Vol.2022-SE-210 No.30
2022/3/12

ソフトウェアプロセスのシミュレーションによる評価

宮永 照二[1,a)]

概要：並列プログラムのようなコンカレンシーをもつプログラムはその振る舞いを予測することが困難な場合がある。そのようなプログラムについて動態シミュレーションを可能にするツール GraphCore を開発している。本稿では、GraphCore の機能を紹介し、ソフトウェアプロセスを GraphCore を用いて並列プログラムとして模し、シミュレーションした結果から得られた知見を述べる。

キーワード：ソフトウェアプロセス, シミュレーション, 定量的評価

Software Process Evaluation Using Simulation

Shoji Miyanaga[1,a)]

Abstract: Concurrent Program is sometimes hard to predict its behavior. For this problem, I am developing simulation tool 'GraphCore', which enables dynamic simulation for concurrent program. In this paper, I introduce features of GraphCore and mention the knowledge induced from simulation results which are outputs of software process simulations using GraphCore.

Keywords: Software Process, Simulation, Quantitative Evaluation

1. はじめに

コンカレントプログラムは振る舞いが予測困難な場合がある。特にそのプログラムの構成要素の数や性質の違い、構成要素間の相互作用が多い場合、その傾向がある。そのような場合には、シミュレーションによる動態予測が有効と考えられる。なぜなら、シミュレーションの主要な目的のひとつは、考察の対象である複雑なシステムのメカニズムを明らかにすることにある [1] からである。実際に、複雑な対象の動態解析のためにシミュレーションが実施され、その性質が理解されてきている [2][3][4][14][17]。それと同時に、対象を適切にシミュレーションするための手法も開発されている [9][10][12][15][18][19]。

その一方で、ソフトウェア全般に関するシミュレーションツールはあまり数は多くない。これはソフトウェアはそれ自体が極めて複雑で、物理化学的な作用をする対象と比べても、その振る舞いはアルゴリズムとして記述されるため、個別の振る舞いのパターンが多くなってしまうことが理由の 1 つと考えている。

シミュレーションのためのモデル化には対象の抽象化が必要であるが、抽象度をあげすぎると他の性質を失うだけでなく、シミュレーションの精度も低下するという欠点がある。さらに、シミュレーションを設計に用いる場合、設計最適化のための試行錯誤が発生するのが通常であり、そのオペレーションは煩雑となる。

このような問題に対応するため、シミュレーションツール GraphCore[*1] を開発している。

本稿では、次章以降、以下の構成で論じる。2 章で、GraphCore について概説する。3 章で、アジャイル型とウォーターフォール型の 2 つのソフトウェアプロセスを GraphCore を用いてシミュレーションを行う。4 章で、シミュレーション結果およびシミュレーションを実行するというオペレーションについて考察し得られた知見と、今後の展望を述べる。

[1] 株式会社ディア
[a)] myngshj3@gmail.com

[*1] https://github.com/myngshj3/graphcore/

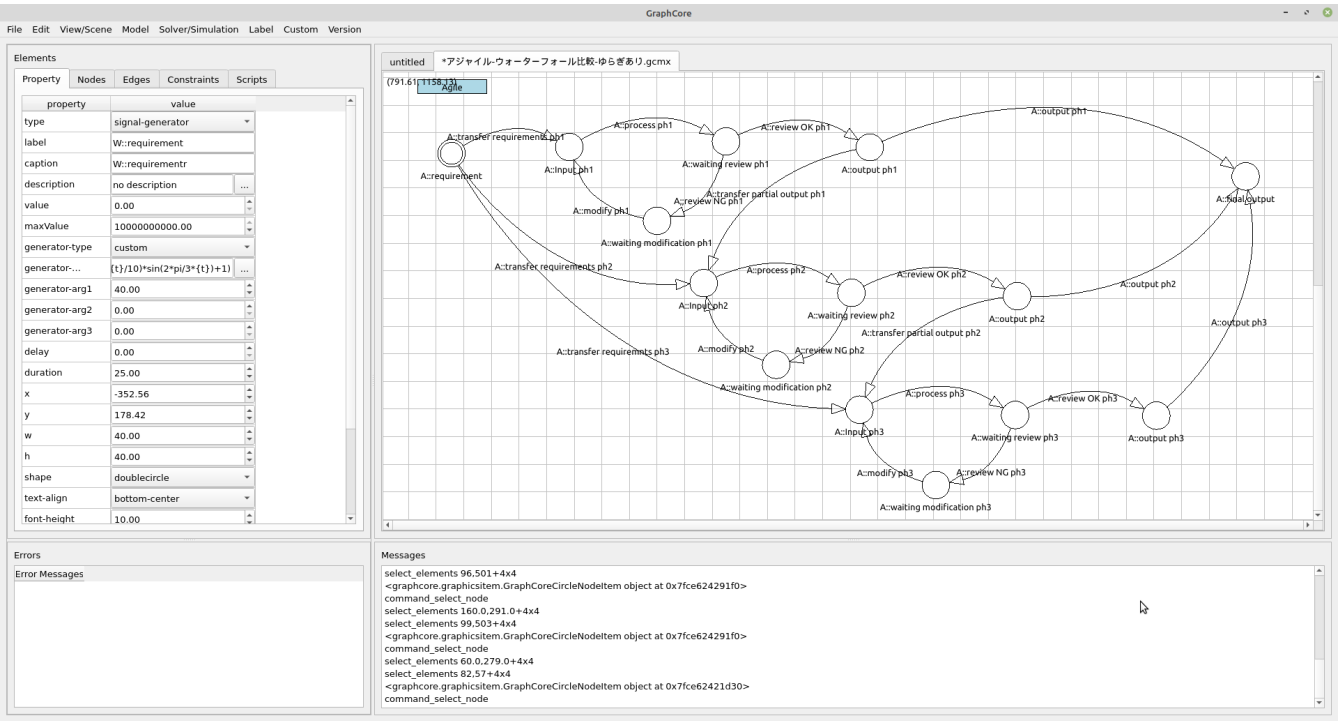

図 1 GraphCore Overview

2. GraphCore

2.1 Biography

GraphCore は、汎用の分散環境シミュレータとして 2019 年に計画され、APRIS2019[*2] と FOSE2019[*3] でその草案が発表された [5][6]。2020 年に現バージョンの開発を開始し、現在、評価版として GitHub で公開している。

2.2 スクリーンショット

図 1 に GraphCore のメイン画面全体のスクリーンショットを示す。

メイン画面は 4 つのセクションに分かれており、右上のペインにノードとエッジでモデルを入力する。左上のペインでは右上のペインの各要素について、そのプロパティを編集したり、入力されたモデルへの制約を入力したり、カスタムメニューで実行可能なスクリプトを編集したりする。左下のペインはモデルのエラーを表示する。右下は一般的なメッセージ表示領域である。

2.3 主な機能

主な機能は以下のようになる。

(1) プリプロセッサー：シミュレーション・モデルの作成
(2) ソルバー：モデルの属性に応じたシミュレーションの実行
(3) ポストプロセッサー：シミュレーション結果の可視化
(4) スクリプティング：スクリプトによる独自処理の実装と自動化
(5) カスタマイゼーション：アプリケーションの振る舞いのカスタマイズ

2.3.1 プリプロセッサー

プリプロセッサーはシミュレーション対象について、グラフによるモデリングを行う。必要な属性をノード、エッジ、グラフに持たせる設計となっている。

2.3.2 ソルバー

ソルバーはモデルの属性に応じてその振る舞いをシミュレーションし、その結果を出力する。

GraphCore は現在、情報の流れを模す 'basic-flow' ソルバーを内臓している。任意のノード i における情報量の計算方法は以下の式であらわされる。

$$x^i_{t+\Delta t} \leftarrow x^i_t + \sum_{s \in S^i} v^{(s,i)}_t \Delta t - \sum_{d \in D^i} v^{(i,d)}_t \Delta t + g^i_t$$

ここで、i は情報量を計算したいノード、S^i はノード i を終端ノードとするエッジの開始ノードの集合、D^i はノード i を開始ノードとするエッジの終端ノードの集合、g^i_t はノード i において時刻 t で生成される情報量、$v^{(s,i)}_t$ はエッジ (s,i) の時刻 t における速度、$v^{(i,d)}_t$ はエッジ (i,d) の時刻 t における速度である。

2.3.3 ポストプロセッサー

ポストプロセッサーはシミュレーション結果を可視化する。'basic-flow' ソルバーは情報の流れに着目しているため、情報の流速や蓄積量でタスクの処理速度の時間変化、

*2 https://www.sigemb.jp/APRIS/2019
*3 https://sites.google.com/site/jssstfose19/

インプットやバッファやアウトプットの蓄積量の時間変化を可視化することができる。

2.3.4 スクリプティング

GraphCore は独自のスクリプト言語を搭載している。スクリプトはモデルを操作するのみならず、ソルバープロセスを制御したり、ポストデータ（解析結果）を操作することができる。

GraphCore が提供するスクリプト言語は、アプリケーションの GUI からのモデルの操作と同様の操作をスクリプトにより可能にする。さらにソルバーの操作とポストデータ（解析結果）の操作をサポートするため、パラメータのチューニングとソルバーの実行とポストデータの解析をアルゴリズム化し自動実行することにより、これまでの解析系ソフトウェアでたびたび発生していた試行錯誤による解析のオペレーションを強力に支援する。

GraphCore ではスクリプトを実行させる方法はいくつか存在するが、図 2 は、その 1 つであるコンソール画面を示している。この例では、複数のシミュレーションを連続で実行する例を示している。

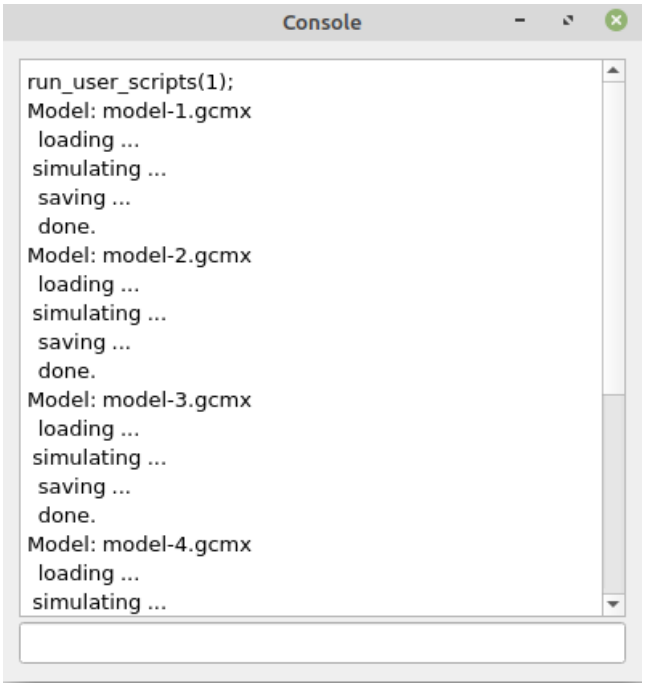

図 2 Console

2.3.5 カスタマイゼーション

GraphCore は汎用のシミュレーションツールを目指して開発されている。そのため、特定のアプリケーションの集合体としてではなく、アプリケーション自体が設定ファイル・構成ファイルで振る舞いを変化させるように設計されている。例えば、プリプロセッサーにおいては、ノードをタイプに応じてアイコン化したり、構成ごとに異なるプロパティをノードに持たせたり、アプリケーション独自のメニューを設定して実行するようなことが可能となっている。

2.4 他のシミュレーションソフトウェア

GraphCore を開発するにあたり、他のシミュレーションソフトウェアについて吟味した。それについて、述べることにする。まず、汎用的なシミュレーションツールとして思い浮かぶ 1 つは、SimuLink[*4]ではないだろうか。SimuLink は MATLAB[*5]と併用され、強力な機能を提供する。そのモデリング方法はマウスを用いてアイコン化されたノード要素を入力する。このノードはプロパティを持ち、ユーザはそれを編集する。複数のノード間を有効エッジで接続することにより、情報や信号の流れを模すことができる。その流れは一方向にとどまらず、フィードバックループを指定することもできる。このあたりの機能は GraphCore でも同様のオペレーションによるモデリング方法を採用している。一方、個別アプリケーションを提供する必要性を鑑みた場合、SimuLink はアプリケーションを構成する機能を提供しているものの、やや煩雑である。GraphCore のアプリケーションカスタマイズ機能は極めてシンプルであり、この点においては SimuLink より優位であると考えている。

SimuLink と似たシミュレーションソフトウェアとして、SciLab[*6]がある。SimuLink が有償の商用ソフトウェアであるのに対して、SciLab は SimuLink と同等の機能を提供する亜流のフリーソフトウェアとして開発された。そのため、使い勝手は SimuLink と同等である。

前述の 2 つとは趣向が異なるが、動的な振る舞いを検証するソフトウェアとして UPPAAL[*7]がある。UPPAAL は時間オートマトンを模し、タイミングに対してシビアな設計が求められるソフトウェアやシステムの検証に用いられる。トップエスイー [7] でも先進的な検証技法としてその用途を教えている。

3. GraphCore によるソフトウェアプロセス・シミュレーション

3.1 ソフトウェアプロセス・シミュレーションの目的

本章では、ソフトウェアプロセスをシミュレーションするが、その目的は以下のものがあげられる。

(1) シミュレーションモデルの妥当性の検証
(2) ソフトウェアプロセスの性質の定量的可視化
(3) シミュレーションのオペレーションの性質の検証

シミュレーションモデルの妥当性の検証として、GraphCore は現在 ‘basic-flow’ という情報の流れに着目したシミュレーションをサポートしているが、このシミュレーション方式によりソフトウェアプロセスが適切にモデル化できるかどうかを検証する。

*4 https://jp.mathworks.com/products/simulink.html
*5 https://jp.mathworks.com/products/matlab.html
*6 https://www.scilab.org/
*7 https://uppaal.org/

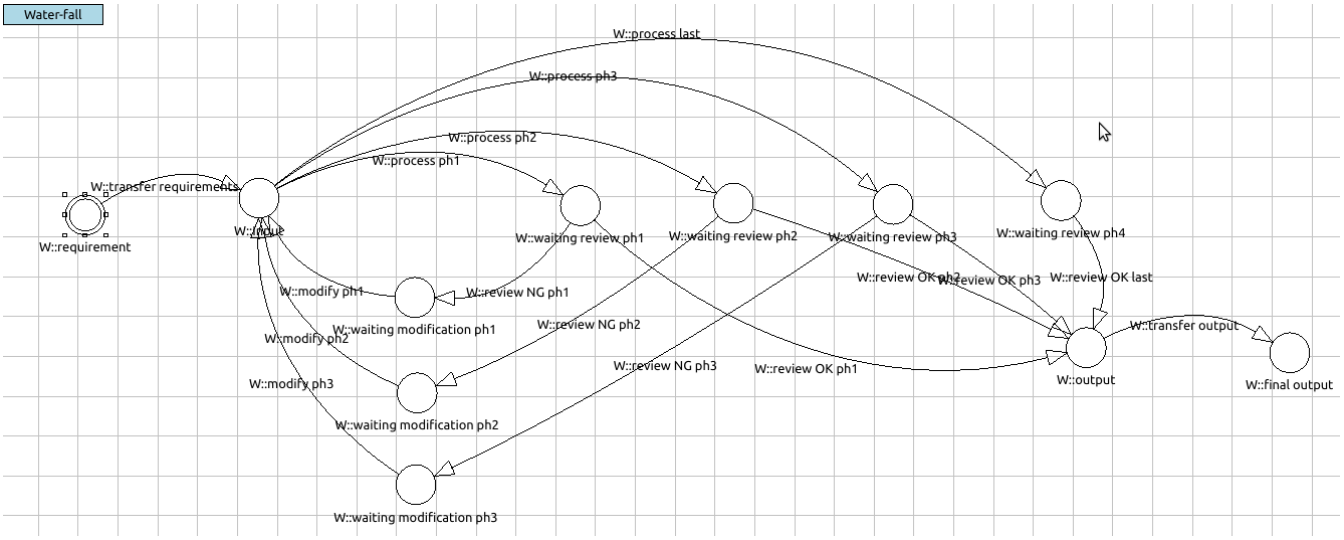

図 3 Water-fall Model Example

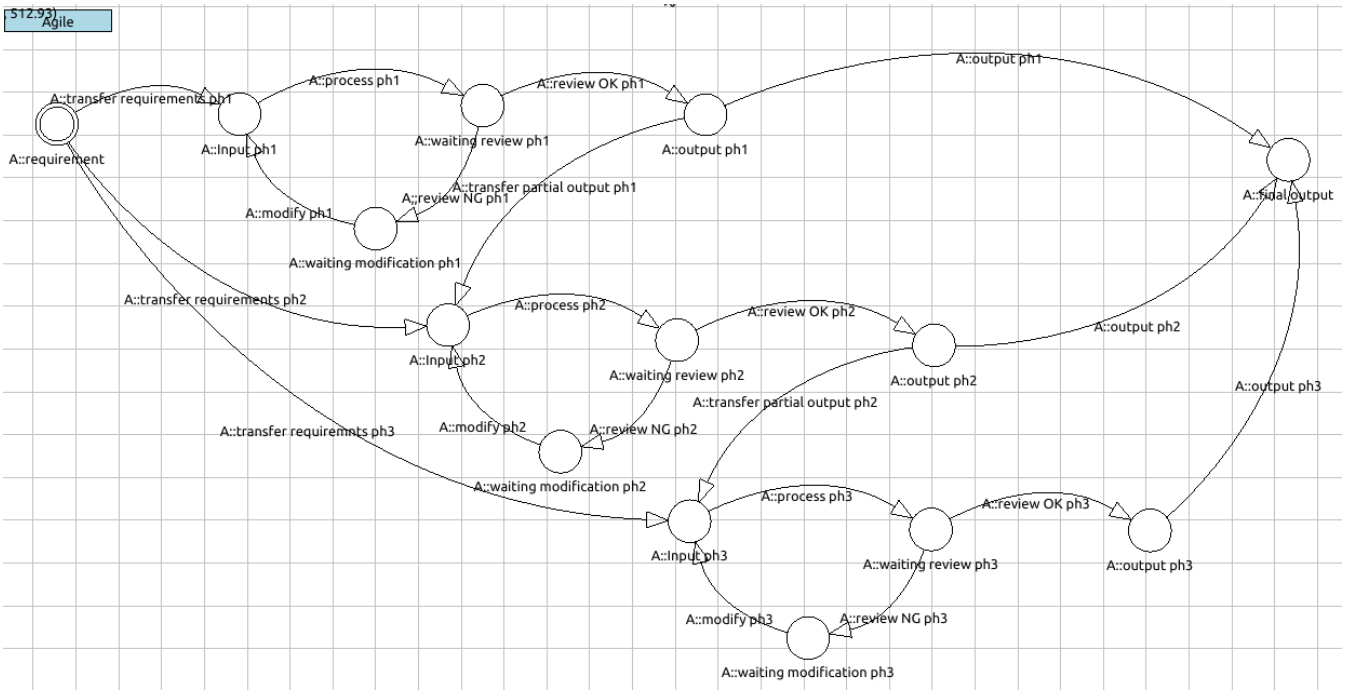

図 4 Agile Model Example

ソフトウェアプロセスの性質の定量的可視化として、アジャイル型とウォーターフォール型の 2 つの主なソフトウェア開発モデルについてシミュレーションし、それらの性質を定量的に検証する。自然言語で定性的に述べられることのある性質や直感的に理解されるが定量的に述べられるのが困難な性質について、それらを定量的に示す。

シミュレーションのオペレーションの性質の検証として、ソフトウェアプロセス・シミュレーションを実行する際にどのようなオペレーションが発生するかを検証し、シミュレーションツールに必要な要件を分析する。

3.2 ソフトウェアプロセスのモデル化

図 3 と図 4 にウォータフォール型とアジャイル型のプロセスについて GraphCore でモデリングしたようすをそれぞれ示す。

これらのシミュレーション・モデルは、図 5 に示すスケジュールをモデル化した。図 5 の上段がウォータフォール型のスケジュール、下段がアジャイル型のスケジュールを示している。

ウォーターフォール型のスケジュールは基本的に直列にタスクが実行されるスケジュールであるが、現実はレビューによる差し戻し等があり、シミュレーション・モデルにはそれを考慮している。

一方、アジャイル型では反復処理による継続的な成果のリリースを行うため、ループが存在する。さらに、単一の

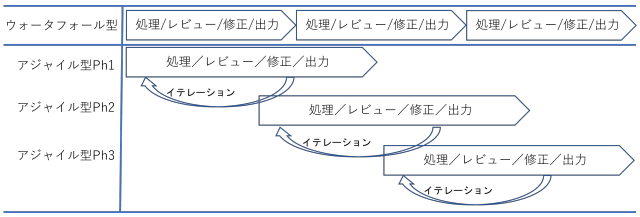

図 5 Scheduling Example

表 1 Water-fall task schedules

フェーズ	タスク	消化速度	実行区間
1	処理	25	[0, 10]
1	レビュー	25	[10, 13]
1	見直し	25	[13, 16]
2	処理	25	[16, 26]
2	レビュー	25	[26,29]
2	見直し	25	[29, 32]
3	処理	25	[32, 42]
3	レビュー	25	[42, 45]
3	見直し	25	[45, 48]
4	処理	25	[48, 58]
4	レビュー	25	[58, 66]

表 2 Agile task schedules

フェーズ	タスク	消化速度	実行区間
1	処理	12.5	[0, 32]
1	レビュー	12.5	[0, 29]
1	見直し	12.5	[0, 32]
2	処理	12.5	[16, 48]
2	レビュー	12.5	[16,45]
2	見直し	12.5	[16, 48]
3	処理	12.5	[32, 64]
3	レビュー	12.5	[32, 61]
3	見直し	12.5	[32, 64]

アジャイル型プロセスにとどまらず、先行タスクと後続タスクの関係とフェーズがオーバーラップして実行されることを加味して、先行フェーズのアウトプットの一部が後続フェーズのインプットとなる様子をモデル化している。

表 1 と表 2 に、図 5 に示すスケジュールのより詳細なタスクの処理速度と処理時間のスパンを、ウォーターフォール型とアジャイル型についてそれぞれ示す。

本シミュレーションでは、第一段階として要求が発生するようすをモデル化する。プロセスの開始時点ですでに要求のすべてが与えられていることは稀で、徐々に要求が与えらえれて、最終的に 100 %与えられるようすをモデリングするためである。この場合、要求ノードに情報がすでに存在するか、いずれかのノードで情報が生成される必要がある。ノードで情報が生成されるようにモデル化する場合、生成式 (generator-equation) を指定することにより、任意の式で表される情報生成パターンを設定可能である。ここでは、時間 t のとき $e^{-(\frac{t}{10})^2}sin(t) + 1.5sin(\frac{2\pi}{4}t)$ で表される情報量が生成されるように設定するために、生成式に '84.5*exp(-({t}/10)**2)+1.5*sin(2*pi/4*{t})' を設定し、入力が与えられる区間を $[0, 25]$ であたえるために、delay 属性と duration 属性にそれぞれ 0, 25 を設定した。これらを設定したノードでは、情報の生成速度は振動しながら次第に 0 に近づき、生成される情報量はおよそ 855 となる。

情報は、情報が存在するノードから他のノードへエッジを経由して輸送される。エッジは速度 (velocity) 属性を持ち、最大速度 (maxVelocity) 属性を閾値としてそれを超えない速さで情報を伝達する。エッジについては速度に関連して振幅 (amplitude) を設定できる。振幅に式を指定したエッジでは、その式であらわされる速度で情報を転送しようとする。これにより、能力習熟度のように処理の初期段階から時間が経つにつれて処理速度が向上し、あるところで安定するような振る舞いであったり、処理の初期段階では大きく揺らぐが、時間がたつにつれて揺らぎが収束するような振る舞いをモデル化することができるが、本シミュレーションにおいては、エッジの処理速度に式を与えていない。

3.3 ソルバーの実行

前項で入力したモデルに対して解析を実行する必要があるが、それはシミュレーションメニューからソルバーを起動することにより実行する。図 6 に、ソルバーの実行中のイメージを示す。

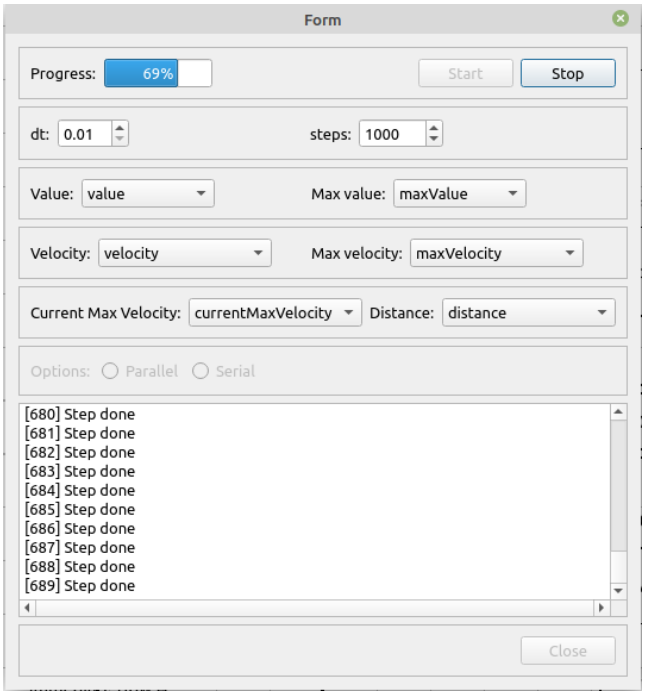

図 6 Solver Execution

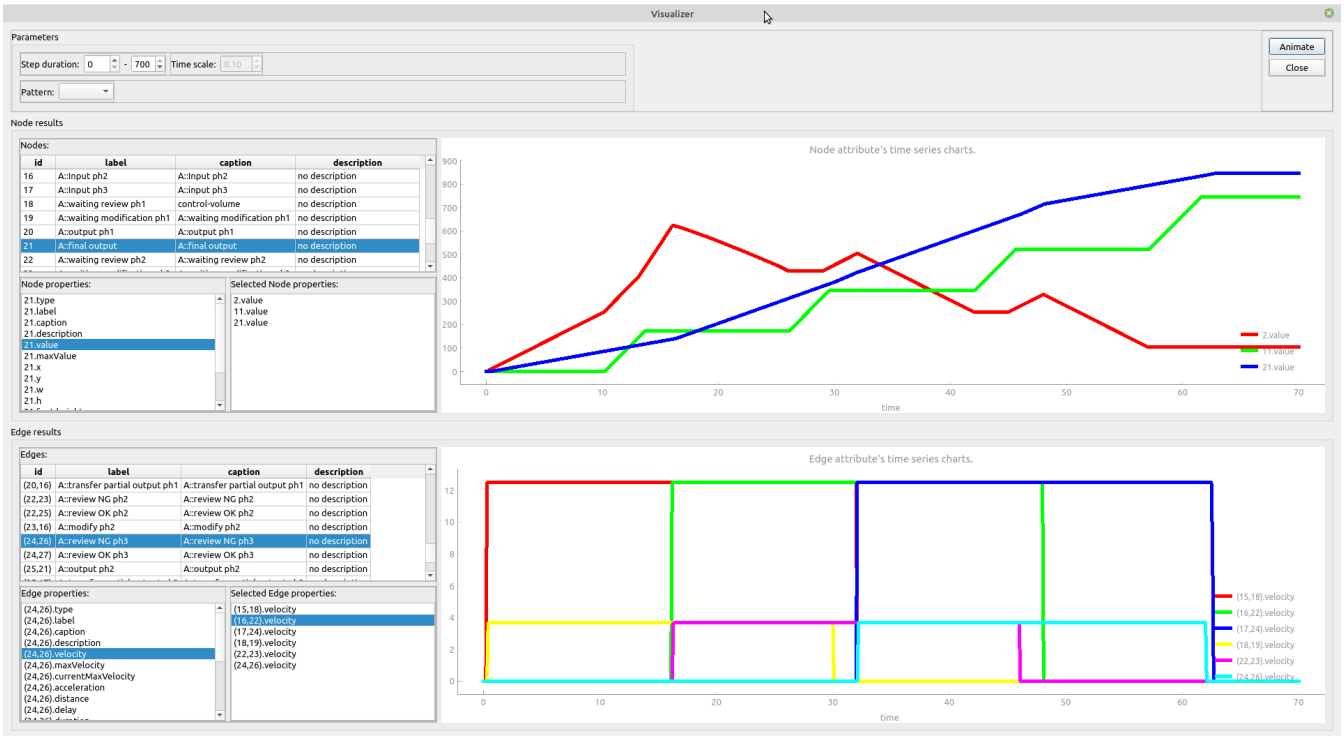

図 7 Visualizer

3.4 ポストプロセッサー

ソルバーの実行によりモデルの振る舞いをシミュレーションしたのち、その結果を可視化し吟味する必要がある。吟味した結果として問題がなければ、そのモデルをソフトウェアプロセスとして採用すればよいし、別のモデルを解析し比較したいという要求があれば、引き続きシミュレーションを実行することになる。

ここで、前述のモデルは 1 つの入力モデルにウォーターフォール型のモデルとアジャイル型のモデルを並べて入力しているが、これはポストプロセスにおいて、両者の振る舞いのシミュレーション結果を同時に可視化して比較検討するためである。

図 7 に、ポストプロセッサーによるシミュレーション結果の可視化のようすを示す。

この図の上段のチャートはノードにおける情報の蓄積量の時間変化を表している。具体的には、赤のラインはウォーターフォール型のモデルの入力に該当するノードの情報量の時間変化を表している。緑のラインはウォーターフォール型の最終成果物に該当するノードの情報量の時間変化を表し、青のラインはアジャイル型の最終成果物に該当するノードの情報量の時間変化を表している。ウォータフォール型の入力は最後までバックログ (積み残し) として残っている。その結果、最終成果物は 100%に到達していない。また、ウォーターフォール型の最終成果物への出力は階段状になるのに対し、アジャイル型の最終成果物への出力は単調増加を示している。これはウォーターフォール型とアジャイル型のそれぞれの性質を定量的に明確に示している。その他では、アジャイル型のほうはウォーターフォール型よりも収束がはやく、徐々に勾配を小さくしながら 100%に到達する様子が観察される。これはアジャイル型のモデルの特徴である。

また、図 7 の下段のチャートはエッジにおける情報処理速度の時間変化を表している。具体的には、入力に対する処理・レビュー・見直しのサイクルにおいて、どのような処理速度でこれらが実行されているかを表している。またこの処理速度の時間積分は処理される情報量に該当する。したがって、ソフトウェアプロセスにおける作業工数がどのようにあらわれるかを定量的に予測することができる。

4. 考察

本章では、GraphCore を用いてアジャイル型とウォーターフォール型の 2 つのソフトウェアプロセスを模し動態シミュレーションを行った結果に基づいて得られた知見と、シミュレーションというオペレーション全体を通して得られた知見について考察したのち、今後の研究開発について展望する。

4.1 シミュレーション・モデルの妥当性

前述したように、本稿で示すシミュレーションの 1 つの目的は、GraphCore が提供する ‘basic-flow’ シミュレーションにより、ソフトウェアプロセスが適切にモデル化されるかどうかを検証することにあった。

前章で示したシミュレーションにより、以下の 3 つの性質が可視化され、明らかに示された。

- ウォーターフォール型のアウトプットの動態
- アジャイル型のアウトプットの動態
- 2 つの型における性質の違い

‘ウォーターフォール型のアウトプットの動態’ については、ウォーターフォール型の特徴である ‘段階的にアウトプットが出力される’ ようすが観察された。‘アジャイル型のアウトプットの動態’ については、アジャイル型の特徴である ‘継続的にアウトプットが出力される’ ようすが観察された。‘2 つの型における性質の違い’ については、同量のモデルに対する入力、同量のタスク処理速度を設定した場合に、アジャイル型がウォータフォール型に優るようすが観察された。

以上の結果から、情報の流れに着目した ‘basic-flow’ シミュレーションのモデル化は、ソフトウェアプロセスのモデリングに妥当であるといえる。

4.2 ソフトウェアプロセスの可視化

前項の考察では、アジャイル型とウォータフォール型のモデルについて、ソフトウェアプロセスの性質をモデル化できることを示したが、前章のシミュレーション結果からは、さらに以下の性質が観察された。

- アジャイルは収束がはやい
- アジャイルはバックログの積み残しリスクが少ない
- アジャイルはノイズに対して安定である

これらの性質は、アジャイルの性質を示す文脈において定量的でない表現で言い表されることが多い。例えば、スクラムやスプリントというタームが登場する文脈における「チームワークを密にすることにより迅速で継続的なリリースがはかれる」というような表現がそれにあたる。今回のシミュレーションによりこれらの性質が定量的に示された。

入力に揺らぎや遅れを発生させるケースでは、ウォータフォール型のようにきっちりとタスクスケジュールを決めるプロセスでは、遅れてきた入力がバックログとして残ってしまう現象が観察された。これはウォータフォール型のスケジューリングの性質から至極当然の結果である。一方で、時間当たりのタスクの処理能力を軽減した上で長いスパンの反復プロセスを実行するアジャイル型では、バックログの積み残しが観察されなかった。これは、ウォータフォール型より長い時間スパンのタスクを設定することにより、処理すべき情報の到達の遅れを吸収していることを示している。

さらに、ノイズなどで入力される情報量が若干増えた場合、ウォータフォール型はそれがダイレクトにアウトプットに影響するのに対して、アジャイル型は反復プロセスがそれを吸収し安定動作することが観察され、アジャイル型はウォータフォール型と比較してロバストなプロセスといえる。

4.3 シミュレーションのオペレーション

シミュレーションの用途の 1 つに、設計の最適化があげられる。つまり、ある対象が設定されたゴールに到達するための設計は複数存在するが、その中から最適な 1 つを一意に決定しづらい場合に、パラメータを変更しながらシミュレーションを繰り返す必要がある。

GraphCore はスクリプト言語を搭載しソルバーを自動実行できるため、繰り返しのプロセスを支援できるが、パラメータの変更パターンについては、プロセスの設計者に依存する。つまり、最適化を支援するには、そのためのパラメータの指定パターンを形式化する必要があるが、現在はそれを実現するまでには到達していない。

しかし、スクリプティングによる自動化は強力なため、試行錯誤による反復作業が軽減されることは、現段階でも期待できる。

4.4 展望

GraphCore の ‘basic-flow’ シミュレーションにより、ソフトウェアプロセスのシミュレーションに適用でき、その動態の性質が予見できることを示した。その一方で、シミュレーションの用途がプロセス設計の最適化である場合、最適化のプロセスを支援するための仕組みが必要であることが示された。

シミュレーションの最適化問題への適用という観点においては、先行研究がある [8][9][10][11][13][15][16]。

いずれの場合も、最適化の支援にはまず最適化のアルゴリズムの理解が必要になる。現段階では、GraphCore はスクリプティングによる自動化をサポートしているものの、次はどのようなパラメータでシミュレーションを実行するのがよいかなどの意思決定を支援できない。最適化のための目的関数に何を設定するか、ということを決定することが重要である。最適化の目的は利用者により変わるため、どのような目的関数を設定する必要があるかの分析が必要である。また、目的関数が設定された場合にも、目的関数を最小化するためのパラメータの決定方法についてアルゴリズム化する必要がある。この点については、自動化によって得られる大量のシミュレーション結果から最適なパラメータの組み合わせを学習する方法が考えられ、シミュレーションツールとしての先行事例もある [16]。

さらに、ソフトウェアプロセスの設計のような場合には、資源の配置計画は、資源数が固定ではなく、その個数を増減させて試したりする必要性がある。このような資源の数の増減をスクリプティングで実現することは可能だが、その場合は実行可能な場合の数が膨大になるという問題がある。これについてもアルゴリズムの開発が必要になるだろう。

謝辞 本稿を執筆するにあたり実施した、GraphCore の

開発を含む研究開発において、九州大学大学院情報科学研究科 久住憲嗣准教授 (現芝浦工業大学准教授) には貴重な意見をご教示をいただいた。この場を借りて感謝申し上げたい。

参考文献

[1] 逆瀬川 浩孝: シミュレーションにおける最適化の手法, 計測と制御 (1991 年 2 月)
[2] Koji Ikago: Real-Time High-Brid Simulation, Information Processing Society Japan (Mar. 2016).
[3] Tatsuya Onoguchi, Ayane Hayashi, Katsuyuki Udaka, Yuichi Matsushima, Keiji Kimura, Horonori Kasahara: Hierarchical Interconnection Network Extention of Gem5 Simulator for Large Scale System, IPSI SIG Technical Report (Mar. 2017).
[4] Kazuki Uehara, Yuhei Akamine, Naruaki Toma, Moeko Nerome, Satoshi Endo: Evaluation of Hierarchical Collaborative Traffic System Using Micro Traffic Simulation, Information Processing Society Japan (Mar. 2016).
[5] Shoji Miyanaga: Simulation Environment for IoT System Quality, APRIS2019(Nov. 2019) http://www.sigemb.jp/APRIS/2019/.
[6] 宮永 照二: ソフトウェア工学ツールとしての総合シミュレータ, ソフトウェア工学の基礎 (2019).
[7] 本位田 真一, 粂野 史洋, 田原 康之, 鷲崎 弘宣: トップエスイー：サイエンスによる知的ものづくり教育, 情報処理学会 (2007).
[8] 坂本 茂: シミュレーションと最適化技術：総論, 日本 AEM 学会誌 Vol.22,No.1(20144)
[9] 朝日 大地, 井上知洋, 筒井章博: 誘導スケジューリングによる集団状態最適化手法のシミュレーション評価, 情報処理学会研究報告 (2015)
[10] 馬場 美也子, 北岡 広宣, 棚橋 巌: GA を用いた経路最適化による広域交通流シミュレータ上での交通状況再現手法, 情報処理学会死 (Dec.2002)
[11] 森戸 晋, 久保 幹雄: 数理計画とシミュレーション、日本オペレーションズ・リサーチ学会 1998 年 2 月号
[12] 小菅 崇裕: シミュレータ・モデリング・最適化手法を統合したモータ汎用最適設計システム, 首都大学東京大学院修士論文 (平成 23 年度)
[13] 阿瀬 始, 勘定 義弘: シミュレーション技術によるマルチロボット経路最適化, NKK 技報 No.177 (2002.6)
[14] 野田 五十樹, 篠田 孝祐, 太田 正幸, 中島 秀之: シミュレーションによるデマンドバス利便性の評価, 情報処理学会論文誌 Vol.49 No.1 (Jan. 2008)
[15] 高屋 圭介, 枇々木 規雄: モンテカルロ・シミュレーションを用いた動的ポートフォリオ最適化モデル, 日本オペレーションズ・リサーチ学会和文論文誌 (2012)
[16] 嶋田 佳明: 汎用シミュレーションシステムの紹介, オペレーションズ・リサーチ (2020 年 4 月号)
[17] 棚橋 巌, 北岡 広宣, 馬場 美也子, 森 博子, 寺田 重雄, 寺本 英二: 広域交通流シミュレータ NETSTREAM, 情報処理学会研究報告高度交通システム (ITS) (2002)
[18] 大鑄 史男, 小野木 基裕: セルオートマトン法による非難流動のシミュレーション, 日本オペレーションズ・リサーチ学会和文論文誌 (2008 年 51 巻)
[19] 高田 祥三: 製品ライフサイクルのシミュレーション, 計測と制御 (2004 年 5 月号)