AgentSimulator: データ駆動型ビジネスプロセスシミュレーションのためのエージェントベースアプローチ

著者：Lukas Kirchdorfer, Robert Blumel, Timotheus Kampik, Han van der Aa and Heiner Stuckenschmidt

翻訳者：堀志織

この論文では、ビジネスプロセスシミュレーション（BPS）の精度と効率性を向上させるため、AgentSimulator という新しいエージェントベースの手法を提案しています。従来のコントロールフロー中心のアプローチでは、リソース固有の行動や分散型意思決定を正確にモデル化することが困難でしたが、AgentSimulator では、リソースをエージェントとしてモデル化することでこれを解決しています。イベントログを基に発見されるエージェントは、それぞれのスケジュールや能力、行動パターンを持ち、現実に即したリソース間の相互作用を再現します。この手法は、中央集権型と分散型のプロセスの両方に対応可能で、従来手法と比較して大幅に短い計算時間で高いシミュレーション精度を達成しました。実験では、提案手法がコントロールフローや時間分布、サイクルタイムといった主要指標で既存手法を上回る結果を示しました。一方で、マルチタスクやバッチ処理といった複雑な行動のモデル化はまだ不十分であり、今後の研究課題として挙げられています。

AgentSimulator: データ駆動型ビジネスプロセスシミュレーションのためのエージェントベースアプローチ

Lukas Kirchdorfer, Robert Blumel, Timotheus Kampik, Han van der Aa and Heiner Stuckenschmidt

（概要）ビジネスプロセスシミュレーション(BPS)は、様々なシナリオにおけるプロセス性能を推定するための汎用的な手法である。従来、BPS アプローチは、プロセスモデルをシミュレーションパラメータで充実させることで、制御フローファーストの観点を採用していた。このようなアプローチは、ワークフローシステムでサポートされるような、中央で編成されたプロセスの挙動を模倣することができるが、現在の制御フロー優先のアプローチでは、明確なリソース動作と分散化された意思決定を伴う実世界のプロセスのダイナミクスを忠実に捉えることができない。この問題を認識し、本論文では、イベントログからマルチエージェントシステムを発見し、異なるリソース動作と相互作用パターンをモデル化して、基礎となるプロセスをシミュレートするリソースファースト BPS アプローチである AgentSimulator を紹介する。我々の実験によると、AgentSimulator は、異なるタイプのプロセス実行シナリオに対して高い解釈可能性と適応性を提供しながら、既存のアプローチよりも大幅に低い計算時間で最先端のシミュレーション精度を達成する。

（キーワード）ビジネスプロセスシミュレーション、マルチエージェントシステム、プロセスマイニング

Ⅰ．はじめに

ビジネスプロセスシミュレーション(BPS)は、サイクルタイム、リソース利用、または与えられたアクティビティの待ち時間などの主要なパフォーマンス指標に関して、プロセスの変更の影響を推定するために広く使用されている技法であり、反事実推論または「what-if」分析として知られている実践である[1]。BPS は、意思決定者がすでに変更を実施することなくプロセス設計を比較できるため、プロセス変更のリスクを大幅に低減し、具体的な成果を伴うプロセス改善を促進する可能性がある。とはいえ、BPS の有効性は、制御フロー、時間、リソースの挙動などの次元にわたって、与えられたプロセスのダイナミクスを正確に反映するシミュレーションモデルの利用可能性に大きく依存している。これらのシミュレーションモデルを手動で構築するのは、いくつかの落とし穴があるため、時間がかかり、エラーが発生しやすい[2]。そのため、イベントログに含まれる過去の実行データに基づいて、プロセスシミュレーションモデルを自動発見するための様々なアプローチが開発されている[3]-[7]。最も一般的なのは、このようなデータ駆動型シミュレーションアプローチは、プロセス全体の制御フローを捉えるプロセスモデルを発見し、その後、到着率、リソース、処理時間などのシミュレーションパラメータでこのモデルを補強することである。

本論文では、このような制御フロー優先シミュレーションモデルでは、実世界のプロセスのダイナミクスを忠実に表現できず、シミュレーションの不正確さにつながる設定があることを主張する。これは特

に、明確な資源行動や分散型の意思決定を伴うプロセスに適用される。ワークフローシステム[1]でサポートされるような、特定のプロセスは確かに中央で編成されるが、他のプロセスは関係者により高い運用柔軟性を提供する。このような設定において、プロセスの各アクターは、自分自身の視点から、ある程度は自分自身の方法で、つまり、同僚からケースを受け取り、必要と思われる 1 つ以上のタスクを実施し、そのケースを次の個人またはシステムに渡す。これは、行為者の特定の特徴や選好が、ケースの実行に影響を与えるプロセスをもたらす可能性がある。このような特性は、既存の制御フロー優先アプローチでは捉えにくい。

そこで本論文では、データ駆動型プロセスシミュレーションのためのリソースファーストアプローチである AgentSimulator を提案する。イベントログからマルチエージェントシステム(MAS)を発見することで、AgentSimulator は、実世界のアクターやシステムに対応する自律的で相互作用するエージェントによるプロセスの実行をシミュレートすることができる。これにより、AgentSimulator は、データ駆動型プロセスシミュレーションのための既存のアプローチと比較して、様々な利点を達成することができる:

- 我々のアプローチは、プロセスに関与する個々のリソースの動作に完全な柔軟性を提供し制御フロー動作、相互作用の好み、および能力の観点からの違いを捉えることを可能にする。
- エージェントベースのシステムは、高度に解釈可能であり、高度な適応性を可能にする。これは、what-if 分析を実行するために重要であり、したがって、ブラックボックス深層学習モデルに対する実質的な利点を提供する。
- 本アプローチは、従来のアプローチよりも実行時間が大幅に短縮されるため、多数のシミュレーションを実行することが可能であり、シミュレーションモデルの確率的性質を考慮すると、安定した信頼性の高い結果を得るために必要なものである。
- そして最後に、我々のアプローチは、様々なイベントログにおいて、最先端のシミュレーション精度をもたらす。

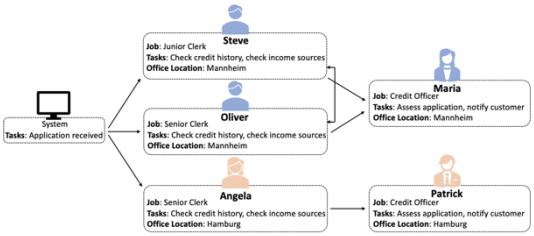

Fig. 1: Sketch of the credit application process.

残りは動機となるシナリオ(セクション II)から始まり、AgentSimulator のアプローチ自体の提示(セクション III)が続く。次に、セクション IV では評価実験について報告し、我々のアプローチの利点を強調する。最後に、セクション VI で本論文の結論を述べる前に、セクション V で関連する研究について述べる。

II. 動機

本節では、シミュレーションモデルを制御フロー優先からリソース優先の視点にシフトすることの利点を説明する。

この図では、簡略化されたクレジット申請プロセスを考え、その概略図を図 1 に示す。描かれているように、このプロセスは、申請者がシステムによって受領されたときに開始され、その後、申請者のクレジット履歴と収入源は、(どのような順序であれ)事務員によってチェックされる必要がある。両方のチェックが完了すると、申請書はクレジットオフィサーに渡され、クレジットオフィサーは申請書を評価し、申請者に結果の通知を行う。このように、このプロセスには 3 人の事務員(スティーブ、オリバー、アンジェラ)と 2 人の信用担当者(マリア、パトリック)が関与している。このような単純なシナリオであっても、あるケースに特定のアクターが関与することが、その実行に影響を与える様々な方法を観察することができる:

- プロセスのパフォーマンス。アクティビティの実行時間は、それを実行する従業員に依存する可能性がある。例えば、スティーブ(経験の浅いジュニア・クラーク)は、信用履歴と収入源をそれぞれ 45 分でチェックできるかもしれない。同じ活動で、シニア・クラークス・オリバー

とアンジェラは 15 分から 20 分しかかからない。このようなリソース間の性能差を考慮することは、最近実証されたように、正確なシミュレーションを行う上で非常に重要である[8]。

- 資源の利用可能性。あるプロセスに関与する従業員は、パートタイム労働やその他の職務などの要因により、異なるアベイラビリティを持つ可能性がある。このような考慮は、ケースを次に利用可能な人に割り当てることができる中央のワークフローシステムではなく、従業員から従業員に直接引き渡される可能性のある分散プロセスにおいて特に重要である。例えば、アンジェラが次に利用可能なクレジットオフィサーではなく、パトリックに特化したアプリケーションを手渡す場合、パトリックが次の営業日に利用できない場合、ケースの実行にかなりの遅れが生じる可能性がある。このような不規則性は、シミュレーションモデルに反映されるべきであり、最近再び精度にプラスの影響を与えることが実証されたように、個々のリソースカレンダーが必要である[8]。
- 制御フローの挙動。制御フロー優先シミュレーションモデルは、あるケースに対して実行される一連の活動が、そのケースに関与するアクターから独立しているという仮定を課す。しかし、そうでない理由は様々である。例えば、我々のシナリオでは、他のアクターがこれらの注文を交互に行うのに対して、アンジェラは常に収入源をチェックする前に、まずクレジット履歴をチェックすることが観察されるかもしれない。さらに、あるケースの可能なシーケンスに影響を与える行為者固有のルールが存在する可能性さえある。例えば、ジュニアクラーク(例:Steve)が扱うアプリケーションは、シニアクラークが実行する追加の検証ステップを経る必要があるかもしれない。このようなアクター依存の挙動を捉えることは困難であり、プロセスの実行にかなりの影響を与えるにもかかわらず、制御フロー優先のシミュレーションモデルでは一般的に省略されることが多い。
- 相互作用のパターン。最後に、中心的なオーケストレーションがないため、アクター間の具体的な相互作用パターンが存在する可能性がある。例えば、スティーブはいつもオリバーと共同作業をしている。なぜなら、二人ともマンハイムで働いているが、ハンブルクのアンジェラと接触していないからだ。このような特定のパターンは、個々のリソースの仕事量に影響を与える。例えば、Oliver と Angela の間の不均衡につながるが、これは一般的なシミュレーションモデルでは見逃される。さらに、上述したように、個々のリソースの利用可能性が異なる場合、特定の相互作用パターンがさらなる遅延につながる可能性がある。

上記のすべての要因は、プロセスにおけるケースの実行に影響を与える。したがって、プロセスシミュレーションモデルは、実世界のプロセスのダイナミクスを適切に模倣するために、これらを可能な限り忠実に反映する必要がある。前者の 2 つの側面は、すでに制御フロー優先モデル[8]によって認識され、捉えられている。しかし、特に後者の 2 つは、各リソースの振る舞いを明示的に捕捉するエージェントベースのシミュレーションモデルによって、より自然に取り入れることができる。これは、次に説明する AgentSimulator のアプローチによって達成される。

III. 方法論: AgentSimulator

本節では、データ駆動型エージェントベースビジネスプロセスシミュレーションアプローチである AgentSimulator を紹介し、図 2 に概要を示す。リソース優先の視点を採用するために、AgentSimulator は、(一般的なシミュレーションパラメータとともに)発見フェーズ(セクション III-B を参照)でイベントログから発見される MAS(セクション III-A を参照)でエージェントをモデル化する。MAS の発見に続いて、プロセスをシミュレートし、新しいイ

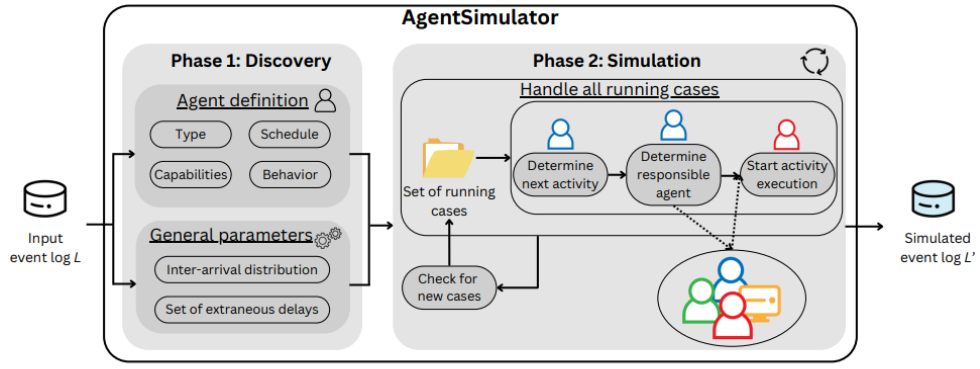

Fig. 2: Overview of the end-to-end AgentSimulator approach.

ベントログを生成することができる(セクション III-C 参照)。発見とシミュレーションの両方に対する我々のアプローチを詳述し、また、異なるプロセスタイプに対する代替設計の選択について議論し、AgentSimulator の適応性を強調する。

A. 定義

本節では、エージェントの定義と本アプローチが採用する MAS を提供する前に、本アプローチの入力(およびそのシミュレーション出力)を提供するイベントログを定義する。

Input. AgentSimulator は、入力としてイベントログ(L)を受け取る。このイベントログは、トレースの有限多集合として定義される。トレース($\sigma \in L$)は、1 つの組織プロセスのケースに対して実行された活動の実行を記録する有限のイベント列 $(\langle e_1, \dots, e_n \rangle)$である。各イベント$(e_i)$はタプル$((act, ts_{\text{start}}, ts_{\text{end}}, res))$で構成され、(act)は対応する活動、(ts_{start})と(ts_{end})はそれぞれ活動実行の開始および終了のタイムスタンプ、(res)はその活動を実行したリソースを表す。なお、[8]に従い、各イベントは開始タイムスタンプと終了タイムスタンプを用いて表現される。この形式は、シミュレーションの設定において活動の持続時間を考慮するために必要であり、トレース内のイベントはその開始タイムスタンプに基づいて順序付けられる。　以降の部分では、タプルの構成要素を参照する際にドット表記を用いることが一般的である。例えば、イベント(e_i)の活動を指す略語として$(e_i.act)$を使用する。また、イベントログ(L)のトレースに含まれる活動とリソースの集合をそれぞれ(ACT_L)および(RES_L)として表記する。

マルチエージェントシステム: エージェントの概念は、人工知能(AI)[9]の基本的な抽象化である。エージェントは、環境(他のエージェントを含む)を知覚して、その知覚を推論し、行動を決定する。マルチエージェントシステム(MAS)では、エージェントは共同目標を達成するために協調することができる。我々の研究では、MAS とエージェントを以下のように定義する:

定義 1(マルチエージェントシステム) AgentSimulator のための MAS（マルチエージェントシステム）をタプル$(m = (A, p))$として定義する。ここで、(A)はプロセスをシミュレーションするために使用されるエージェントの集合を表し、(p)はプロセスの環境の側面を反映する一般的なシミュレーションパラメータのタプルを表す。(p)は、ケース間到着分布と外来遅延に関する確率密度関数（PDF）の集合から構成される（詳細はセクション III-B 参照)。

定義 2(エージェント) ビジネスプロセス環境におけるエージェントは、実世界のアクターやシステムを表す自律的な存在である。MAS (m) において動作するエージェント ($a \in A$) をタプルとして定義し、$(a = (t, s, c, b))$と表す。ここで：

1) (t)はエージェントのタイプを指す。エージェントタイプは、例えば、エージェントが実行できる活動の観点から、類似した特性を持つエージェントを示すために使用される。(A)のタイプの集合を指すのに(T)を使う。

2) (s)は、以下のようなスケジュール(間隔の集合)を指す。エージェントがアクティビティを実行できる。シミュレーションでは、プロセスの時間的次元を忠実に反映するために、エージェントの明確な利用可能性(例えば、フルタイム対パートタイム)を捉える必要がある。
3) (c)はエージェントの能力を意味し、タプルとして(c = (ALLOC,PT))と表記される。
   - ALLOC は(ACT_L)の部分集合であり、エージェント(a)が実行可能な活動、すなわちエージェント(a)に割り当て可能な活動の集合を指す。
   - (PT)は確率密度関数(PDF)の集合を指し、各($f_{pt}(act) \in PT$)は、活動($act \in ALLOC$)に対する処理時間($pt \in [0, \infty)$)の分布を表す。エージェントごとに(PT)を定義することで、リソースごとのプロセス性能の違いを明確にすることが可能となる。
4) (b)はエージェントの振る舞いを指し、エージェント(a)が活動を完了した後にケースをどのように引き渡し、その実行を継続させるかを捉える要素である。(b)の具体的な内容は、シミュレーション対象のプロセスの種類に依存する。中央で統制されたプロセスでは、次の活動の決定は中央で行われる一方で、より自律的なプロセスでは、エージェントが直接他のエージェントにケースを引き渡す。このため、本論では以下のセクションにおいて、詳細な説明および正確な定義を提供する。

B. フェーズ 1: MAS の発見

本節では、AgentSimulator がイベントログ L からエージェント A と一般的なシミュレーションパラメータpを発見し、MAS mを定義する方法について説明する。ビジネスプロセスの領域では、エージェントの一般的な定義は、多くの実装の可能性を提供する。このように、我々の発見アプローチを説明する以外に、代替案についても議論する。エージェントシステムの柔軟性は、我々の AgentSimulator の重要な利点であり、多様な協調作業ダイナミクスへの適応を可能にする。エージェントのインスタンス化。ビジネスプロセス環境では、リソースはアクションを実行するアクティブなエンティティである。したがって、各資源$res \in RESL$に対して 1 つのエージェント$a \in A$をインスタンス化する。このように、動機となる例では、6 つのエージェント(5 人の人間アクターとシステム)をインスタンス化する。ある事象が資源情報を欠いている場合、さらに、そのような情報を欠いている各個別の活動に対してダミーエージェントを生成する。資源情報がない場合、瞬時の活動やシステム資源を指すことが多いことに気づいた。まず、これらをエージェントとして明示的にモデル化することで、シミュレーションに必要な、少なくとも 1 つのエージェントに関連する活動が保証される。第二に、これらのエージェントが自分の活動を適切にシミュレートするための明確な行動を発見することができる。例えば、瞬間的な活動を行うには、対応するエージェントを待つ必要はないと考えることができる。

エージェントタイプ: 各エージェントにタイプを割り当てるために、[10]で提案されたアルゴリズムを使用し、実行された活動の類似性に基づいてエージェントをタイプごとにクラスタリングする。このアルゴリズムは、[4]でもシミュレーションされたリソースをグループ化するために使用されている。

スケジュール: 各エージェント($a \in A$)のスケジュールを発見するために、[8]で提案されたアルゴリズムを使用する。なお、エージェントタイプごとにスケジュールを発見することも可能であり、これは本アプローチが許容する設計選択の一つである。さらに、確定的なカレンダーではなく確率的なカレンダーを発見するための最近のアプローチ[11]も統合することが可能である。

能力: エージェント(a)の能力($a.c$)は、活動の集合と処理時間に関する PDF（確率密度関数）の集合で構成される。

活動の集合: エージェント(a)が実行可能な活動の集合($a.c.ALLOC$)を発見するために、イベントログ(L)内でリソース(a)が実行した(ACT_L)の活動を確認する。

処理時間: 処理時間に関する PDF の集合($a.c.PT$)には、各活動に対して 1 つの分布が含まれ、エージェント(a)が各活動を実行するのに必要な時間を反映する。これに従い、[4]のアプローチに基づいて、活動の継続時間に対してさまざまな分布を適合させる。その後、ワッサースタイン距離で測定した誤差が最小となる分布を選択する。分布のセットには、以下のものが含まれる：指数分布、ガンマ分布、正規分布、一様分布、対数正規分布、および固定期間を持つ活動を表す固定値。ただし、各活動ごとに 1 つの PDF を発見する代わりに、エージェントと活動の組み合わせごとに 1 つの PDF を発見する。これは、($ACT_L \times A \rightarrow PTT$)として表され、($PTT = \bigcup_{a \in A} a.c.PT$)である。この方法により、例えば、ジュニア社員がシニア社員よりも同じ活動に多くの時間を要する場合など、より現実的な活動処理時間が保証される。

タイプ、スケジュール、能力のプロパティは、エージェントベースでないシミュレーションアプローチでも一般的に考慮される側面であり、その発見において既存の研究に概ね従っている。しかし、エージェント固有の行動は、我々の AgentSimulator の主要な要素として機能する。

エージェントの行動: エージェントの行動($a.b$)は、その活動遷移およびエージェント間の引き渡しパターンを記述する。したがって、エージェントの行動は(i)進行中のケースにおける次の活動が何であるか、(ii)その活動を誰が実行するかを決定する。エージェントの行動はさまざまな方法で学習可能であり、これにより MAS（マルチエージェントシステム）のアーキテクチャ設計には多数の選択肢が生じる。以下では、行動を発見するための 2 つの明確なアプローチを概説する。一つはオーケストレーションされた引き渡しを伴うプロセスを捉えるのに適したものであり、もう一つは自律的な引き渡しを伴うプロセスに適している。

オーケストレーションされた引き渡し。この最初の構成では、ワークフローやビジネスプロセスマネジメントシステム[1]でサポートされるような、中央でオーケストレーションされたプロセスを模倣する。このようなプロセスでは、ケースの実行が中央で管理される。そのため、この種のアーキテクチャでは、特定のエージェントに依存しないグローバルな実行パターンを発見するだけでよい。この目的を達成するために、ログレベルで活動遷移確率を学習する。

具体的には、進行中のケースにおける活動プレフィックスが与えられた場合、プレフィックス($\sigma_{\text{prefix}} = \langle e_1, e_2, \ldots, e_k \rangle$)は、イベント(σ)の開始からイベント(e_k)までのイベントのシーケンスであり、それに対応する活動シーケンスを($\sigma^{\text{act}}_{\text{prefix}}$)とする。この遷移確率($P\left(\text{act} \middle| \sigma^{\text{act}}_{\text{prefix}}\right)$)は、ログ(L)内で可能な各活動プレフィックス($\sigma^{\text{act}}_{\text{prefix}}$)の後に各活動($\text{act} \in \text{ACT}_L$)が現れる頻度を計算することで求められる。具体的には、特定の遷移($\sigma^{\text{act}}_{\text{prefix}} \rightarrow \text{act}$)が発生する回数を、そのプレフィックスがログ内で発生する回数で割ることで計算される。

なお、もしログ内で活動プレフィックス($\sigma^{\text{act}}_{\text{prefix}}$)が観測されていない場合、プレフィックスの最初の活動を順次削除し、ログ(L)で観測された部分シーケンスに到達するまで繰り返す。例えば、シーケンス($\langle a, b, c, d \rangle$)が観測されていない場合、次に($\langle b, c, d \rangle$)の出現を確認する。ケースの終了への遷移には、各ケースの最後の活動に続くプレースホルダー終了イベントを導入する。さらに、一般性を損なうことなく、次の活動予測技術[12], [13]を用いるなど、他の方法で遷移確率を計算することも可能である。

自律的な引き渡し: セクション II で述べたように、多くのプロセスでは、関与するエージェントに高い柔軟性と意思決定権が与えられる。そのため、AgentSimulator は、プロセスインスタンスの次の活動とその実行エージェントをエージェント自身が決定する分散型 MAS（マルチエージェントシステム）の発見にも使用できる。この場合、活動遷移確率はグローバルではなくローカルに学習する必要がある。このローカ

リティは、個々のエージェントまたはエージェントタイプによって定義される。すなわち、活動およびエージェント遷移確率は、各エージェントごとに固有であるか、または同じタイプ($t \in T$)のすべてのエージェント間で一般化される。後者は、各エージェントに利用可能なデータが限られている場合に適している。

与えられた活動プレフィックス($\sigma^{\text{act}}_{\text{prefix}}$)から活動(act $\in \text{ACT}_L$)への遷移確率を各エージェント($a \in A$)ごとに計算するためには、オーケストレーションされた引き渡しのために説明した計算を拡張し、エージェントに依存させることができる。したがって、($P\left(\text{act}\middle|\sigma^{\text{act}}_{\text{prefix}}, a\right)$)は、エージェント(a)がプレフィックスの最後の活動を実行した場合に($\sigma^{\text{act}}_{\text{prefix}}$)から活動(act)への遷移が発生する回数を数え、それを(a)がプレフィックスの最後の活動を実行した回数で割ることで計算される。例えば、図 1 におけるアンジェラの場合、プレフィックス「申請受領」が与えられた場合、クレジット履歴を収入源よりも先にチェックする。一方、スティーブとオリバーには固定された順序がなく、時には協力して作業を行うことがある。

活動遷移に加えて、自律的な引き渡しアーキテクチャではエージェント間の特定の相互作用パターンも考慮する。そのため、あるエージェント(a_j)から別のエージェント(a_i)へのケース引き渡しの頻度確率を計算する。エージェント(a_j)から(a_i)へのタスク引き渡しの条件付き確率($P\left(a_i\middle|a_j\right)$)は、($a_j$)が活動を実行し、その次の活動が($a_i$)によって実行されるすべての事例を数え、それを($a_j$)が実行した活動の総数で割ることで計算される。これに基づき、アンジェラからマリアへの引き渡し確率 ($P(\text{Maria}|\text{Angela}) = 0.0$)、アンジェラからパトリックへの引き渡し確率 ($P(\text{Patrick}|\text{Angela}) = 1.0$)となる。

他にも引き渡し確率を計算する方法は多数存在する。例えば、条件付き確率 ($P\left(a_i\middle|a_j, act\right)$)に特定の活動の依存性を含めることで、相互作用パターンをさらに精密にすることが可能である。相互作用パターンを決定する異なるアプローチを比較することは、将来の研究課題として取り組むことができる。($P\left(act\middle|\sigma^{act}_{prefix}, a\right)$)と ($P\left(a_i\middle|a_j\right)$)の組み合わせは、自律的な引き渡しアーキテクチャにおけるエージェントの行動を定義し、ケースの進行に影響を与えるエージェント固有の特性を捉える。

一般的なシミュレーションパラメータ: エージェントの集合¥(A ¥)をインスタンス化した後、いくつかの一般的なシミュレーションパラメータ($m.p = (fiat, D)$)を発見する。ここで、($fiat$)はケース間到着時間の確率密度関数(PDF)を表し、(D)は外来遅延に関する PDF の集合を表す。これらのパラメータは、エージェントベースのシミュレーションモデルに特有のものではなく、他の BPS アプローチ[4]でも使用される。

ケース間到着時間: ケース間到着時間は、連続する 2 つのケースの開始の間隔を示す。シミュレーション中、ケース間到着時間の PDF を使用して新しいケースをサンプリングする。ケース間到着時間の PDF($fiat$)を発見するために、[4]に従い、到着時間にさまざまな分布を適合させ、誤差が最小となる分布を選択する。

外来遅延: プロセスの異なるインスタンスは通常、限られたリソースを競合しており、リソースが活動を可能な限り早く開始しない場合があるため、プロセスは待ち時間の影響を受ける。外来遅延は、リソースの競合や利用不可が原因ではない待ち時間(例: リソースが顧客からの電話応答を待つ場合)を指し、明示的にモデル化する必要がある。[14]のアルゴリズムに従い、各活動に対して外来遅延に関する PDF を発見し、外来遅延分布の集合(D)を構築する。

C. フェーズ 2: シミュレーション

このセクションでは、AgentSimulator アプローチが発見された MAS (m) を使用してイベントログ (L')をシミュレーションする方法を説明する。アルゴリズム 1 の疑似コードに示されるように、シミュレーションは時間刻み（タイムティック）を表す離散的なステップで進行する。各ステップで、新しいケースが到着したかどうかを確認し、システム内の各進行中のケースについて、(例えば、前の活動の完了後に）処理待ちの状態にあるかを確認して次の活動

をインスタンス化する。シミュレーションステップは、すべてのケースを確認した後に終了する。詳細なシミュレーションステップは以下のようになる。

1) **新しいケースの確認**: 発見されたケース間到着分布 $(p.fiat)$に基づいて、新しいケースが到着したかを最初に確認する（ライン 3)。システム内に複数のケースがある場合、その処理順序は先入先出方式で決定される。テストログとの評価時には、Simod [4]の慣例に従い、シミュレーション対象のケース数のみを到着させる。しかし、通常のシミュレーションでは、シミュレーションが終了するまでケースが到着し続けるようにする。これにより、クールダウンフェーズを避け、より現実的なシミュレーションを実現する。

2) **進行中のケースの処理**: ケースは一つずつ処理される（ライン 4）まで、すべてのケースが確認される：

a) 次の活動の決定: まず、指定されたケース (σ_{prefix})が処理待ちの状態にあるかを確認する（ライン 5)。処理待ちの場合、新しいイベント (e) が作成され、その対応する活動 $(e.act)$が決定される（ライン 7)。この決定は、現在の活動プレフィックスに基づくグローバルモデリング（すなわち、中央のエンティティが制御フロー実行をオーケストレーションする）か、ケース内で最後にアクティブだったエージェントを通じたローカルモデリング（自律的な引き渡しのモデリング）のいずれかによる。ケースの最初の活動は、前のエージェントがいないため、常にグローバルに決定される。

b) 責任エージェントの決定: 指定されたケースに対して新しい活動が決定された後、その実行のためのエージェントが必要である。エージェントが実行可能な活動の集合$(a.c.ALLOC)$に基づき、可能な責任エージェント (p_a)を特定する（ライン 11)。特定のエージェントを決定する際（ライン 12)、以下の 2 つのアーキテクチャ間で区別される：

- オーケストレーションされた引き渡し。オーケストレーションされた引き渡しを持つ MAS では、エージェント間の相互作用パターンはモデリングされない。そのため、可能な責任エージェントの集合が決定された後、この集合をエージェントの利用可能性に基づいて順序付け、次の活動を実行するために利用可能なエージェントが見つかるまで順次依頼する（これを反復タスク割り当てと呼ぶ)。最初に依頼されるエージェントは、スケジュール $(a.s)$ と現在の作業状況に基づいて最も早く利用可能なエージェントである。エージェントは、活動の推定持続時間 $((f\_\{pt\}(act) \in a.c.PT)$ に基づく）が占有または非稼働時間枠と競合する場合、活動の受け入れを拒否する。可能なエージェントが利用可能でない場合、そのケースは現在処理できず、次のシミュレーションステップで再確認される。このため、ケースは競合による待機時間を受け取る（ライン 14)。
- 自律的な引き渡し。自律的な引き渡しを持つ MAS では、計算されたエージェント引き渡し確率 $(P(a_i|a_j))$を使用して次のエージェントを決定する。これには 2 つの設計選択肢がある：(i)反復タスク割り当て、(ii)直接タスク割り当て。(i)反復タスク割り当てを使用する場合、オーケストレーションされた引き渡しと同様に、エージェント (a_j)が他のエージェントに活動を引き受けるよう順次依頼する。しかし、ここでは依頼するエージェントの順序がエージェント引き渡し確率に基づいてランク付けされる。つまり、(a_j) からの引き渡し確率が最も高いエージェント(a_i)に最初に依頼される。(ii)直接タスク割り当てでは（アルゴリズム 1 には表現されていない)、現在のエージェント (a_j)は他のエージェントに依頼するのではなく、活動を担当するエージェントに直接割り当て、エージェントは時間を見つけ次第実行を開始する。この方法は、例えば、従業員がケースを同僚にメールで転送するような実際のプロセスで一般的なエージェント相互作用パターンとなる。この直接割り

Algorithm 1 Simulation

Input: A: set of agents; n: number of to-be simulated cases
Input: f_{iat}, D: general simulation parameters
1: $\Sigma \leftarrow \{\}, L' \leftarrow \{\}$ ▷ init. set of running traces Σ and output log L'
2: **while** $len(L') \leq n$ **do**
3: $\Sigma \leftarrow check\_for\_new\_cases(f_{iat}, \Sigma, evaluation = False)$
4: **for** σ_{prefix} in Σ **do**
5: **if** $is\_waiting(\sigma_{\text{prefix}})$ **then**
6: $e \leftarrow ()$
7: $e.act \leftarrow get\_next\_activity(\sigma_{\text{prefix}})$
8: $p\_a \leftarrow \{\}$
9: **for** a in A **do**
10: **if** $e.act$ in $a.c.$ALLOC **then**
11: $p\_a \leftarrow add\_to\_potential\_agents(a)$
12: $e.a \leftarrow get\_agent(p\_a)$
13: **if** $e.a =$ None **then**
14: $is\_waiting(\sigma_{\text{prefix}}) \leftarrow True$
15: **else**
16: $e.ts_{start}, e.ts_{end} \leftarrow execute\_act(e, D)$
17: $\sigma_{\text{prefix}} \leftarrow add\_to\_prefix(e)$
18: **if** $e.act = end$ **then**
19: $L', \Sigma \leftarrow exit\_and\_add\_to\_log(\sigma_{\text{prefix}})$

TABLE I: Description of log properties.

Log	Type	#Traces	#Events	#Activities	#Resources	#Agents
Loan Application	syn	1000	7492	12	19	19
P2P	syn	608	9119	21	27	27
CVS	syn	10000	103906	15	6	8
Confidential 1000	syn	1000	38160	42	14	26
Confidential 2000	syn	2000	77418	42	14	26
ACR	real	954	6870	18	432	432
Production	real	225	4503	24	41	41
BPI12W	real	8616	59302	6	52	56
BPI17W	real	30276	240854	8	136	136

当てのために、エージェント引き渡し確率に基づいてエージェントをサンプリングする。

なお、ケースの最初のエージェントは、オーケストレーションされた引き渡しに記述されている方法で常に決定される。

c）活動の実行を開始。エージェントが特定された場合、そのエージェントは割り当てられた活動の実行を開始する。終了タイムスタンプは、エージェント固有の活動期間分布$(f_{pt}(act) \in a.c.PT)$からサンプリングし、必要に応じて外来遅延分布$(f_d(act) \in p.D)$を追加することで決定される（ライン 16）。

各ケースの処理が完了すると、1 つのシミュレーションステップが終了する。ケースは、最終活動が実行されるとシステムを退出する（ライン 19）。シミュレーション全体は、指定されたケースの数が処理され、終了した時点で終了する。シミュレーションの出力はイベントログ(L')であり、各イベントはタプル$(e = (act, ts_{start}, ts_{end}, a))$である。

IV. 実験と結果

本節では、AgentSimulator の性能評価に用いた実験について述べる。表 I は、BPS 評価[5], [6], [15]で一般的に使用されている、開始と終了の両方のタイムスタンプを含む 9 つの公開イベントログの特徴をまとめたものである。我々の実装、イベントログ（訓練とテストの分割付き）、追加結果は、我々の公開リポジトリ 1 から入手可能である。

A. 実験の設定

実装: 我々のアプローチは、エージェントベースのモデリングフレームワークである mesa [16]を使用して Python で実装した。

ベンチマークアプローチ: 我々の AgentSimulator（AgentSim）を、3 つの一般的なデータ駆動型 BPS アプローチと比較する。まず、[8]で提案された最新のデータ駆動型プロセスシミュレーション（DDPS）アプローチを採用し、これを Simod と呼ぶ。このアプローチは、元の Simod[4]と、リソースの利用可能性と性能の差異を考慮し、元の Simod を上回る性能を示した Prosimos シミュレーターを組み合わせたものである。DeepGenerator（DGEN）[12]は純粋なディープラーニング（DL）アプローチであり、DeepSimulator（DSIM）[6]は DDPS と DL のハイブリッドアプローチである（詳細はセクション V 参照）。

データ分割: 我々は既存の BPS アプローチ[5], [6], [12]の評価に従い、時間的ホールドアウト分割を実施する。訓練セット（最初のケースの 80%）とテストセット（最後のケースの 20%）の分割時にまたがるケースはすべて除外する。

ハイパーパラメータ: AgentSim には、2 つの自動的に決定されるハイパーパラメータがある：(1)アーキテクチャ（オーケストレーションまたは自律的な引き渡し）と(2)外来遅延を考慮するかどうか。この 2 つにより、4 つの可能な構成が得られる。我々は外来遅延をハイパーパラメータとして扱う。なぜなら、イベントログごとに外来遅延を考慮することの利益に大きな違いが見られたためである。これらのハイパーパラメータは、訓練セットの最後の 20%を 4 つの可能な構成のそれぞれでシミュレーションし、サイクルタイムの観点で訓練サブセットに最も近いシミュレーションを選択することで決定する。公平な比較を確保

TABLE II: Comparison of simulation approaches.

Log	Method	Metrics NGD	AED	CED	RED	CTD
Loan Appl.	Simod	0.15	13.55	0.40	9.22	20.42
	DGEN	0.21	212.27	13.40	5.26	9.38
	DSIM	n/a	n/a	n/a	n/a	n/a
	AgentSim	**0.07**	**2.78**	**0.21**	**1.34**	**1.49**
P2P	Simod	0.42	**1044.25**	2.21	840.19	677.05
	DGEN	**0.20**	1481.46	2.55	828.09	670.05
	DSIM	0.22	1310.03	1.15	722.33	566.63
	AgentSim	0.25	1161.32	**1.02**	**658.61**	**525.15**
CVS	Simod	0.44	52.95	**0.44**	39.43	54.59
	DGEN	0.21	310.39	11.69	176.65	294.21
	DSIM	0.20	**36.23**	8.98	**19.74**	**52.43**
	AgentSim	**0.12**	89.31	7.48	81.76	101.49
C. 1000	Simod	0.25	344.48	3.01	468.81	804.07
	DGEN	0.58	462.84	18.93	8.11	13.92
	DSIM	**0.20**	246.41	2.28	**5.34**	**7.29**
	AgentSim	0.25	**127.01**	**1.68**	16.84	26.10
C. 2000	Simod	0.24	820.45	2.96	952.37	1614.91
	DGEN	**0.16**	857.68	18.09	4.58	8.12
	DSIM	0.18	591.13	2.84	**1.7**	**2.26**
	AgentSim	0.26	**212.54**	**1.41**	9.29	17.87
ACR	Simod	**0.23**	287.27	**2.60**	32.46	93.51
	DGEN	0.31	559.67	17.84	30.87	95.11
	DSIM	0.26	**273.46**	4.64	**15.62**	**48.24**
	AgentSim	0.36	333.28	7.95	27.12	76.50
Production	Simod	0.93	146.38	2.82	83.88	89.15
	DGEN	**0.52**	224.45	9.30	70.11	90.82
	DSIM	0.86	154.31	**2.66**	33.30	43.26
	AgentSim	0.61	**65.29**	5.83	**14.20**	**25.79**
BPI12W	Simod	0.72	**71.97**	**1.71**	95.72	155.46
	DGEN	0.43	306.28	4.53	116.18	176.79
	DSIM	0.65	78.62	2.88	119.12	173.49
	AgentSim	**0.15**	79.89	1.87	**47.83**	**90.12**
BPI17W	Simod	0.59	300.28	3.34	136.63	148.40
	DGEN	0.67	4557.19	3.39	118.84	172.94
	DSIM	0.53	**54.61**	3.35	33.10	30.26
	AgentSim	**0.30**	220.98	**1.64**	**26.03**	**22.75**

するために、Simod にも同じ外来遅延の選択を適用する。

評価指標: 異なるシミュレーションアプローチを評価・比較するために、最近提案された評価指標を使用する。これらの指標は、制御フロー、時間、および混雑の 3 つの次元にわたるシミュレーションモデルを評価するために設計されており、包括的な視点を提供する[15]。すべての指標は、シミュレーションログとテストログの間の距離を計算し、値が低いほど結果が良好であることを示す。制御フローを測定するために、イベントログで観測された n-グラムの頻度差を計算する N-Gram Distance（NGD）を使用する。シミュレーションの時間的性能を測定するために、Absolute Event Distribution（AED）、Circadian Event Distribution（CED）、および Relative Event Distribution（RED）を使用する。混雑を表現するモデルの能力を測定するために、Cycle Time Distribution（CTD）を使用する。我々は Case Arrival Rate を測定しない。なぜなら、Simod と同じケース到着方式を適用しているためで

Fig. 3: Resource interactions for the BPI12W log.

あり、AgentSim が他のアプローチと異なる指標に焦点を当てる。

B. 結果

全体の結果: 表 II は、9 つのイベントログにおける結果をまとめたものであり、各ログごとに 10 回のシミュレーション実行から得られた平均的な評価指標を示している。各ログおよび指標において最良の値は太字で、2 番目に良い値は下線で示されている。これまでの BPS 研究[5], [6]でも観察されているように、すべてのデータセットと指標において一貫して他のアプローチを上回るものは存在しない。しかし、AgentSim は 9 つのログのそれぞれで最も頻繁に最高の性能を達成しており、その際立った成果を示している。指標が捉える 3 つの次元に着目すると、以下のような主な知見が得られる。

制御フロー: AgentSim は 9 つのログのうち 4 つで最高の N-Gram Distance（NGD）を達成し、Simod や DSIM を大幅に上回る結果を示している。この結果は、プロセスモデルに依存しないリソースファーストの AgentSim アプローチが、Simod のような制御フローファーストのアプローチと比較して制御フローの次元における精度を向上させることを示している。特に、2 つの実世界の BPI ログを分析した際に顕著であり、シミュレーションの中心にリソースを置くことで、制御フローの挙動をより忠実に表現できることが多い。

時間: 時間に基づく絶対（AED）、サーカディアン（CED）、および相対（RED）のイベント分布に関して、AgentSim は 3 つの指標すべてでリーダーであり、DSIM がそれに続く。Simod と DGEN は平均して AgentSim より大幅に劣る結果を示している。この結果は、AgentSim がログの時間的パターンを比較的正確に捉えることを示している。

混雑: プロセスインスタンスのサイクルタ

イムを正確に捉えることは、シミュレーションアプローチの精度を評価する重要な指標となる。CTD（Cycle Time Distribution）指標は、個々の活動の処理時間とそれに対応する待機時間によって影響を受ける。これらはリソースの利用可能性に依存するため、サイクルタイム指標は包括的な要因を捉えている。AgentSim は CTD において 9 つのログ中 5 つで最高の精度を示し、残りの 4 つでは DSIM がリーダーとなっている。特に、Confidential ログにおいて、AgentSim は Simod を大きく上回る優れた結果を示している。

Simod のシミュレーションログでは、リソース競合とそれに伴う待機時間が原因で、現実よりもサイクルタイムが著しく長くなることが多い。AgentSim は、瞬間的な活動を実行するエージェントなど、一部のエージェントには待機時間が不要であることを認識することで、この問題を軽減している。

引き渡し構成の影響: 中央オーケストレーション方式と自律方式という 2 つの異なる引き渡し構成を使用して同じプロセスをシミュレーションすると、一部のログで結果にかなりの違いが見られる（両オプションの完全な結果はリポジトリに記載）。全体的に、AgentSim における自動的な引き渡し構成の選択は、9 つのプロセスのうち 5 つをオーケストレーション方式でシミュレーションする結果につながり、例えば P2P や Production のようなプロセスでは、リソースファーストアプローチであるにもかかわらず、AgentSim は依然として強力な結果を示している。一部のログでは、自律方式ではなくオーケストレーション方式でシミュレーションすることで、大幅な改善が見られる。例えば、Production ログでは、CTD が 45.65 から 25.79 に、NGD が 0.77 から 0.61 に改善し、標準化された生産プロセスの性質と一致している。一方、自律方式でシミュレーションされたログでは、特定のプロセスに対してエージェント固有の行動を考慮することの明確な利点も観察される。例えば、実世界の BPI17W ログでは、CTD と RED の両方がそれぞれ 22.75 と 26.03 に半減し、オーケストレーション方式と比較して大幅に改善されている。また、C. 1000 および Loan Appl. では、自律方式を使用することで全指標にわたってやや良い結果を示している。これらの知見は、AgentSim が多様なプロセスタイプに適応可能であり、結果の大幅な改善につながることを示している。

エージェント間の相互作用に関する事後分析: AgentSim がエージェント相互作用を表現する能力を具体例として示すために、実世界の BPI12W ログにおける相互作用パターンを Simod と比較する。図 3 は、AgentSim がトレーニングログの相互作用ダイナミクスを正確に反映し、リソースの活動連鎖（対角線部分参照）を捉えていることを示している。一方で、Simod はランダムな相互作用を示し、これらのパターンを捉えることができず、特定のプロセスにおいて個々の相互作用パターンを考慮することの重要性を示している。

実行時間: シミュレーションの確率的性質を考慮すると、信頼性の高い予測を達成するために多数のログをシミュレーションする必要がある。そのため、実行時間は BPS を実際に適用する際の重要な要素である。この点において、AgentSim は一貫してベンチマークよりもはるかに高速に動作する。例えば、AgentSim は Production ログを発見しシミュレーションするのに約 30 秒しか必要としない（32GB の RAM および Intel Core i7 2.3 GHz CPU のマシン上）。一方、Simod はその 20 倍、DSIM は 80 倍以上の時間を要する。BPI12W ログにおいて、AgentSim は約 9 分で完了するのに対し、DSIM は 10 時間以上を要する。

V. 関連研究

本節では、自動化された BPS（ビジネスプロセスシミュレーション）およびエージェントベースのモデリングとシミュレーションに関連する研究について簡単に述べる。

自動化された BPS: 自動化された BPS アプローチに関する既存の文献は、大きく 3 つのカテゴリに分けられる。データ駆動型プロセスシミュレーション（DDPS）、ディープラーニング（DL）、およびハイブリッドアプローチである。DDPS アプローチは、イベントログからシミュレーションモ

デルを発見するプロセスを自動化し、最初にプロセスモデルを特定し、それにシミュレーションパラメータを追加する。カラー付きペトリネットを使用した半自動アプローチが[3]で提案されており、[7]ではリソースを考慮せずにデータ駆動型アプローチを導入している。より最近では、Simod[4]がハイパーパラメータチューニングを取り入れたアプローチを提供している。BPS の DL アプローチは通常、リカレントニューラルネットワークに依存している。LSTM モデルは、イベントとタイムスタンプを予測するために[13]で使用され、その後、n-グラムと埋め込みを組み込んだ DGEN[12]によって改善された。しかし、DL モデルはブラックボックス的性質を持つため、what-if 分析には適用できない。ハイブリッドモデルは、DDPS と DL アプローチを組み合わせたものである。DSIM[6]はイベントのタイムスタンプを生成するために確率的プロセスモデルと DL を組み合わせており、RIMS はランタイムで予測を統合することでこれを拡張している[5]。

エージェントベースのモデリングとシミュレーション: 過去数十年にわたり、MAS（マルチエージェントシステム）のさまざまな分野への適用が広く研究されてきた（詳細は[17]のレビューを参照）。エージェントをビジネスプロセスマネジメント（BPM）に適用するという概念は 1990 年代に最初に提案され[18]、ビジネスプロセスは交渉するエージェントのシステムとしてモデリングされた。より最近では、エージェントシステムマイニングの概念が導入され、プロセスは自律的なエージェントの相互作用からしばしば生じることが認識された[19]。この概念は、[20]でのエージェントベースの発見アルゴリズムや[21]でのシミュレーションによって実証されている。エージェントベースの BPS の一般的な導入については[22]を参照されたい。しかし、BPM におけるこのようなエージェントベースのシミュレーションアプローチは、特定のプロセスをシミュレーションするために手動での設定に依存しており、例えば工場生産領域における事例が挙げられる[23]。我々の知る限りでは、イベントログを使用してプロセスシミュレーションのための MAS モデルを自動的に推論するアプローチは、本研究が初めてである。

VI. 結論

本論文では、データ駆動型ビジネスプロセスシミュレーションのためのエージェントベースアプローチである AgentSimulator を紹介した。イベントログが与えられた場合、本アプローチは実世界のアクターやシステムを表現する MAS（マルチエージェントシステム）を発見し、それぞれが独自の行動と相互作用パターンを持つようにモデル化する。その発見された MAS はプロセスの実行をシミュレートするために使用される。我々のリソースファーストアプローチは、従来の制御フローファーストアプローチよりも、リソース固有の行動と相互作用を捉えるためのより多くの手段を提供し、大幅に短縮された計算時間で最先端の結果を達成している。評価の結果、中央オーケストレーション型プロセスと分散型プロセスはしばしば異なる方法で捉える必要があり、AgentSimulator はその両方に自動的に適応可能であることが示された。人的行動のモデリングは複雑な課題である。我々のアプローチはエージェント固有の行動をいくつかうまく捉えているものの、現時点ではマルチタスク、バッチ処理、または疲労効果を考慮していない。これらの要素を今後の研究で組み込むことを計画している。さらに、MAS 自体のための追加のアーキテクチャの活用についても探求していく予定である。

参考文献

[1] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, Fundamentals of Business Process Management. Springer, 2013.

[2] W. M. P. van der Aalst, “Business process simulation survival guide,” in Handbook on Business Process Management 1, Introduction, Methods, and Information Systems, 2nd Ed, ser. International Handbooks on Information Systems, J. vom Brocke and M. Rosemann, Eds. Springer, 2015, pp. 337–370.

[3] A. Rozinat, R. S. Mans, M. Song, and W. M. P. van der Aalst, “Discovering simulation models,” Inf. Syst., vol. 34, no. 3, pp. 305–327, 2009.

[4] M. Camargo, M. Dumas, and O. Gonzalez-Rojas, “Automated discovery of business process simulation

models from event logs," Decision Support Systems, vol. 134, p. 113284, 2020.

[5] F. Meneghello, C. D. Francescomarino, and C. Ghidini, "Runtime integration of machine learning and simulation for business processes," in ICPM. IEEE, 2023.

[6] M. Camargo, M. Dumas, and O. G. Rojas, "Learning accurate business process simulation models from event logs via automated process discovery and deep learning," in CAiSE. Springer, 2022.

[7] I. Khodyrev and S. Popova, "Discrete modeling and simulation of business processes using event logs," in ICCS. Elsevier, 2014.

[8] O. Lopez-Pintado and M. Dumas, "Business process simulation with differentiated resources: Does it make a difference?" in BPM. Springer, 2022.

[9] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach (4th Edition). Pearson, 2020.

[10] M. Song and W. Aalst, "Towards comprehensive support for organizational mining," Decision Support Systems, vol. 46, pp. 300–317, 2008.

[11] O. Lopez-Pintado and M. Dumas, "Discovery and simulation of business processes with probabilistic resource availability calendars," in ICPM. IEEE, 2023.

[12] M. Camargo, M. Dumas, and O. G. Rojas, "Learning accurate LSTM models of business processes," in BPM. Springer, 2019.

[13] N. Tax, I. Verenich, M. L. Rosa, and M. Dumas, "Predictive business process monitoring with LSTM neural networks," in CAiSE. Springer, 2017.

[14] D. Chapela-Campa and M. Dumas, "Enhancing business process simulation models with extraneous activity delays," Information Systems, vol. 122, p. 102346, 2024.

[15] D. Chapela-Campa, I. Benchekroun, O. Baron, M. Dumas, D. Krass, and A. Senderovich, "Can I trust my simulation model? measuring the quality of business process simulation models," in BPM. Springer, 2023.

[16] J. Kazil, D. Masad, and A. T. Crooks, "Utilizing python for agent-based modeling: The mesa framework," vol. 12268. Springer, 2020, pp. 308–317.

[17] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," IEEE Access, vol. 6, pp. 28 573–28 593, 2018.

[18] N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. D. O'Brien, and M. E. Wiegand, "Agent-based business process management," Int. J. Cooperative Inf. Syst., vol. 5, no. 2&3, pp. 105–130, 1996.

[19] A. Tour, A. Polyvyanyy, and A. A. Kalenkova, "Agent system mining: Vision, benefits, and challenges," IEEE Access, vol. 9, pp. 99 480–99 494, 2021.

[20] A. Tour, A. Polyvyanyy, A. A. Kalenkova, and A. Senderovich, "Agent miner: An algorithm for discovering agent systems from event data," in BPM. Springer, 2023.

[21] M. Halaska and R. Sperka, "Is there a need for agent-based modelling and simulation in business process management?" Organizacija, vol. 51, no. 4, pp. 255–269, 2018.

[22] E. Sulis and K. Taveter, Agent-Based Business Process Simulation - A Primer with Applications and Examples. Springer, 2022.

[23] M. Dornhofer, S. Sack, J. Zenkert, and M. Fathi, "Simulation of smart factory processes applying multi-agent-systems—a knowledge management perspective," JMMP, vol. 4, no. 3, 2020.