにつれて，収集量も高くなり，図表７のように高い水準（1000サイクルあたり90個以上）を維持していく．つまり，高い収集量を得るためには単純に往復運動を続けることが効果的であることが分かる．

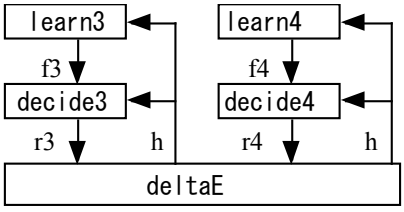
４　複数要素のモデル

次に学習機能をもつ２つのロボットが部屋にいて掃除する状況（図表８）を考える．ここでは，robot4の学習機能と選択機能はrobot3と同じものとする．

learn4(fw,h) = learn3(fw,h)

decide4(f,h) = decide3(f,h)

また実行プロセス δ_E は２つの入力 r3，r4 に対応するよう修正する．



図表８　複数要素のシステム

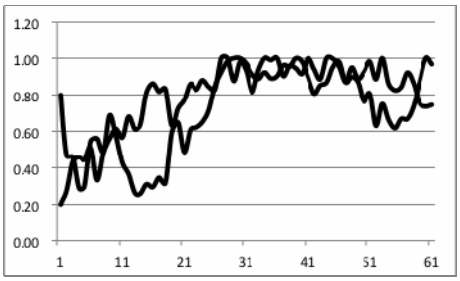
全体システムはそれらが図表８のように結合した複合システムである．したがって，通常のオートマトン理論により，全体システムは状態遷移システムとなる．その「状態」は (h,(fw3,fw4)) であり，「状態集合」

は H×(Fw3×Fw4)である．また，「状態遷移関数」は δ: H×(Fw3×Fw4) → H×(Fw3×Fw4) となり，次のように定義できる．

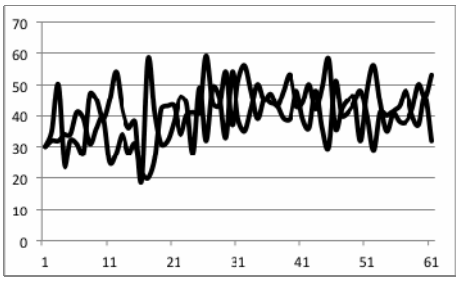
δ([h,(fw3,fw4)]) = c' ⇔
(fw3', fw4') = (learn3(fw3, h), learn4(fw4, h)),
(f3', f4') = (out(fw3'), out(fw4')),
(r3', r4') = (decide3(h, f3'), decide4(h, f4')),
h' =deltaE(h, (r3', r4')),
c' = (h', (fw3', fw4')). ----- (2)

よって，全体システムはオートマトン < H×(Fw3×Fw4), δ > としてモデル化できた．

付録２はそのような場合の言語 CAST によるシミュレーションモデル（ファイル名 walker30.set）である．シミュレーション実行途中の様子と，シミュレーションサイクル300万回の実行のうち，５万回ごとに記録した実績を図表９，10に示す．



図表９　前進確率 f3, f4の変化



図表10　吸引量 w3, w4の変化

前進確率については，単純に考えて f3 = f4 = 0.5となるものと事前に予想していたが，そうではなかった．図表9が示すように，確率 f3と f4はほぼ並行に高い水準に向かっていく．数値の大きさが逆転することもある。一方，収集量については，図表10にあるように，200万サイクルを過ぎれば，ゴミ収集量は両者とも1000サイクル毎に常に30～60個で，変動しつつもほぼ同等水準を保っている（発生したゴミはすべて吸引されている）．よって，高い収集量を得るためには，両者ともできるだけ前進を継続することが効果的である．ただし，ゴミの発生量を少なくし30サイクルごとに2つに設定すると棲み分けが発生し，前進確率 f3と f4のどちらかが高い水準になり他方が低い水準になるケースもあったが，正確な実証はしていない．

５．考察

本稿では自律分散システムの数理モデルの構造を提案した（第２節）．また，簡単な例として細長い部屋での2つの掃除ロボットの動作の数理モデルとシミュレーションモデルを作成し，動作することを確認した（第３節，第４節）．この簡単なモデルと初期条件に限って言えば，ロボットが一つの場合でも二つの場合でも，「高い収集量を得るためには，両者ともできるだけ前進を継続することが効果的である」ことが分かった．シミュレーション結果としては特に新奇ではない．

しかしながら，本稿の目的はシミュレーションそのものではなく，開発手順の特徴を抽出することであった．第２節～第４節で見たように，基本的な開発手順は次のようになる．

システム構造（ブロック線図）→部分機能のモデル→全体のモデル
→コーディング→コンパイル→実行

流れの上段は数理モデルの開発であり，下段はコンピュータへの実装である．

一般のシミュレーション開発と異なる点は，モデル理論の意味のモデルを構成していくことにある．全体の見取り図に相当するブロック線図（図表２）をもとに，各部分機能のモデルをまず作成し，それらを結合して全体システムのモデルを作成する．モデルは「アルゴリズム」ではなく，論理式による「定義」である．部分機能のモデルを結合するためには，オートマトン理論の標準的な知識（複合システムのモデル）が必要である．また状態遷移関数 δ や δ_E だけでなく，定義域や値域を入力集合や出力集合として明示することが特徴的である．そのため，基礎となる定義を組合せて複雑なものを定義するという意味の論理的な順序を維持でき，記述の論理性が高まる．開発者にとって説明記述が容易で，かつ読者にとって理解しやすいという利点があろう．

コーディングについては，コンピュータ言語としての CAST が数式（論理式）をほとんどそのまま記述することに優れているので手間はかからない．数理モデルを開発することが直ちにシミュレーションにつながることが特徴である．当然ながら，状態の定義や状態遷移関数の定義が直ちに重要である．一方，入力集合や出力集合の明示はコーディング時の入力エラー処理に役立つものと思われる．慣れてくれば，むしろ数理モデルを作らず，ブロック線図を眺めながら，いきなり部分機能をコーディング・実行しつつシミュレーションモデルを作成することになろう．

また，副産物として，自律分散システムという特定のシミュレーション開発にあたっての次のような示唆を得た．

1) 自律分散システムの構造を学習機能（認識→調節）と選択機能（認識→決定）と実行プロセスの標準的な3階層に分けて考えることができる．
2) しかし，記憶が必要な学習機能と実行プロセスは状態遷移システムとして定式化するものの，選択機能を静的システムとして定式化することは特徴的である．決定機能だけ静的システムとして定式化する必要があるのは，リアルタイムな実行が要求されるからである．
3) 学習機能と選択機能の両方に，状況の認識が必要であるが，その内容はもちろん異なるはずである．定式化においては認識機能を分離せず，それぞれの機能の中に部分機能として記述するほうがモデルとしては明晰である．
4) 状況の認識は，もちろん全状況を認識することではなく，意味的にもプログラム的にも自分に関連する部分的認識にすぎないが，定式化においてはそれぞれの機能への「入力」として全状況 h を記述する必要がある．その理由は，全体からどの部分を切り出す（部分を特定化する）かは場合によって異なり，シミュレーションに自由度を持たせるためである．

【付録 1】

```
//walker21.set : One learning Walker
func([trash,learn,decide,deltaE,mover]);
inputsequence()="d";
times()=3000001;
initialstate()=c0 <-> c3:=[3,-1,0],
        [n,u,w]:=c3, e:=walker10wall.lib,
        e0:=replaceList(e,4,n),
        c0:=[[e0,c3],[0.2,0.2,60]], cycle.g:=0;
delta([h,fws],a)=cc <->
        cycle=cycle.g+1, cycle.g:=cycle,
        fwss:=learn(fws,h),
        [f33,f3,w3]:=fwss,
        r3:=decide(h,f33),
        hh:=deltaE(h,r3),
        cc:=[hh,fwss];
learn([f3,f0,w0],[e,c3])=fwss <->
        (cycle.g%1000=1)->(
            [n,u,w]:=c3,
            (w>=0.9*w0)->(fwss:=[f3,f3,w]),
            (w <0.9*w0)->(
                tt:=f0+(2*myrandom()-1)/20,
                ff3:=max([0,min([1,tt])]),
                fwss:=[ff3,f3,w] ) )          //方針変更
        otherwise (fwss:=[f3,f0,w0]);
decide([e,c3],f3)=r3 <->
        [n,u,w]:=c3,
        p:=project(e,invproject(e,n)+u),          //前方認識
        myrandom(t),
        (p<2,t<f3)->(r3:=1) otherwise (r3:=-1); //方向判断
deltaE([e,c3],r3)=[ee,cc3] <->
        (cycle.g%30=1)->(e1:=trash(e,3)) otherwise (e1:=e),
        [ee,cc3]:=mover([e1,c3],r3);
mover([e,c3],r3)=[ee,cc3] <->
        [n,u,w]:=c3, uu:=r3*u, x:=invproject(e,n),
        (cycle.g%1000=1)->(ws:=0) otherwise (ws:=w),
        p:=project(e,x+uu),                          //衝突有無
        (p<2)->(xx:=x+uu) otherwise (xx:=x),//位置決定
        (p=1)->(ww:=ws+1) otherwise (ww:=ws),//ゴミ吸引
        [ee]:=replaceMatrix([e],[[1,x,0],[1,xx,n]]),
        cc3:=[n,uu,ww];
trash(e,m)=e1 <->
        L:=defList(p(z,n,[e]),member(n,genIndex(1,m))),
        [e1]:=replaceMatrix([e],L);
        p(z,n,[e]) <-> myrandom(r), x:=floor(28*r+2),
        q:=project(e,x),q=0,z:=[1,x,1];
```

【付録 2】

```
//walker30.set : Two learning Walkers
func([trash,learn,decide,deltaE,mover]);
inputsequence()="d";
times()=3000001;
initialstate()=c0 <-> c3:=[3,-1,0], c4:=[4,1,0],
        e:=walker30wall.lib,
        [e0]:=replaceMatrix([e],[[1,4,3],[1,15,4]]),
        c0:=[[e0,c3,c4],[0.2,0.2,30],[0.2,0.2,30]], cycle.g:=0;
delta([h,fw3,fw4],a)=cc <->
        cycle=cycle.g+1, cycle.g:=cycle,
        [e,c3,c4]:=h,
        fws:=[learn(fw3,[e,c3]), learn(fw4,[e,c4])],
        [[f33,f3,w3], [f44,f4,w4]]:=fws,
        [r3,r4]:=[decide([e,c3],f33), decide([e,c4],f44)],
        hh:= deltaE(h,r3,r4),
        cc:= [hh, [f33,f3,w3], [f44,f4,w4]];
learn([f33,f3,w3],[e,c3])=cc <-> 【付録 1 と同じ】
decide([e,c3],f3)=r3 <-> 【付録 1 と同じ】
deltaE([e,c3,c4],r3,r4)=[ee,cc3,cc4] <->
        (cycle.g%30=1)->(e1:=trash(e,2)) otherwise (e1:=e),
        [e2,cc3]:=mover([e1,c3],r3),
        [ee,cc4]:=mover([e2,c4],r4);
mover([e,c3],r3)=[ee,cc3] <-> 【付録 1 と同じ】
trash(e,m)=e1 <-> 【付録 1 と同じ】
```

【注】

[1] 高原康彦ほか（2007）『形式手法 モデル理論アプローチ：情報システム開発の基礎』日科技連

[2] 旭貴朗，高原康彦，中野文平ほか（2008. 03）「経営情報システム開発のためのモデル記述言語 CAST」経営情報学会誌，Vol.16，No.4，pp. 19-30

[3] 旭貴朗（2009. 03）「モデル理論アプローチによるシミュレーション---開発実行環境 Simcast」東洋大学経営論集73号，pp. 33-51

[4] 高原康彦（1983）「階層的組織における意思決定」宮沢光一編著『経営意思決定』ダイヤモンド社

[5] ただし，本稿の提案する階層構造は伝統的なものであり，人工知能・ロボット工学における包摂アーキテクチャー（Subsumption Architecture）を含んではいないことに注意すること.

（2011年 9 月12日受理）