Towards Reliable Business Process Simulation: A Framework to Integrate ERP Systems

Gyunam Park and Wil M.P. van der Aalst

Process and Data Science Group (PADS), Department of Computer Science,
RWTH Aachen University, Aachen, Germany
{gnpark,wvdaalst}@pads.rwth-aachen.de

Summary. A digital twin of an organization (DTO) is a digital replication of an organization used to analyze weaknesses in business processes and support operational decision-making by simulating different scenarios. As a key enabling technology of DTO, business process simulation provides techniques to design and implement simulation models that replicate real-life business processes. Existing approaches have been focusing on providing highly flexible design tools and data-driven evidence to improve the accuracy of simulation models. Provided with such tools and evidence, business analysts are required to reflect comprehensive aspects of reality with subjective judgments, including the design of ERP systems and the organizational interaction with the system. However, given the complexity of ERP systems, it is infeasible and error-prone to manually incorporate the business logic and data restrictions of the system into simulation models, impeding the faithfulness and reliability of the following analysis. In this work, we propose a framework to integrate ERP systems in business process simulation to overcome this limitation and ensure the reliability of the simulation results. The framework is implemented in *ProM* using the SAP ERP system and CPN Tools.

Key words: Digital Twin, Business Process Simulation, ERP Systems, Business Process Improvement, Process Mining

1 Introduction

A digital twin of an organization (DTO) is a digital representation of business processes and assets across an organization. By simulating this mirrored representation, business analysts can identify operational frictions in the organization with data-based analytics like process mining [1] and evaluate the efficiency of the decisions that are too expensive or dangerous to experiment with, e.g., assigning more resources to a task and increasing the capacity of machines.

Business process simulation is a key enabling technology of DTOs. A simulation model represents reality in a simplified manner and generates hypothetical instances of the business process, enabling the simulation of various scenarios and "what-if" analysis. Many tools have been developed to support the design and implementation of simulation models, including Arena and CPN Tools [2].

The successful implementation of DTOs using business process simulation relies on how accurately the simulation model represents reality. Especially, given the increasing relevance of Enterprise Resource Planning (ERP) systems in organizations, it is essential to accurately model the business logic and data restriction of ERP systems (e.g., SAP, Salesforce, and Microsoft Dynamics).

Traditional simulation approaches left them to the subjective judgments by domain experts, focusing on providing easy-to-use design tools with high flexibility in the implementation. Despite the flexible design tools, it is infeasible and error-prone to design and implement simulation models reflecting the complex design of ERP systems, making the simulation results unconvincing and unreliable [3]. For instance, the SAP ERP system may produce over 2,821 different errors for violations of the business logic and data restrictions.

The violations mainly happen in two formats: *omission* and *commission* [3]. The former occurs when simulation models fail to reflect the behaviors required by the system. For instance, in the SAP ERP system, each material has a different warehouse management policy, thus having different control flows (i.e., sequences of tasks). Although different control flows should be modeled depending on the policy, simulation models often omit it, resulting in a huge difference in the simulated behaviors. On the other hand, the *commission* problem occurs when simulation models simulate the behaviors not allowed in the system. For instance, ERP systems are established upon data restrictions, limiting arbitrary creations, deletions, and updates of data. For instance, price changes (i.e., updates of price information) are not allowed for specific types of items, even though simulation models often ignore this data restriction.

In this work, we propose a framework to integrate ERP systems to business process simulation to 1) effectively incorporate the complex design of the system into simulations without modeling it in simulation models and 2) provide simulated event data without the omission and commission issues. Figure 1 conceptually describes how the proposed framework integrates ERP systems into business process simulation and how it realizes DTOs.

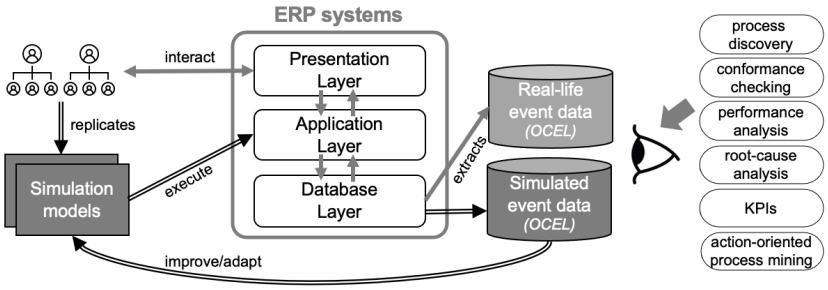

Fig. 1: A conceptual overview of integrating ERP systems in business process simulation

As shown in Figure 1, ERP systems consist of three layers: representation, application, and database layers. In reality, human resources interact with the system through the representation layer. The commands by human resources are

processed according to the business logic defined in the application layer, whose records are stored in the database layer. Event data, e.g., *Object-Centric Event Logs (OCEL)*[1], are extracted from the database layer and used for analysis, e.g., process discovery and root-cause analysis.

In the proposed framework, simulation models, designed by analyzing interactions in the presentation layer, produce commands replicating the behaviors of human resources. We execute the commands directly in the application layer of the ERP system, ensuring the non-existence of the omission and commission problems. The simulated event data are extracted from the database layer in the same manner that real-life event data are extracted. The simulated data may provide feedback to improve the design of the simulation model and adapt the model to changing circumstances.

Since the simulated and real-life data are extracted from the same database in the same manner, they can be analyzed using the same process mining tool. Besides, they support action-oriented process mining [4], which transforms insights from diagnostics to actions, by monitoring the undesired behaviors using real-life data and evaluating the effects of actions using simulated data (e.g., A/B testing between different actions).

To show the effectiveness of the proposed framework, we implement it using the SAP ERP system and CPN Tools. We have tested the implementation by simulating common scenarios in the order-to-cash process of the SAP ERP system and extracting process-centric insights from simulated event data.

The remainder is organized as follows. We discuss the related work in Sect. 2. Next, we present the framework for integrating ERP systems into business process simulation and its implementation in Sect. 3 and Sect. 4, respectively. Sect. 5 presents the validation of the framework and Sect. 6 concludes the paper.

2 Related Work

Simulation has been adopted for analyzing business processes since the seventies [5]. Nowadays, various simulation tools are available to design and implement simulation models. In [6], the simulation tools are classified into two types: *simulation language* and *simulation package*. The former is a programming language supporting the implementation of simulation models, including Simula and GPSS [7]. The latter is a tool providing graphical building blocks to enable the rapid creation of simulation models such as ARENA and CPN Tools [8].

Data-driven approaches have been proposed to provide conceptual guidance to design simulation models. Martin et al. [9] identify modeling tasks (e.g., modeling gateways, modeling activities, etc.) and present the relevant process mining techniques to support each of the modeling tasks. In [10], authors utilize the process model discovered using process mining techniques as the reference for the current state of the process and design simulation models with re-engineered

[1] `http://ocel-standard.org/`

business processes by manually identifying the possible improvement points from the reference model.

Furthermore, several techniques have been suggested to automatically discover simulation models from event data. Rozinat et al. [11] discover simulation models based on Colored Petri Nets (CPNs) in a semi-automated manner. Carmargo et al. [12] propose a method to optimize the accuracy of the simulation models discovered from an event log. It searches the space of all possible configurations in the simulation model to maximize the similarity between the discovered simulation model and the event log.

The existing approaches have focused on improving the accuracy by supporting business analysts to better reflect reality in simulation models using domain knowledge and data-driven insights. However, the question still remains: is the simulated behavior the one that can be supported (accepted) by the underlying ERP systems? Since the approaches do not explicitly involve the system in simulations, business analysts are left to ensure it with their subjective judgments. In this work, we tackle this question by proposing a framework to integrate ERP systems in the course of business process simulation.

3 A Framework for the integration of ERP systems

The proposed framework consists of three major components: *ERP system*, *simulation engine* and *transformation engine*. The ERP system creates, updates, and deletes objects in *object model* based on *executions*. The simulation engine simulates organizational behaviors in business processes and produces *commands* describing the behaviors. The transformation engine translates the commands into the executions according to which the system updates the object model. The behavior of the simulation may again depend on the updated object model. In the following, we explain the framework with formal definitions and examples.

3.1 ERP System

In this work, we define ERP systems in a narrow sense, focusing on its bookkeeping purpose (i.e., creating, updating, and deleting database tables based on transactions). To this end, we first abstract databases in the system as *object models*.

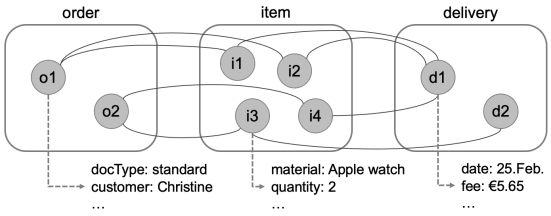

Fig. 2: We use object models to mimic the databases in ERP systems

As described in Figure 2, an object model contains objects (e.g., *o1*, *o2*, and *i1*) with different types (e.g., *order*, *item*, and *delivery*). Also, the objects have relationships (e.g., *o1* is related to *i1*). Besides, each object involves attribute values (e.g., order has *docType* and *customer* information). We formally define object models as follows:

Definition 1 (Object Model). *Let $\mathbb{U}_o$ be the universe of object identifiers, $\mathbb{U}_{ot}$ the universe of object types, $\mathbb{U}_{attr}$ the universe of attribute names, and $\mathbb{U}_{val}$ is the universe of attribute values. An object model is a tuple $OM = (O, OT, OR, otyp, oval)$ where*

- *$O \subseteq \mathbb{U}_o$ is a set of object identifiers,*
- *$OT \subseteq \mathbb{U}_{ot}$ is a set of object types,*
- *$OR \subseteq O \times O$ is a set of object relationships,*
- *$otyp \in O \to OT$ assigns precisely one object type to each object identifier, and*
- *$oval : O \to (\mathbb{U}_{attr} \nrightarrow \mathbb{U}_{val})$ is the function associating an object to its attribute value assignments. We denote $oval(o)(attr) = \perp$ if $attr \notin dom(oval(o))$ for any $o \in O$.*

$\mathbb{U}_{om}$ denote the set of all possible object models.

For instance, $OM_1 = (O, OT, OR, otyp, oval)$ describes the object model described in Figure 2, where $O = \{o_1, o_2, i_1, \dots\}$, $OT = \{order, item, delivery\}$, $OR = \{(o_1, i_1), (o_1, i_2), (i1, d1), \dots\}$, $otyp(o_1) = order$, $otyp(i_1) = item$, $oval(o_1)(docType) = standard$, $oval(o_1)(customer) = Christine$, etc.

Definition 2 (Transaction). *Let $\mathbb{U}_{pval} \subseteq \mathbb{U}_{attr} \nrightarrow \mathbb{U}_{val}$ be the universe of parameter value assignments. A transaction $tr \in \mathbb{U}_{om} \times \mathbb{U}_{pval} \to \mathbb{U}_{om}$ is a function that modifies object models based on parameter value assignments. $\mathbb{U}_{tr}$ denotes the set of all possible transactions.*

For instance, $po \in \mathbb{U}_{tr}$ is a transaction that places an order. Assume that $pval_1 \in \mathbb{U}_{pval}$ is a parameter value assignment such that $pval_1(customer) = Marco$, $pval_1(docType) = quoted$, $pval_1(item) = iPad$, $pval_1(quantity) = 1$, etc. $po(OM_1, pval_1) = (O', OT, OR', otyp', oval')$ such that $o_3, i_5 \in O'$, $(o_3, i_5) \in OR'$, $otyp'(o_3) = order$, $otyp'(i_5) = item$, $oval'(o_3)(customer) = Marco$, etc.

An execution specifies a transaction to be executed in the system along with the execution time, responsible resource, and parameter value assignment.

Definition 3 (Execution). *Let $\mathbb{U}_{res}$ be the universe of resources and $\mathbb{U}_{time}$ the universe of timestamps. An execution $exec \in \mathbb{U}_{tr} \times \mathbb{U}_{res} \times \mathbb{U}_{time} \times \mathbb{U}_{pval}$ is a tuple of a transaction, a resource, timestamp, and a parameter value assignment. $\mathbb{U}_{exec}$ denotes the set of all possible executions.*

For instance, $exec_1 = (po, Adams, \text{10:00 23.02.2021}, pval_1)$ describes the order placement by *Adams* at 10:00 23.02.2021 with the parameter $pval_1$.

Definition 4 (ERP System). *An ERP system $sys \in \mathbb{U}_{om} \times \mathbb{U}_{exec} \to \mathbb{U}_{om}$ updates object models according to executions.*

For instance, $sys(OM_1, exec_1) = (O', OT, OR', otyp', oval')$ such that $o_3, i_5 \in O'$, $oval'(o_3)(timestamp) = \text{10:00 23.02.2021}$, $oval'(o_3)(resource) = Adams$, etc.

3.2 Simulation Engine

A simulation engine aims at producing commands describing which activity is done by whom at what time with what information. We formally define commands as follows:

Definition 5 (Commands). *Let $\mathbb{U}_{act}$ be the universe of activities. A command $cmd \in \mathbb{U}_{act} \times \mathbb{U}_{res} \times \mathbb{U}_{time} \times \mathbb{U}_{pval}$ is a tuple of an activity, resource, timestamp, and information value assignment. $\mathbb{U}_{cmd}$ is the set of all possible commands.*

For instance, $cmd_1 = (place\_order, Adams, 10{:}00\ 23.02.2021, ival_1) \in \mathbb{U}_{cmd}$, where $ival_1(customer) = Marco$ and $ival_1(price) = €100$, clones the behavior of *Adams* who places an order of €100 by *Marco* at 10:00 23.02.2021.

Since simulating behaviors of human resources (i.e., generating commands) share commonalities to simulating events of business processes (e.g., placing order occurred at 10:00 23.02.2021 by *Adams*), we can deploy techniques for business process simulation to generate commands.

As presented in Sect. 2, various simulation tools are available for this purpose such as CPN Tools and ARENA. Remaining tool-independent, we focus on explaining essential components of simulation models to produce commands. In Sect. 4, we explain how these components are designed and implemented using CPN Tools.

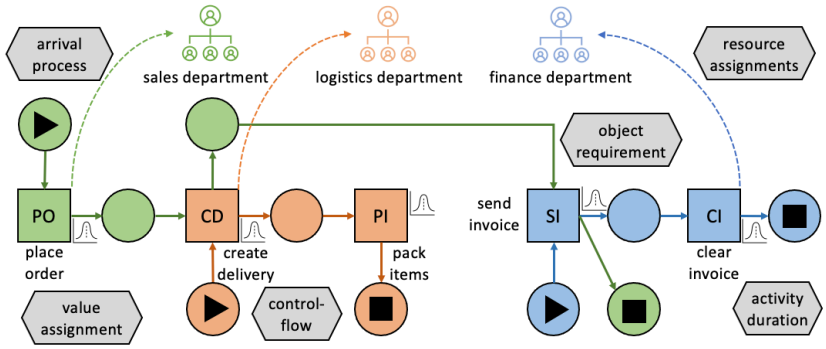

Fig. 3: Core components of business process simulation [6]

Figure 3 explains the core ingredients of business process simulation including *control-flows*, *resource assignments*, *arrival processes*, and *activity duration*, *object assignments* and *object value assignments*. Below is the explanation of each component:

- **Activities** represent the behaviors that human resources do to serve business processes (e.g., place order, send invoice, etc.).
- **Control-flows** determine the sequence of activities. For instance, the process model in Figure 3 describes one sequence of activities, i.e., *send_quotation*,

place_order, *create_delivery*, *pack_items*, *create_invoice*, and *clear_invoice* in order.
- **Object requirements** explain the required objects for the execution of activities. Each activity may involve multiple objects in its execution. For instance, the execution of *create_delivery* involve an order and a delivery since the order information is required to create the delivery.
- **Value assignment** specifies how the execution of activities updates information of the involved objects. For instance, the execution of *place_order* updates the document and customer information of the order.
- **Resource assignments** define who is eligible to perform activities in the business process. For instance, *place_order* can be performed by the resources from a sales department.
- **Arrival processes** and **activity duration** define the inter-arrival time between arrivals and the duration required for the execution of activities, respectively.

3.3 Transformation Engine

The goal of the transformation engine is to convert the commands to the executable formats supported in the underlying ERP system (i.e., executions). To this end, we need two components: *transaction mapping* and *parameter mapping*. First, the transaction mapping translates the activity in commands to the transaction defined in ERP systems.

Definition 6 (Transaction Mapping). *A transaction mapping* $\mu_{tr} \in \mathbb{U}_{act} \rightarrow \mathbb{U}_{tr}$ *relates transactions to activities.*

Assume *place_order* in cmd_1 corresponds to $po \in \mathbb{U}_{tr}$ in an ERP system. A transaction mapping μ'_{tr} connects them, i.e., $\mu'_{tr}(place\_order) = po$.

Next, parameter mapping connects the parameters in commands to the system-defined parameters in ERP systems.

Definition 7 (Parameter Mapping). *A parameter mapping* $\mu_{pr} \in \mathbb{U}_{attr} \rightarrow \mathbb{U}_{attr}$ *relates system-defined parameters of an ERP system to parameters in commands.*

Assume $dom(ival_1) = \{docType, customer\}$ where $(place\_order, Adams, t_1, ival_1) \in \mathbb{U}_{cmd}$ and DOC_TYPE and PART_NUMB are the corresponding parameters defined in the system. A parameter mapping μ'_{pr} connects the parameters, i.e., $\mu'_{pr}(\text{docType}) = \text{DOC\_TYPE}$ and $\mu'_{pr}(\text{customer}) = \text{PART\_NUMB}$.

Given transaction and parameter mapping, a transformation engine transforms commands to executions by translating transactions and parameters. produces the executions. Note that we assume that the resource/timestamp in commands is compatible with the one in executions.

Definition 8 (Transformation Engine). *Let* μ_{tr} *be a transaction mapping and* μ_{pr} *a parameter mapping. A transformation engine maps executions onto commands, i.e., for any* $cmd=(act, res, time, ival) \in \mathbb{U}_{cmd}$, $trans(\mu_{tr}, \mu_{pr})(cmd) = (\mu_{tr}(act), res, time, pval)$ *s.t.* $\forall_{attr \in dom(ival)}\ pval(\mu_{pr}(attr)) = ival(attr)$.

4 Implementation

In this section, we implement the proposed framework using the SAP ERP system as the underlying ERP system. We design and implement simulation models using *CPN Tools*[2]. The transformation engine is implemented as a plug-in in *ProM*[3] and translates commands into executions in the SAP ERP system given a transaction and parameter mapping.

4.1 ERP system: *SAP ERP ECC 6.0*

The SAP ERP system is the most widely-adopted ERP system, supporting more than 400,000 businesses worldwide. In the implementation, we utilize *SAP ERP ECC 6.0*[4] supporting Global Bike Inc., an imaginary enterprise producing and distributing bicycle products where all relevant SAP solutions are represented.

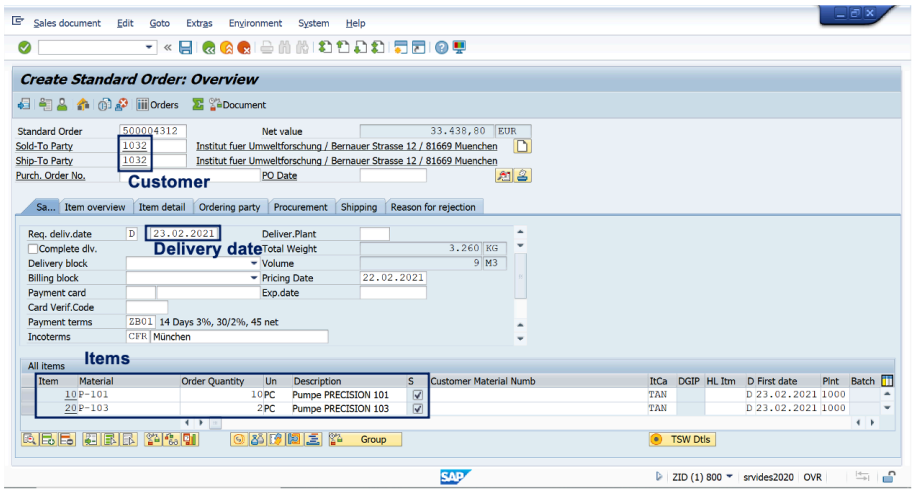

Fig. 4: A screenshot of the user interface in the SAP ERP ECC 6.0 (place an order)

Figure 4 shows the user interface in the representation layer where sales staff places orders. Given the inputs such as customer, delivery date, and items (i.e., parameters) by users (i.e., resources), the transactions in the application layer (e.g., BAPI_SALESORDER_CREATEFROMDAT2) are executed to update the database supported by Oracle Database Management System (i.e., object models).

[2] https://cpntools.org/
[3] http://www.promtools.org
[4] https://www.sap.com/

4.2 Simulation Engine: *CPN Tools*

CPN Tools is a toolset providing support for editing, simulating, and analyzing Colored Petri Nets (CPNs). For the detailed information of *CPN Tools*, we refer readers to [8].

In the following, we explain how *CPN Tools* is used to implement the core ingredients of business process simulation introduced in Subsect. 3.2 with the example described in Figure 3. Note that there exist various design choices resulting in different executable models in *CPN Tools*.

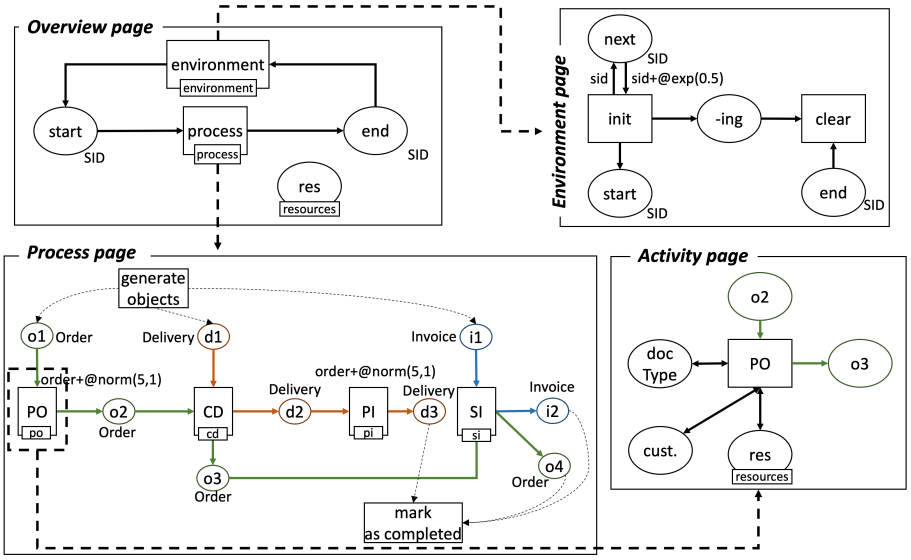

Fig. 5: A schematic overview of the CPN used to implement our simulation framework

Figure 5 shows the overall composition consisting of multiple CPN pages, i.e., *overview page*, *environment page*, *process page*, and *activity pages*.

The overview page connects the environment page, process page, and resource pool. The environment page describes the **arrival process** implemented as a negative exponential distribution. A simulation instance is generated according to the arrival process and passed into the process page.

In the process page, relevant objects for the simulation instance are generated by transition "generate object". In our example, each simulation instance associates an order, a delivery, and an invoice. The transitions, including "PO", "CD", "PI", and "SI", represent **activities** in the process.

The **object requirement** for the execution of an activity is indicated with the incoming arcs from the places representing object types to the corresponding transition. For instance, "create delivery" involves an order and a delivery (i.e.,

incoming arcs from the place for order type (i.e., o_1) and the place for delivery (i.e., d_1) to "create delivery").

Control-flows are represented using the semantics of CPNs, i.e., a transition is triggered by consuming tokens from its input places and producing tokens to its output places. In our example, *place_order* is triggered first by consuming a token at o_1 and producing a token at o_2. Next, "create delivery" is triggered by consuming tokens from o_2 and d_1 and producing tokens at o_3 and d_2.

Each transition has a sub-page (i.e., activity page) where **resource assignments** and **value mappings** are modeled. First, in each execution of the transition, a resource is selected from the resource pool based on the role. Next, the relevant information for the execution of the activity (e.g., the customer and document type in *place_order*) is passed by the tokens from the connected places.

Activity duration is implemented as the timed property of *CPN Tools*. The activity duration is inscribed on the transition. For instance, the duration of the *place_order* activity is populated from a normal distribution.

We generate commands using the designed CPNs. Below is an example of the commands in XML-based *CMD* formats. In $(act, res, time, ival) \in \mathbb{U}_{cmd}$, Lines 4-6 correspond to act, res, and $time$, while Lines 7-12 specify $ival$.

Listing 1: An example of *CMD* format

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <commands>
3      <command>
4          <activity>place_order</activity>
5          <resource>Adams</resource>
6          <timestamp>2021-02-23 10:00:00</timestamp>
7          <orderId>500004312</orderId>
8          <customer>1032</customer>
9          <docType>TA</docType>
10         <salesOrg>1000</salesOrg>
11         <materialList>P-101,P-103</materialList>
12         <quantityList>6,5</quantityList>
13     </command>
14     ...
15 </commands>
```

4.3 Transformation Engine: ProM plug-in

The transformation engine is implemented as a plug-in of *ProM*, an open-source framework for the implementation of process mining tools. Our new plug-in is available in a new package named *ERPSimulator* in the nightly build of *ProM*. The main input objects of the plug-in are transaction mapping, parameter mapping, and commands, whereas the outputs are SAP executions.

The transaction mapping is stored as an XML-based *AMAP* format, storing relations between activities and transactions. Below is an example of the transaction mapping for the commands generated by the simulation engine described in Figure 5. Four activities in the simulation engine are assigned to corresponding transactions defined in the SAP ERP system.

Listing 2: An example of *AMAP* format

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <transactionMapping>
3     <string key="place_order" value="BAPI_SALESORDER_CREATEFROMDAT2"/>
4     <string key="create_delivery" value="BAPI_OUTB_DELIVERY_CREATE_SLS
          "/>
5     <string key="pack_items" value="L_TO_CREATE_DN"/>
6     <string key="create_invoice" value="BAPI_BILLINGDOC_CREATEMULTIPLE
          "/>
7 </transactionMapping>
```

The parameter mapping is stored as an XML-based *PMAP* format, storing relations between the parameter in commands and the system-defined parameter. Below is an example of the parameter mapping for the commands produced by the simulation engine described in Figure 5. In Line 3-4, "docType" and "customer" are matched into DOC_TYPE and PARTN_NUMB in the SAP ERP system.

Listing 3: An example of *PMAP* format

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <parameterMapping>
3     <string key="docType" value="DOC_TYPE"/>
4     <string key="customer" value="PARTN_NUMB"/>
5     <string key="orderId" value="SALESDOCUMENTIN"/>
6     <string key="salesOrg" value="SALES_ORG"/>
7     ...
8 </parameterMapping>
```

Given the transaction and parameter mapping, the transformation engine translates commands in *CMD* format into SAP Remote Function Calls (RFCs) that can be directly executed in the system. The SAP RFC is an SAP interface protocol managing the communication between the SAP ERP system and any external system. For instance, the command in Listing 1 is translated to the RFC specifying BAPI_SALESORDER_CREATEFROMDAT2, as defined in Listing 2, with the parameters such as DOC_TYPE and PARTN_NUMB, as described in Listing 3.

5 Proof of concept

In this section, we validate the feasibility of the proposed framework in generating simulated event data that contain no omission and commission problems (i.e., reliable) and have the same view as the real-life event data (i.e., realistic) without having to manually model the complex design of ERP systems in simulation models. To this end, we simulate common business challenges in the order-to-cash (O2C) process using the implementation presented in Sect. 4. The CPN files and commands are publicly available via `https://github.com/gyunamister/ERPSimulator`, as well as the user manual.

5.1 Experimental Design

The O2C process deals with customer orders. First, customers send inquiries and, in return, the company sends corresponding quotations. Sales staff converts the

quotations into orders if the customers confirm them. Afterward, deliveries are prepared by picking up and packing the items of the orders. Next, invoices are sent to customers and the corresponding payments are collected.

In the following, we simulate common business challenges in the O2C process using the implementation presented in Sect. 4, i.e., 1) low conversion rate, 2) frequent price change, and 3) order cancellation. In each scenario, we evaluate if simulated data have omission and commission problems by measuring the number of executions with errors using the error handling module in the SAP system. Besides, we apply process mining techniques, such as process discovery and root-cause analysis, to simulated event data to show that they have the same view as real-life event data, containing insightful knowledge.

5.2 Scenario 1: Low Conversion Rate

The low conversion rate from quotations to orders is not desired because not only of the lost revenue but also of the waste of resources. We design and implement the simulation model where quotations are not converted to orders mostly due to the late response to the corresponding inquiry using *CPN Tools*. 288 commands are generated by the simulation model and transformed into 288 RFCs using the *ProM* plug-in. Among the 288 RFCs, 288 are successfully executed in the system. As a result, 286 objects of 8 different object types, including inquiry, quotation, order, etc., are created, updating more than 11 tables in the database.

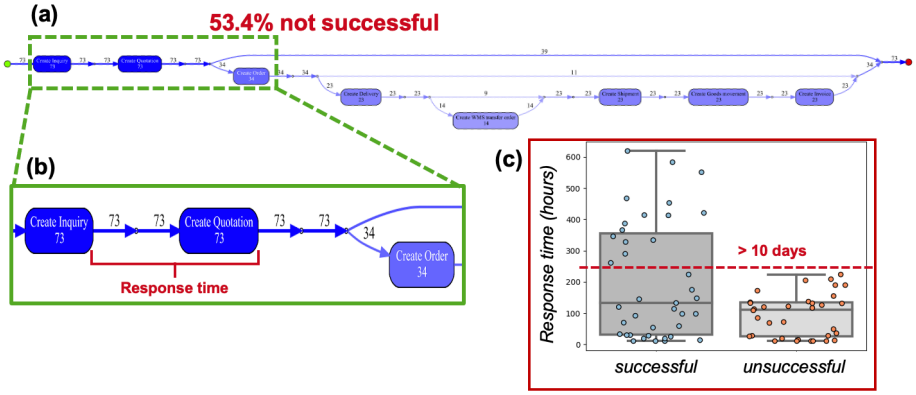

Fig. 6: (a) a discovered process model of the O2C process in BPMN notation, (b) a low conversion from quotations to orders (53.4%), (c) a box plot showing the correlation between the response time and (un)successful conversions

We analyze the behaviors in the business process using the Inductive Visual Miner in *ProM* [13]. Figure 6-(a) describes the process model in BPMN notation. As shown in Figure 6-(b), only 34 out of 73 quotations from the company are converted to orders, showing the conversion rate of 46.6%. We define the response time as the time difference between the completion of "create inquiry" and "create quotation". Figure 6-(c) shows the difference in the response time

between the successful and unsuccessful confirmed orders. Especially, the quotations that are responded to later than 10 days are all rejected, showing the correlation between the response time and unsuccessful conversion.

5.3 Scenario 2: Manual Price Changes

Due to different reasons (e.g., outdated pricing policy in the system), manual changes in prices are carried out. We design this scenario in *CPN Tools* and produce 4,249 commands that are transformed into 4,249 RFCs by the *ProM* plug-in. All of the 4,249 RFCs are successfully executed in the system without errors, creating 4,093 objects and updating more than 15 tables in the database.

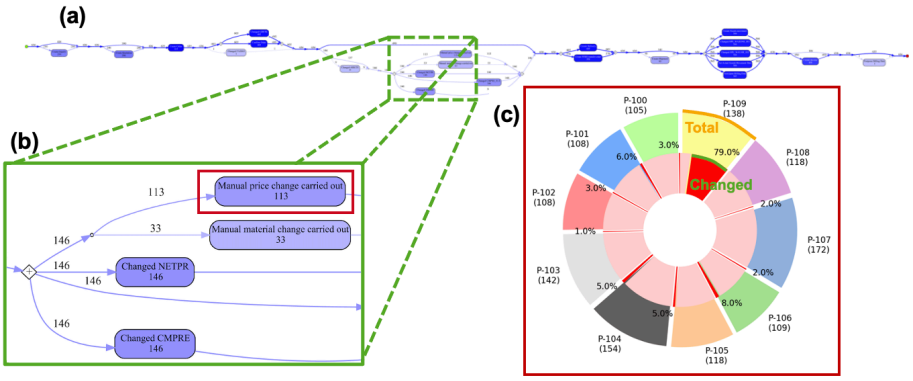

Fig. 7: (a) a discovered process model, (b) manual price changes required for 113 orders, (c) a pie chart describing the ratio of price changes to total orders per products

Figure 7-(a) depicts the process model discovered using the process discovery technique. As shown in Figure 7-(b), for 113 orders out of 402 orders, the manual price change occurred. Figure 7-(c) describes manual price changes per product (e.g., *P-100* and *P-101*). The outer pie indicates the total number of orders per product, while the red part in the inner pie represents the proportion of the changes for each product. For instance, *P-109* is included in 138 orders and 79% of them require manual changes.

5.4 Scenario 3: frequent cancellation of orders

For the frequent cancellation of orders, we first generate 4,540 commands using *CPN Tools* and transform them into 4,540 SAP RFCs using the *ProM* plug-in. We successfully execute the 4,540 RFCs without errors and, accordingly, 4,384 objects of 8 different object types are created.

Figure 8 shows the process model discovered with the process discovery technique. As shown in Figure 8-(b), 97 out of 562 orders are canceled in the process. We conduct further analysis on these canceled orders by analyzing the reasons for the cancellation. Figure 8-(c) shows the pie chart explaining the proportion of different reasons. The most frequent reason is the delivery date set too late.

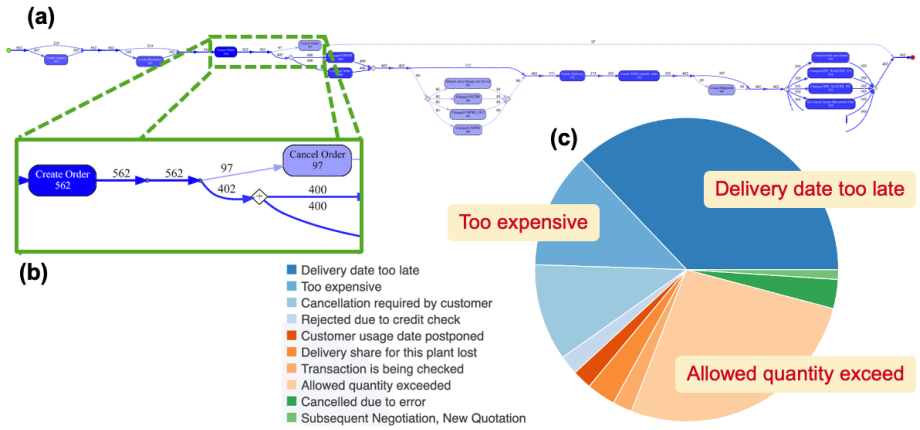

Fig. 8: (a) a discovered process model, (b) order cancellations (97 out of 462 orders), (c) a pie chart depicting the frequency of reasons for the order cancellation

The second most frequent reason is that the order exceeds the quantity limit of one of its items, followed by the high price of the order.

6 Conclusion

In this paper, we proposed the framework for integrating ERP systems into business process simulation to realize DTOs. The framework is composed of three components: the ERP system, simulation engine, and transformation engine. Commands are generated by the simulation engine and translated to system-executable formats by the transformation engine. The executions are applied to the system to update the object model in the system. The framework is implemented using the SAP ERP system as the underlying ERP system, CPN Tools as the simulation engine, and a ProM plug-in as the transformation engine.

By integrating ERP systems, we can effectively reflect the complex design of the system into simulation models. Moreover, the resulting simulated data have no omission and commission issues, ensuring the reliability of simulation results. Also, having the same data structure as the real-life event data, the simulated event data can be analyzed by existing analysis techniques. Furthermore, it supports action-oriented process mining by providing a digital twin where different actions can be implemented and tested.

As future work, we plan to improve the implementation to support the feedback loop between the simulated data and simulation engine. In addition, we plan to deploy the implementation to the techniques for action-oriented process mining to evaluate the efficiency of actions. In the proposed framework, we assume a one-to-one relationship between activities (i.e., human behaviors) and transactions. However, in real-life business processes, a single human behavior may involve multiple transactions and vice versa. In this work, we resolve the

issue in the implementation by manually aligning the level of simulated human behaviors to the level of transactions. Future work should present a method to support the resolution of the many-to-many relationship.

Acknowledgements We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

1. van der Aalst, W.M.P.: Data Science in Action. In: Process Mining. Springer, Heidelberg (2016)
2. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer Berlin Heidelberg (2018)
3. Rashid, A., Tjahjono, B.: Achieving manufacturing excellence through the integration of enterprise systems and simulation. Production Planning & Control **27**(10) (2016) 837–852
4. Park, G., van der Aalst, W.M.P.: A general framework for action-oriented process mining. In Del Río Ortega, A., et al., eds.: Business Process Management Workshops, Springer International Publishing (2020) 206–218
5. Shannon, R., Johannes, J.D.: Systems simulation: The art and science. IEEE Transactions on Systems, Man, and Cybernetics **SMC-6**(10) (1976) 723–724
6. van der Aalst, W.M.P.: Business process simulation survival guide. In vom Brocke, J., Rosemann, M., eds.: Handbook on Business Process Management. Springer Berlin Heidelberg (2015) 337–370
7. Dahl, O.J., Nygaard, K.: Simula: An algol-based simulation language. Commun. ACM **9**(9) (September 1966) 671678
8. Jensen, K., Kristensen, L.M., Wells, L.: Coloured petri nets and CPN tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer **9**(3) (2007) 213–254
9. Martin, N., Depaire, B., Caris, A.: The use of process mining in business process simulation model construction: Structuring the field. Business & Information Systems Engineering **58**(1) (2016) 73–87
10. Mruter, L., van Beest, N.R.T.P.: Redesigning business processes: a methodology based on simulation and process mining techniques. Knowledge and Information Systems **21**(3) (2009-12) 267–297
11. Rozinat, A., Mans, R., Song, M., van der Aalst, W.: Discovering simulation models. Information Systems **34**(3) (2009) 305–327
12. Camargo, M., Dumas, M., González-Rojas, O.: Automated discovery of business process simulation models from event logs. Decision Support Systems **134** (2020) 113284
13. Leemans, S., Fahland, D., Aalst, van der, W.: Process and deviation exploration with inductive visual miner. In Limonad, L., Weber, B., eds.: BPM Demo Sessions. (2014) 46–50