個体群動態論に特化したマルチエージェントシミュレーション基盤 Re:Mobidyc

小田 朋宏
株式会社 SRA
tomohiro@sra.co.jp

Gael Dur
静岡大学
dur.gael@shizuoka.ac.jp

要旨

Re:Mobidyc は生物の個体群動態に特化したマルチエージェントシミュレーション *(MAS)* 基盤である．科学研究の道具としての要請に応えるために，*Re:Mobidyc* ではスケーラビリティ，再現性，検証可能性を重視してモデリング言語，モデリング環境および実行系を設計した．本論文では，*Re:Mobidyc* のモデリング言語，モデリング環境および実行系について，個体群動態研究からの要求に基づいた設計指針および実現について紹介し議論する．

1. 背景

Agent-Based Modelling and Simulation (ABMS) は，1990 年代以降の環境学研究に広く応用され，様々な構文および意味論を持つシミュレーションツールが開発されてきた．特に，個体の行動をエージェントとしてモデリングする個体ベースモデル (Individual Based Model, IBM) は，個体差の表現や，各個体の一生の中での行動の変化（生活史段階），環境への適応と進化，個体から環境への影響などを表現するための柔軟性を持つことから，環境学や生物学の研究に用いられてきた [1]．現在でも NetLogo [2] をはじめとする多様なマルチエージェントシミュレーション (Multi-Agent Simulation, MAS) 環境が開発され，環境学や生物学の多様な領域の ABMS に利用されている．

ソフトウェア研究の視点からは，ABMS は LOGO をはじめとするプログラミング言語の応用と見なすことができる．それらのプログラミング言語は汎用のプログラミング言語であり，プログラムの構成要素は生物学や環境学の用語ではなく，プログラミングの用語で定義されている．例えば，LOGO は手続き型のプログラミング言語であり，タートルの移動のための専用の命令文や，変数への代入文や，条件分岐やループなどの制御構造や，手続き呼び出しにより，タートルの動きを定義する．プログラミング言語のこれらの機能は，チューリング完全なプログラムを構成するものであり， 計算の表現の強力さとともに，ループの停止性やプログラムの副作用など，ソフトウェア研究がこれまで取り組んできた多くの困難をもたらす．また，環境学や生物学で扱う個体の機能要素は必ずしも汎用プログラミング言語の機能要素と対応していない．例えば，LOGO のタートルを生物の個体のメタファーとして利用する場合，生物の変態による個体の行動の変容をシミュレーションに反映させるためには，ユーザプログラムによりその行動の変容の仕組みを実装する必要がある．IBM を構成するためには，プログラミング言語と環境学，生物学の間のギャップを埋める必要がある．

ABMS を科学研究に応用する場合，そのモデルおよびシミュレーション結果が，他の科学者に理解され，その特性が検証され，再現可能であることが望ましい．本研究では，生物学研究における個体群動態を ABMS の適用対象として，モデリング言語およびシミュレーション環境を一種のドメイン特化言語 (Domain Specific Language, DSL) として設計し実現した．DSL を設計実現する上で，プログラミング言語の構文定義や意味定義，実行系の設計や実装については多くの技術が確立されている．本研究では，特に汎用のプログラミング言語および処理系にはない，ABMS に特化した要求に注目して，モデリング言語およびシミュレーション環境を設計・実装した．

以後，第 2 節では，個体群動態の研究に利用されてきた他の ABMS ツールを示し，本研究と比較する．第 3 節では，個体群動態のシミュレーションに求められる要求

を議論し，第 4 節では本研究で開発しているシミュレーション環境 Re:Mobidyc で要求をどのように満たしたかを説明する．第 5 節で Re:Mobidyc 上のシミュレーションの例を紹介し，第 6 節でまとめと今後の課題を示す．

2. 関連研究

2.1. NetLogo

NetLogo [2] は Java で実装された LOGO ベースのマルチエージェントモデリング言語であり，自然環境や社会での現象のモデリングに広く利用されている．NetLogo は移動可能なタートルと空間を格子状に分割したパッチの，2 種類のエージェントを扱い，それぞれのエージェントを並行動作させる．各エージェントの動作は命令的な手続きにより定義される．

NetLogo は個体群動態にも適用可能であるが，生活史段階など個体群動態に必要な概念が言語機能に反映されていないため，例えば各エージェントの生活史段階を属性として保持して，各手続きの中でその生活史段階に対する条件分岐によって，各段階での振る舞いを記述するなどの工夫をする必要がある．また，各エージェントの行動の記述は命令型プログラミングそのものであり，モデリングにはプログラミング特有のスキルが要求される．Re:Mobidyc は個体群動態のシミュレーションに特化し，プログラミング特有のスキル要求を排除することを設計指針としている点で異なる．

2.2. Cormas

Cormas [4] は Smalltalk で実装されたマルチエージェントモデリング言語であり，共有資源管理のための住民参加型の意思決定の支援を目的として設計された．シミュレーションによる理解共有を促進するために，Cormas のシミュレーション環境は対話性および可視化を大きな特徴として備えている．

Cormas は，移動可能な社会的エージェント，および，空間を格子状に分割した空間的エージェント，さまざまな仲介を行う受動的エージェントの 3 種類のエージェントによりシミュレーションモデルを構成する．各種エージェントは Smalltalk のサブクラスとして定義され，それぞれのエージェントの行動は Smalltalk ベースの内部 DSL により定義される．内部 DSL により統計関数など多くの機能が提供されるが，モデリングにおけるエージェントの行動は基本的に Smalltalk のメソッドとして記述される．したがって，モデリングには Smalltalk プログラミングのスキルが要求される．Re:Mobidyc は，Cormas と同じく Smalltalk 上に構築されているが，エージェントの行動を記述する DSL を外部 DSL として，構文や言語機能について個体群動体論特有の概念を取り入れている点が異なる．

2.3. Mobidyc

Mobidyc [3] は個体群動態のシミュレーションを目的に Smalltalk 上に開発されたマルチエージェントモデリング環境であり，生活史段階を言語の基本要素として表現可能なモデリング言語を備えている．Mobidyc では，移動可能な個体を表す Animat，空間を格子状に分割した Cell，時系列データの読み込みや記録をおこなう NonLocatedAgent の 3 種類のエージェントでシミュレーションモデルを構成する．Animat の行動は，他の Animat や Cell との相互干渉を行う Task のリストとして定義される．Eat や Move 等の多くの生物種の個体に共通した行動が Task として組み込まれているほか，MathScript と呼ばれる DSL によってユーザ定義することも可能である．ユーザーは，シミュレーションに用いる生物種のそれぞれについて，卵，幼生，成体など複数の生活史段階を定義し，生活史段階ごとに Task のリストを定義する．

Re:Mobidyc は Mobidyc の設計指針を受け継いで，個体群動態に必要な言語機能やモデリング環境を実現した上で，スケーラビリティおよび再現性，検証可能性の向上を図り設計された．

3. 個体群動態シミュレーションに求められる特性

本節では，現在おこなわれている個体群動態でのシミュレーションの利用状況から，個体群動態のためのシミュレーションに求められる特性を挙げる．

R1：生活史段階の表現

個体群動態において重要な概念が，生活史段階である．生活史段階とは，個体の一生のなかで生態の変化を捉えたもので，卵生の動物では大きく分けて卵，幼生，幼体，未成体，成体などの段階を持ち，種によってそれぞれより細分化された段階を持つ．図 1 に動物性プランクトンの一種であるカイアシ類 Eurytemora affinis の生活史段

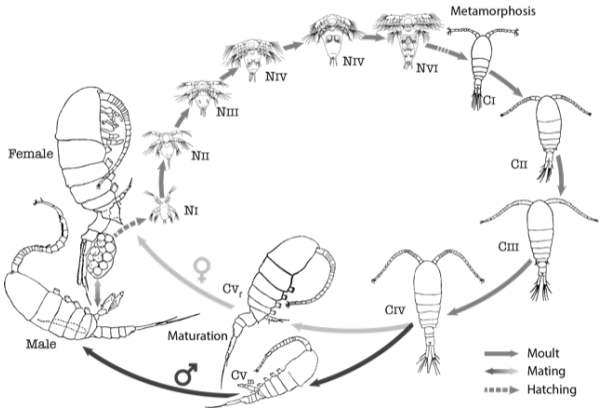

図 1. 生活史段階の例

階を示す．卵から孵化してノープリウス幼生 (N_I〜N_{VI}) の各生活史段階を脱皮により順に進み，N_{VI} からコペポディド幼生 C_I に変態し，ふたたび脱皮により順に C_{II} から C_V に脱皮し，C_V では雌雄に分化する．

個体群動態のシミュレーションでは，大量の個体の行動をモデリングしコンピュータ上でシミュレーションすることで，群れの個体数の分布や経時変化を予測する．個体の行動特性は生活史段階により大きく異なるため，個体の行動モデルは生物種ごとではなく，生活史段階ごとに定義される．生活史段階ごとに，個体の移動，捕食，代謝，交配，生殖などの行動の定義に加え，次の生活史段階への遷移（脱皮または変態）が定義される．

R2：個体および環境の相互作用

個体の行動をモデリングするために，個体と他の個体および環境との相互作用を扱うことが必要である．例えば，動物の行動には，近接する餌を捕食するなど，他の個体との間の相互作用として行動をモデリングする．また，個体の行動は，水温，水の透明度，酸素濃度，二酸化炭素濃度，餌など，位置に依存した局所的な環境が持つ量により影響を受けると同時に，個体が二酸化炭素を排出するなど環境に影響を与えることから，個体と局所的環境の間の相互作用を表現する必要がある．既存のシミュレーション環境の多くでは，パッチまたはセルと呼ばれる小区画をエージェントとして扱い，個体をあらわすエージェントとの間の相互作用が記述可能である．

R3：プログラミングスキル要求の軽減

個体群動態シミュレーションの目的は生物学や環境学の研究であり，求められるプログラミング言語への習熟が可能な限り低いことが望ましい．命令的なプログラミングは，破壊代入や制御構造や再帰呼び出しなど，プログラミング特有の概念を習得することが求められるため，その利用は最低限に留めることが望ましい．

R4：分析可能性，再現性

シミュレーションを仮説検証に用いる場合，シミュレーションするモデルが，検証対象である仮説に忠実であることが求められる．モデリング言語が LOGO 等の命令型言語である場合，個体の行動を変数への代入やループによる繰り返しによる手続きとして記述することになるが，その手続きが仮説を忠実にプログラム化したものであることを検証することは，一般に容易ではない．個体の行動を記述するモデリング言語は，数理的な分析に適していることが求められる．シミュレーションを科学研究で用いるため，シミュレーション上で発生した事象は，同じモデル同じ初期条件から再現できることが望ましい．

R5：スケーラビリティ

シミュレーションは一定の時間幅を決めた離散時間により，状態の更新を行う．筆者の 1 人はカイアシ類の個体群動態の研究で Mobidyc 上のシミュレーションを行なっている．これまでは，最長 5 年分のシミュレーションを 6 時間刻みで，すなわち 7,300 ステップを実行し，最大 300,000 個体のシミュレーションを行なった．1 個体あたり 64 ビットの数値データを格納する領域を 10 スロット保持すると，300,000 個体を 7,300 ステップ実行すると，163G バイト強のデータ量になる．

今後，シミュレーションによる個体群動態論研究を進めるためには，より長いシミュレーション期間，より短い刻み幅，より多い個体数を対象にしたシミュレーションが求められることが予想される．Re:Mobidyc は，より多くのステップ数や，より多くの個体数を求める要請に応えることが期待されている．

4. Re:Mobidyc の概要

筆者らは現在，Re:Mobidyc の実装を進めている．Re:Mobidyc は Pharo Smalltalk [1] 上に構築され，github レポジトリ [2] 上にオープンソースソフトウェアとして公開されている．開発はまだ完了していないが，既にいくつかの簡単なシミュレーションが実行可能になっている．本節では，Re:Mobidyc の設計および実装にあたって基準にした重要な指針を示す．

[1] https://pharo.org/
[2] https://github.com/tomooda/ReMobidyc/

4.1. モデリング言語

Re:Mobidyc はマルチエージェントシミュレーション環境であり，シミュレーションモデルを構成する基本単位は，動物の個体を表す Animat，環境を格子状に分割した Cell，環境全体を表す World の 3 種類のエージェントである．各エージェントはその状態を特定するための属性と，取り得る行動を記述したタスクにより定義される．本論文では，エージェントの種類とインスタンスを区別するために，後者を指す時には明示的にインスタンスと呼ぶ．また，コンピュータ操作環境と，シミュレーションモデル中の空間を指す環境を区別するために，本論文ではコンピュータ操作環境をシミュレーション環境またはモデリング環境と呼び，シミュレーションモデル中の個体の外界としての環境を単に環境と呼ぶ．

Animat は具体的な対象の種類ごとに定義される．〈**R1：生活史段階の表現**〉として挙げた通り，個体群動態では脱皮や変態によって生活史段階を遷移していくことが重要である．Re:Mobidyc ではそれぞれの種類の Animat は，それぞれ 1 つの生活史段階を表現する．すなわち，Re:Mobidyc では，1 つの個体は，その個体の誕生から死までのシミュレーション時刻を連続的にカバーし，かつ，同一シミュレーション時刻に重複しない，いくつかの Animat のインスタンスを継いで接続したものとして表現される．図 1 のカイアシ類 Eurytemora affinis の雌の個体の場合には，`Egg`，`N1`〜`N6`，`C1`〜`C4`，`C5f` のそれぞれのインスタンスを連結したものになる．

Cell は環境をグリッド状に分割した区画をあらわす．現在は矩形分割のみを実装しているが，六角形分割にも対応する予定である．Cell は 1 つのシミュレーションモデルに 1 種類のみ定義され，`Cell` という識別子で参照される．World は環境全体をあらわすエージェントであり，Cell と同様に，1 つのシミュレーションモデルに 1 種類のみ定義され，`World` として参照される．

図 2 に Animat の定義例を示す．`Goat` は，その属性として `x`，`y`，`blood_sugar`，タスクとして `respire`，`move`，`graze` の 3 つを遂行する Animat として定義されている．ただし，`x` および `y` は Animat に暗黙に定義されている属性である．

属性の定義

Re:Mobidyc では，各インスタンスが保持する属性は数値のみを扱い，全ての属性宣言に単位を宣言することを義務付けている．宣言された単位は後述のタスク定義の計算式の中で次元の整合性の検査に使われる．属性の単位の宣言は，自然科学研究で用いられる単位を使って計算式の整合性を検査してモデルの誤りを発見することで，〈**R3**: プログラミングスキル要求の軽減〉の実現に貢献することを目的に導入した．また，計算式中の数値リテラルや外部ファイルから読み込まれる数値データの単位の違いによる誤りを防ぐことを目的に，数値リテラルおよび外部ファイル中の数値データには単位を明示することを要求する．Re:Mobidyc 実行系は数値を全て国際単位系 (SI) に変換して計算を行う．属性値はメモリ中には SI に変換された値で保持され，属性宣言で与えられた単位に変換して表示する．

```
Goat is Animat with
    blood_sugar [kcal].

Goat respire namely lose where
    my amount -> my blood_sugar
    my loss ->
        min(my amount, 2000 [kcal/day] * Δtime).

Goat move where
    my heading -> direction neighbor's grass
    my speed -> 1[km/day].

Goat graze namely collect where
    my thing -> my blood_sugar
    here's thing -> here's grass
    my greed -> min(
        10000[kcal]-my blood_sugar,
        3000[kcal/day] * delta time,
        here's grass).
```

図 2. Animat 定義の例

図 2 の中で，エージェント `Goat` の属性は以下の部分で宣言されている．

```
Goat is Animat with
    blood_sugar [kcal].
```

属性 `blood_sugar` が単位 `kcal` 付きで宣言されているが，Animat は必ず空間中の位置を持つため，x 座標を表す `x [m]` および y 座標を表す `y [m]` が暗黙に宣言されている．したがって，エージェント `Goat` には `x`，`y`，`blood_sugar` の 3 つの属性が宣言されている．`Goat` の各インスタンスは，`x` 属性および `y` 属性として x 座標および y 座標をメートル単位で，`blood_sugar` 属性として血中に蓄えられたエネルギーを kcal と同次元の SI 単位であるジュール単位で格納する．

一般にマルチエージェントシミュレーションでの各エージェントの属性値の更新には，非同期的な更新を行うも

のと同期的な更新を行うものがある．非同期的な更新では，タスクの実行による属性値の更新は直ちに反映され，同一ステップでの他のエージェントあるいは他のタスクの実行に影響を与える．同期的な更新では，各エージェントタスクによる属性値の変更は即座には反映されず，シミュレーション時刻の 1 ステップ終了時に全インスタンスの全属性値が一斉に更新される．非同期的な更新は，手続き型言語をベースにしたシミュレーション環境や分散環境でのシミュレーション環境に多くみられる方式で，属性値更新のオーバーヘッドが少なく効率的に処理できる利点がある一方，エージェントおよびタスクの実行順序によってシミュレーション結果が影響されるという欠点を持つ．Re:Mobidyc では，〈 **R4：分析可能性，再現性** 〉を目的に，同期的更新を採用した．同期的更新と擬似乱数のシードの管理によって，Re:Mobidyc では任意のシミュレーション時刻での状態から次の状態の再現性を確保している．タスクを実行するインスタンスの順序やタスクの順序に関わらず，同じ属性に対する同じ値への更新が再現される．また，各シミュレーション時刻での全てのインスタンスの属性の値は上書きされずに記録される．したがって，Re:Mobidyc 上でのシミュレーションは，初期状態から開始したそれぞれのシミュレーション時刻における全属性値が記録され，かつ，再現されることから，シミュレーションモデル，初期状態，シミュレーション結果の間の整合性を各シミュレーション時刻ごとに検証することができる．

タスク定義

Re:Mobidyc では，〈 **R2：個体および環境の相互作用** 〉はタスクにより表現する．すなわち，各エージェントの移動，捕食，排泄，代謝，生殖などの行動は，タスクとして定義される．タスクは，タスクを実行するインスタンスを含む 1 つ以上のインスタンスの属性を変化させることで，エージェント間の相互作用を定義する．〈**R3：プログラミングスキル要求の軽減** 〉および〈 **R4：分析可能性，再現性** 〉のために，タスクによる属性値の変更は，各シミュレーション時刻での処理の最後に一括して同期的に行われる．各シミュレーション時刻において，それぞれの属性の値は一定である．

タスクの定義は，タスク名，相手エージェント，事前条件，属性更新定義のリスト，局所定義のリストからなる．ただし，相手エージェント，事前条件，局所定義は省略可能である．図 3 に，標準ライブラリに組み込まれているタスク `move` の定義のソースコードを示す．1 行目では，タスク名 `move` が宣言され，2 行目と 3 行目で，それぞれ属性値 `x` および `y` の更新が定義されている．4 行目は以降が局所定義であることを示し，5 行目と 6 行目でそれぞれ局所定数 `theta` および `r` が定義されている．

```
to move is
    my d/dt x' = cos(theta) * r
    my d/dt y' = sin(theta) * r
where
    theta = my heading
    r = my speed.
```

図 3. タスク定義の例

表 1. 属性値更新の左辺値の種類と構文

左辺値種別	構文
更新値	‘my’, 属性名, ‘ ’’ エージェント参照,‘ ’s’, 属性名, ‘ ’’
差分値	‘my delta’, 属性名, ‘ ’’ ‘my Δ’, 属性名, ‘” エージェント参照,‘ ’s delta’, 属性名, ‘ ’’ エージェント参照,‘ ’s Δ’, 属性名, ‘ ’’
微分値	‘my d/dt’, 属性名, ‘ ’’ エージェント参照,‘ ’s d/dt’, 属性名, ‘ ’’

属性値の更新定義には，(1) 値の直接代入，(2) 値の差分値，(3) 値の微分値の 3 種類の定義がある．表 1 に 3 種類の更新定義の左辺値をまとめる．いずれの形式でも，左辺値は属性値への参照と区別するために，末尾に *'* が付けられる．右辺値には，左辺値と互換性のある単位を持つ任意の数式を記述することができる．表 2 に主な数式を示す．

タスクによる属性値の変更は，直ちには反映されず，各シミュレーション時刻での処理の最後に一括して同期的に行われる．値の直接代入は，右辺を評価した結果得られた数値を次のシミュレーション時刻におけるそのインスタンスの属性の値になる．1 つのインスタンスの同一属性には，1 シミュレーション時刻につき 1 回しか直接代入できない．値の差分値による更新定義は，右辺を評価した結果得られた数値を，次のシミュレーション時刻におけるそのインスタンスの属性の値の増分とする．値の微分値による更新定義は右辺を評価した結果得られた数値とシミュレーション時刻の刻み幅の積を，次のシミュレーション時刻におけるそのインスタンスの属性の

表 2. 主な表現式の種類と構文

数式の種類	構文
リテラル	数値, '[', 単位, ']'
属性参照	'my', 属性名 エージェント参照, ' 's', 属性名
差分参照	'my delta', 属性名 'my Δ', 属性名 エージェント参照, ' 's' delta, 属性名 エージェント参照, ' 's Δ', 属性名
微分参照	'my d/dt', 属性名 エージェント参照, ' 's d/dt', 属性名
算術演算	数式 ('+' | '-' | '*' | '/' | '^') 数式
負数	'-', 数式
関数適用	関数名, '(', 数式, {',', 数式 }, ')'
場合分け	数式, 'if', 条件式, ',', { 数式, 'if', 条件式, ','}, 数式
擬似乱数	'uniform', リテラル, 'to', リテラル
方向	'direction', エージェント参照
属性勾配	'direction neighbor 's', Cell 属性名
距離	'distance', エージェント参照
時刻	'time'
刻み幅	(' Δ ' | 'delta'), 'time'

値の増分とする．差分値による更新定義および微分値による更新定義は，1 つのインスタンスの同一属性に 1 シミュレーション時刻につき複数回の更新定義することができる．シミュレーション時刻 t から $t+1$ に移行する時，インスタンス i の属性値 pi は以下の漸化式に従って更新される．

$$pi_{(t+1)} = pi'_{(t)} + \Sigma_{jk}(\Delta pi'_{jk_{(t)}}) + \Sigma_{jk}(d/dt\, pi'_{jk_{(t)}} * \Delta\ time)$$

ただし，$pi_{(t+1)}$ は時刻 $t+1$ におけるインスタンス i の属性 p の値である．$pi'_{(t)}$ は，時刻 t にインスタンス i の属性 p への直接代入が行われた場合にはその値，行われなかった場合には時刻 t における当該属性の値とする．また，$\Delta pi'_{jk_{(t)}}$ は時刻 t に実行されたインスタンス j のタスク k による差分更新の右辺値，$d/dtpi'_{jk_{(t)}}$) は時刻 t に実行されたインスタンス j のタスク k による微分更新の右辺値，$\Delta time$ はシミュレーション時刻の刻み幅とする．

キーワード `where` 以下では，局所定義を列挙する．局所定義の右辺値には，属性更新定義の右辺値と同様に，数値リテラルや属性への参照を使った数式を記述する．図 3 の記述では，`theta` および `r` の 2 つの局所定義がされ，それぞれ，タスクを実行しているインスタンスの `heading` 属性と `speed` 属性の値を取る．

置換によるタスクの適応

Re:Mobidyc では，〈**R3**: プログラミングスキル要求の軽減〉を目的として，ライブラリ化された抽象的なタスクを，構文的置換により特化させることで，具体的なタスクを定義する仕組みを実装した．図 2 の中で，エージェント `Goat` のタスクとして `move` が，以下の記述によって標準ライブラリの組み込みタスク `move` の特化として定義されている．

```
Goat move where
    my heading -> direction neighbor's grass
    my speed -> 1[km/day].
```

エージェント `Goat` のタスク `move` は，汎用の組み込みタスク `move` の `my heading` を `direction neighbor's grass` に，`my speed` を `1[km/day]` に置き換えたものとして定義される．すなわち，下記のタスク定義と等しい．

```
to move is
    my d/dt x' = cos(theta) * r
    my d/dt y' = sin(theta) * r
where
    theta = direction neighbor's grass
    r = 1[km/day].
```

ここで，`direction neighbor's grass` は Re:Mobidyc の組み込み命令で，実行するエージェントが位置している Cell インスタンスの周辺の，属性 `grass` の勾配の方向を返す．したがって，このタスクは，インスタンスを `Cell` の `grass` 属性の値がより大きな方向に向けて移動させる．移動量は，1 日かけて 1km 移動する速度であり，仮にシミュレーション時刻の刻み幅が 6 時間の場合，1 ステップあたり `250[m]` 移動させる．

置換によりタスクを特化させる時，特化によりタスク名をより具体的な名前が適切になることがある．Re:Mobidyc では，置換対象となるタスク名に続いて `namely` 句により新しいタスク名を宣言することができる．図 2 の `Goat` エージェントの定義では，`respire` タスクを以下のように `lose` タスクから特化させている．

```
Goat respire namely lose where
    my amount -> my blood_sugar
    my loss ->
```

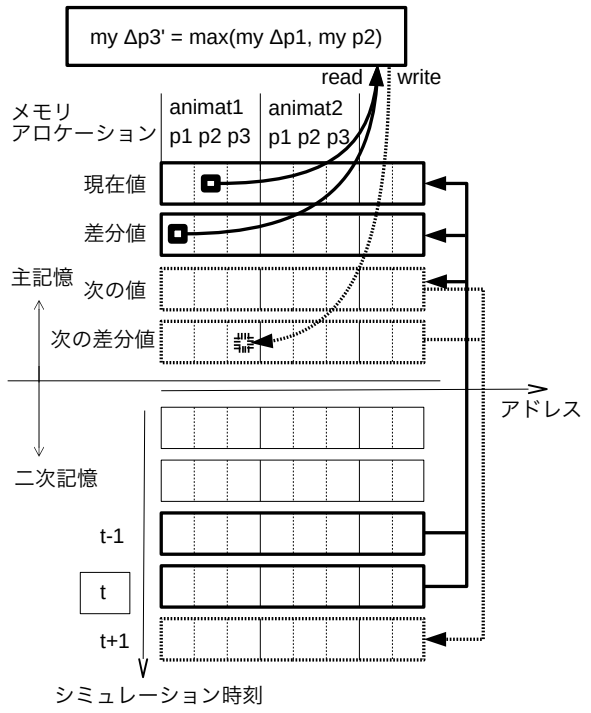

図 4. 時系列メモリモデルの概要

```
min(my amount, 2000 [kcal/day] * Δtime).
```

4.2. 時系列数値データに特化したメモリモデル

Cormas 等の汎用プログラミング言語をベースにした MAS 環境の多くは，ユーザが記録する属性を指定して，ファイル等にデータを出力する機能を持つ．それらの MAS 環境では，各インスタンスはヒープメモリ上にアロケートされ，シミュレーション時刻における属性値をメモリ上に持つ．Re:Mobidyc は，〈**R4：分析可能性，再現性**〉のために，ユーザが指定した属性だけでなく，初期状態からの各シミュレーション時刻における全ての属性値を記録する．全ての属性値の記録は Re:Mobidyc の設計の基盤である，時系列メモリモデルが行う．

図 4 に時系列メモリモデルの概要を示す．いくつかの，水平方向に区分けされた横長の矩形が並んでいる．これらの矩形はメモリイメージを表しており，複数のメモリイメージが 1 つのアドレス空間上にオーバーレイされる形で統合されている．このメモリモデルでのアドレス空間は，この横長の矩形を水平方向に区分けしたスロットの整数インデックスである．

中央の水平線より上の 4 つのメモリイメージは，主記憶上に保持された，現在実行中のシミュレーション時刻における各属性値の読み出しや更新のための領域である．最上段の現在値とラベルが付けられたメモリイメージは，現在シミュレーションしている時刻での各属性の値を格納している．2 段目の差分値とラベルが付けられたメモリイメージは，現在シミュレーションしている時刻での各属性の，前のシミュレーション時刻での値からの差分値を格納している．これら 2 つのメモリイメージは，インタプリタがタスクの属性更新定義の右辺値を計算するために利用され，インタプリタからは read-only である．例えば，このアドレス空間上に，2 つのインスタンス `animat1` と `animat2` について，それぞれ属性数分の連続領域がアロケートされ，それぞれ 3 つの属性 `p1`，`p2`，`p3` を持っていることを仮定する．ここで，インタプリタがインスタンス `animat1` のタスクとして属性地更新定義 `my Δp3′ = max(my Δp1, my p2)` を評価実行する時，まずは右辺値を計算する．`my Δp1` の評価実行では，上から 2 番目の差分値のメモリイメージから，インスタンス `animat1` の属性 `p1` がアロケートされている左から 1 番目のスロットの値を読み込む．続いて，`my p2` の評価実行では，上から 1 番目の現在値のメモリイメージから，インスタンス `animat1` の属性 `p2` がアロケートされている左から 2 番目のスロットの値を読み込む．2 つの値のうち，より大きな値が右辺値の値となる．

属性更新定義の左辺値への更新には，3 段目と 4 段目のメモリイメージが使われる．3 段目と 4 段目のメモリイメージはそれぞれ，直接値による属性更新定義での更新値と差分値および微分値による属性更新定義での更新値を保存するための領域である．例えば，インタプリタがインスタンス `animat1` のタスク中の `my Δp3′ = max(my Δp1, my p2)` を評価実行する場合，4 段目の次の差分値とラベル付けられたメモリイメージの，`animat1` の `p3` がアロケートされた左から 3 番目のスロットに，右辺値の評価結果の数値を加算して上書きする．

中央の水平線より下のメモリイメージは，二次記憶に確保された，各シミュレーション時刻での全属性値を保存するメモリイメージである．シミュレーション開始時には，まず，初期状態 ($t = 0$) でのメモリイメージが初期化される．各シミュレーション時刻 (`t`) の処理の開始時と終了時に，時系列メモリモデルはインタプリタを利

用せずに主記憶上の 4 つのメモリイメージの更新処理を行う．この更新処理は，インタプリタとは独立して，時系列メモリモデルが内部処理として行う．

シミュレーション時刻 (t) の処理の開始時の更新処理として，時系列メモリモデルは主記憶中の現在値，差分値，次の値，次の差分値の 4 つのメモリイメージを初期化する．初期化には，まずその時刻でのメモリイメージを現在値メモリイメージと次の値メモリイメージに転送し，前時刻でのメモリイメージとの差分を差分値メモリイメージの全スロットに書き込む．また，次の差分値メモリイメージの全スロットを 0 に初期化する．以上の初期化により，タスクの評価実行に必要な主記憶上のメモリイメージが準備され，インタプリタは各インスタンスでの各タスクを評価実行する．評価実行により，次の値メモリイメージと次の差分値メモリイメージが更新される．シミュレーション時刻 (t) の処理の終了時の更新処理として，時系列メモリモデルは，次の値メモリイメージの各スロットと，対応する次の差分値メモリイメージのスロットの和を，二次記憶上のシミュレーション時刻 $t+1$ のメモリイメージに格納する．

インタプリタは時系列メモリモデルを通して主記憶上の 4 つのメモリスロットに対してのみ読み出しと書き込みを行い，二次記憶上の各シミュレーション時刻でのメモリイメージを扱う必要はない．この機構により，Re:Mobidyc は〈**R5：スケーラビリティ**〉について，シミュレーション時刻に関するスケーラビリティを確保している．また，時系列メモリモデルはファイルシステムなど二次記憶の種類によって，インタプリタに対して互換性を確保した複数の実装が可能である．現在はメインメモリに保存するものとファイルシステム上に保存するものが実装されている．今後，DBMS などに向けた時系列メモリモデルを実装していく予定である．

5. 例題：牧草地と山羊

Re:Mobidyc は現在も開発が続けられているが，現時点で簡単なモデルを構築しシミュレーションを実行することができる．本節では，動物と環境の相互作用を表現した簡単な例として，牧草地と山羊のモデルを Re:Mobidyc 上で実行する例を紹介する．

牧草地と山羊は，エネルギーの供給と消費のバランスを再現するモデルである．牧草地は太陽からのエネルギーを，光合成により牧草の栄養価に変換して蓄積する．山羊は，牧草地の中をより豊かな牧草地へ移動しながら，牧草を食べて，牧草の栄養価を血中糖という形でエネルギーを獲得する．ただし，山羊はその代謝活動によりエネルギーを失う．すなわち，牧草地と山羊が持続可能なエネルギー系を構成するためには，太陽から牧草地に供給されたエネルギーと代謝がバランスする必要があり，その間のバッファとなるのが牧草の栄養価と血中糖である．

```
Cell with
        grass [kcal].

Cell grow_grass namely gain_upto where
    my thing -> here's grass
    my amount -> 2000 [kcal/day Δ]*time
    my maximum -> 1000000 [kcal].
```

図 5. Cell 定義の例

牧草地と山羊のモデルでは，2 種類のエージェントを定義する．牧草地の小区画を表す `Cell` と山羊の個体を表す `Goat` である．このモデルでの `Cell` の定義を図 5 に示す．`Goat` の定義は図 2 で示された通りである．

`Cell` のタスク `grow_grass` は，太陽から供給されたエネルギーを牧草の栄養価に変換する相互作用を表す．`grow_grass` は，当該区画の牧草の量を毎日 2000kcal のペースで最大 1000000kcal まで成長させる．このタスク定義でベースにしている組み込みタスク `grain_upto` は，インスタンスの 1 つの属性値を，上限値付きで一定量増加させるための抽象タスクであり，`my thing`，`my amount`，`my maximum` の 3 つの参照を具体化することで，牧草の栄養価を獲得する具体的なタスクを定義する．この `grow_grass` タスクを定義している GUI のスクリーンショットを図 6 に示す．タスク定義 GUI は上段と下段に分かれており，上段でタスクの原型となる既存の抽象タスクを選択し，下段でその抽象タスクに対する置換操作によって，特化したタスクを定義する．

`Goat` のタスクも同様にして定義することができる．特に，タスク `graze` は，`Cell` との相互作用によって，牧草の栄養価を山羊の血中糖に変換するタスクであるが，これは組み込みタスクである `collect` をベースにしている．`collect` は Animat が位置する `Cell` から，ある量を獲得し，`Cell` から同量を減少させる抽象タスクである．Re:Mobidyc では，同一インスタンス内での量の移動，インスタンス間での量の移動などを，抽象タスクとして定義して，標準ライブラリの中で提供している．

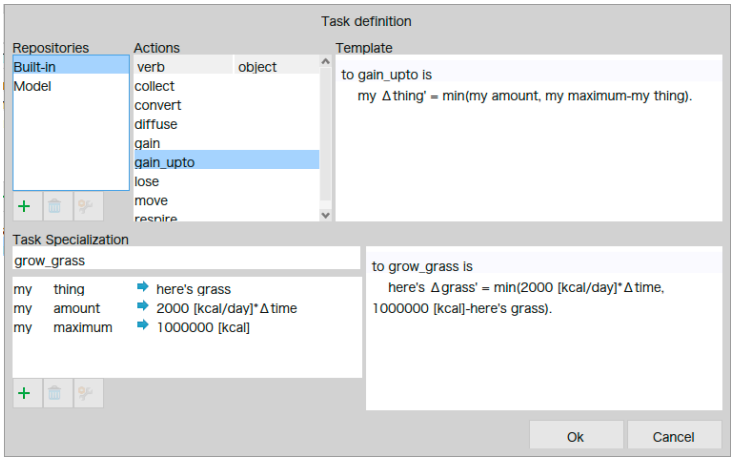

図 6. タスク定義 GUI のスクリーンショット

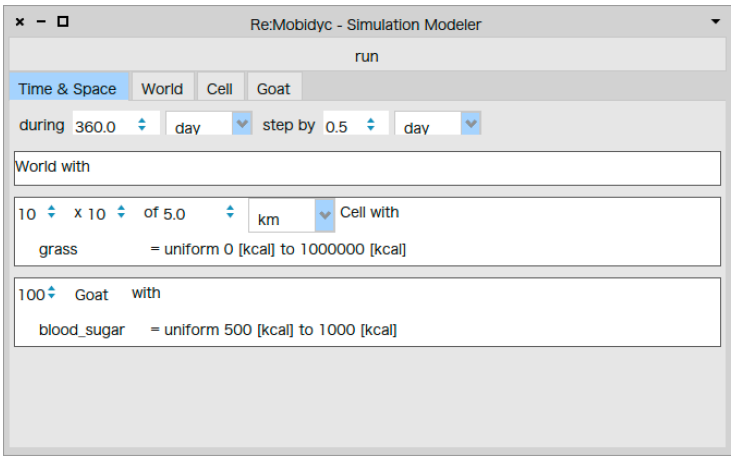

図 7. シミュレーション条件設定 GUI のスクリーンショット

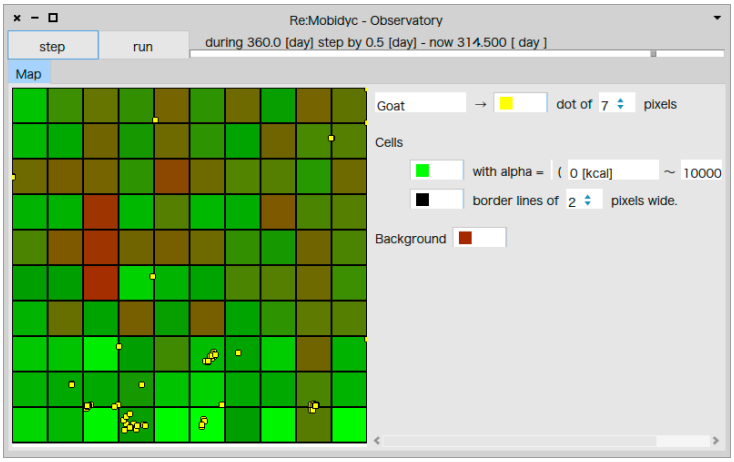

図 8. シミュレーション環境のスクリーンショット

ユーザーはこれらの基礎的な抽象タスクを利用して，さまざまなエージェント間の相互作用を表現することができる．

図 6 の上段左側の Repositories とあるリストには，Built-in および Model の 2 項目があり，組み込みタスクのレポジトリを表す Built-in が選択されている．上段中央の Actions には，左側で選択されたレポジトリに格納されているタスクの一覧が表示され，特化のベースとなる抽象タスクとして `gain_upto` が選択されている．その定義のソースが上段右側に表示される．下段では，左側で 3 つの置換が定義され，その置換を適用した結果が右側に表示される．

シミュレーションの期間やセル分割，Animat の初期化について設定する GUI のスクリーンショットを図 7 に示す．during で始まる行で，シミュレーション期間を単位付きで指定されている．ここでは単位として `day` が使われているが，`hour`，`second` など他の時間単位を選択することもできる．`Cell` は 5km 四方の矩形が 10*10 個の計 100 個設定され，その属性 `grass` の初期値を 0kcal から 1000000kcal までの一様分布から設定する．また，エージェント `Goat` のインスタンスを 100 個，初期状態として生成し，その属性 `blood_sugar` を 500kcal から 1000kcal の一様分布から設定する．最上段の run ボタンを押すと，図 8 にあるように時間経過とともにモデルの状況が可視化される．

6. まとめと今後の展望

筆者らは，個体群動態のシミュレーションを目的としたマルチエージェントシミュレーション環境 Re:Mobidyc について，個体群動態の科学的モデリングに求められる特性を起点として，モデリング言語および実行系を設計し実装してきた．特に，シミュレーション環境の基礎となるモデリング言語では，生活史段階を言語機能に取り入れることで，既存の汎用プログラミング言語をベースにしたシミュレーション環境では直接的な表現が困難なモデルを，ユーザである環境学や生物学の研究者がその研究領域の語彙でモデリングすることを可能にした．また，その実行系の基礎となるメモリモデルでは，時系列の数値データに特化したメモリモデルを設計し実装することで，シミュレーションの計算過程を検証可能な形で記録に残すことを実現した．

Re:Mobidyc の開発はまだ初期段階であり，これから追加実装すべき機能として，研究チームによるモデリングおよびシミュレーションを想定した，ネットワーク機能が挙げられる．1 つは，マルチユーザに対応したバージョン管理機能付きのモデルレポジトリである．シミュレーションモデル全体だけでなく，個別のエージェントの定義，抽象タスクの定義，シミュレーション結果など，研究チーム内での共有を可能にする機能が必要である．もう 1 つは，複数のシミュレーション実行をモニタリングするサーバである．LAN 上に Re:Mobidyc の実行状況を監視・表示するためのモニタリングサーバを設置し，複数のコンピュータ上で Re:Mobidyc が実行されている時，どのコンピュータ上でどのモデルのシミュレーションが実行されているかを把握できるようにすることで，研究チームのメンバーがコンピュータ資源を有効に利用できるようにする予定である．

参考文献

[1] Sameera Abar, et al. ”Agent Based Modelling and Simulation tools: A review of the state-of-art software”, *Computer Science Review*, vol 24, pp. 13-33, 2017

[2] Seth Tisue and Uri Wilensky. ”Netlogo: A simple environment for modeling complexity”, *Proceedings of the 5th International Conference on Complex Systems (ICCS’04)*, pp. 16-21, 2004.

[3] Vincent Ginot and Christophe Le Page. ”Mobidyc, a generic multi-agents simulator for modeling populations dynamic”, *Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial In Telligence and Expert Systems: Tasks and Methods in Applied Artificial Intelligence (IEA/AIE 1998)*, pp. 805-814, 1998.

[4] Pierre Bommel, et al. ”Cormas: an agent-based simulation platform for coupling human decisions with computerized dynamics”, *Simulation and gaming in the network society*, Translational Systems Sciences, vol 9, pp. 387-410, 2016.