National Research University Higher School of Economics

as a manuscript

Julio Cesar Carrasquel Gamez

Formal Modeling and Validation of Stock Trading Systems Behavior: A Petri Net Approach

PhD Dissertation Summary

for the purpose of obtaining the academic degree
Doctor of Philosophy in Computer Science

Moscow — 2022

The **PhD Dissertation was prepared at** National Research University Higher School of Economics.

Academic Supervisor: Irina A. Lomazova, Doctor of Science, Professor, National Research University Higher School of Economics.

Abstract

Trading systems are software platforms used in financial markets to support the exchange of financial instruments between participants. Particularly, this thesis considers trading systems operating in *stock exchanges*, where participants submit orders to buy and sell securities (e.g., shares of companies). Given the pivotal role of stock trading systems in global finances, it becomes a task of utmost importance to guarantee the correct execution of their processes. This is why experts in this domain constantly look for novel ways to validate such systems, including approaches that can detect differences between a real platform and its specification.

In this thesis, Petri nets and process mining have been chosen for modeling and validating the behavior of trading systems in stock exchanges. *Petri nets* are a well-known formalism for modeling concurrent distributed systems. Also, there exist different *extensions of Petri nets* which allow to describe multiple aspects such as data, resources, interaction of agents, etc. *Process mining* is a data science discipline that focuses on the analysis of processes in information systems on the basis of *event logs*, which record the real behavior of such processes. Notably, *conformance checking* is family of process mining techniques used to compare the real behavior of processes (as observed in event logs) with their expected behavior modeled by Petri nets. Experts have acknowledged the potential of Petri nets and process mining methods on trading systems: one can build a Petri net specifying the expected behavior of a process in a trading system, and use conformance checking to compare the model with an event log from such a system in order to detect errors. This approach may support engineers on tasks such as maintenance and improvement of a trading platform. Yet, the usage of Petri nets and process mining in this domain had not been considered before, and thus this thesis aims to solve the question on how to properly model and analyze crucial behavioral aspects of trading systems using these techniques.

As the *main research contributions*, this thesis first provides a study on how extensions of Petri nets can be used to describe and analyze processes in trading systems. Rather than providing a universal model, we present how each of the chosen extensions can be used to build a model of a certain process, while focusing on a precise aspect to validate for such a process (for example, the management of data or the interaction between agents). Also, *novel conformance checking methods* that make use of Petri net extensions were developed to validate whether real processes in trading systems comply with their specification models. As output, these methods indicate "deviations", that is, concrete differences where a real system (as observed in an event log) violates its specification. To showcase the practical value of this research, the proposed methods were implemented as a software tool and we report their usage for validating a real trading platform. Experimental evaluations with artificial data are also reported.

Dissertation Summary

The thesis presents a study on the formal modeling of processes in trading systems using different extensions of *Petri nets* — a well-known formalism for modeling and analysis of concurrent distributed systems [33]. Novel methods to validate the real behavior of trading systems by comparing models of Petri net extensions using event data from these systems are also presented. In this dissertation summary, we first introduce what are trading systems and how *process mining*, a data science discipline that focuses on the analysis of processes [3], can be used to validate these systems. Also, we introduce and motivate our choice to employ Petri nets and *conformance checking*, a kind of process mining techniques [7], as the specific approaches for modeling and validation of processes in trading systems. Afterwards, we define the research problem, the main research contributions, and finally we outline the structure of this thesis.

Research Background

Trading Systems

Trading systems are software platforms used in financial markets to support the exchange of financial instruments [19]. The kind of such instruments depends on the market type that a trading system works with [36]. For instance, in *commodity markets* participants trade primary goods ranging from cocoa to gold and oil. In *foreign exchange* markets, people trade currencies. In this thesis, we consider trading systems in *stock exchanges*, where participants submit orders to buy and sell securities. A *security* is an asset that gives to its holder an ownership right over a share of a company. The size of a share is measured by the number of stocks associated to the company's security. *Orders* submitted by participants indicate what security they want to buy or sell, how many stocks of such security, and what price per stock. Trading systems benefit buyers and sellers. Companies sell their shares to raise capital for business development. Investors seek to buy securities from companies with promising growth in such a way their investment may give favorable returns. These are some of the benefits that have turned trading systems into a vital component of global finances [25].

Modern stock exchanges rely their operability on software platforms, but their roots can be traced back as early as 1602, when the Amsterdam Stock Exchange was created by the Dutch East India Company [39]. The New York Stock Exchange founded in 1792 and the London Stock Exchange founded in 1801 are yet another cases of long-established trading venues. Elements such as securities, orders, and order books that are used to store orders, emerged in those venues and nowadays compose the core of trading systems. The migration from manual transactions to the use of electronic trading systems began in the 1970s. A long renowned case is the National Association of Securities Dealers Automated Quotations (NASDAQ), created in 1971 and known as the world's first electronic trading platform [34]. In the last decades, the use of modern software technologies in trading systems has enabled an automated control over transactions, providing reliable services to participants. Also, the digital transformation of trading systems has diversified the kinds of participants and services in trading systems. For example, *trading robots* or *algorithmic trading software* can be configured to analyze real-time market data and to automatically buy/sell securities based on their analysis [4, 35].

Given the pivotal role of trading systems in global finances, it becomes crucial the assurance of their *correctness*; this means that *software processes* in trading systems must behave as intended. Generally, the expected behavior of software processes is documented in *system specifications*, which serve as a guide for engineers when developing trading systems. However, the increase of the number and the variety of participants, as well as the growing complexity of services turns the specification-driven development of trading systems a complex and error-prone task. Errors during a system's development may provoke the real behavior of software processes to differ from the expected behavior described in the system's specification. Such a gap between a real system and its specification may provoke failures with disastrous consequences for stock exchanges and their participants [26]. In 2012, a software error of the trading company Knight Capital led to an uncontrolled submission of orders, accumulating a loss position of $440 million within 45 minutes [37]. In the same year, NASDAQ was fined $10 million due to technical malfunctions during the initial public offering of Facebook's shares [38]. To avoid errors of such magnitude, experts constantly seek for novel ways to *validate* trading systems, that is, approaches that can opportunely uncover differences between a system and its specification [6, 22, 41].

When evaluating new approaches for validating trading systems, experts also consider the computational overhead they may cause. Instrumentation placed in a system for monitoring its behavior may affect the system's overall performance. The time for serving user requests may increase. This is why approaches that diagnose a system from the outside are preferred. In this regard, *passive analysis* (also known as *passive testing*) has gained the attention of experts [23]. Instead of interacting directly with a system, passive analysis tools read system outputs such as logs or network messages. This output is processed by *data science* methods such as machine learning or data mining to get insights about the system's behavior. For example, in [22] logs produced by *settlement and clearing* components of a trading system are diagnosed by text analysis and clustering techniques. Thus, experts aim that passive analysis grounded on data science can support engineers to detect errors in processes of trading systems.

Process Mining

A promising data science approach for validating trading systems is *process mining*. Process mining is a data science discipline that focuses on the analysis of processes in information systems [3]. When analyzing a process in an information system, process mining makes use of *event logs* and *process models*. An event log records *real behavior* of a process, and it consists of cases, each of them associated to a particular run of the process (also known as a "process instance"). Figure 1 presents the main types of methods in process mining: *conformance checking*, *process discovery*, and *enhancement*. Conformance checking seeks to compare models describing *expected behavior* of processes (for example, based on a system's specification) against event logs. The latter allows to validate if real processes (as observed in the logs) behave as the models specify. Conversely, process discovery uses event logs as input in order to automatically build models that describe real processes. *Enhancement* techniques allow to enrich process models, either hand-made or discovered a priori, by using information available in event logs.

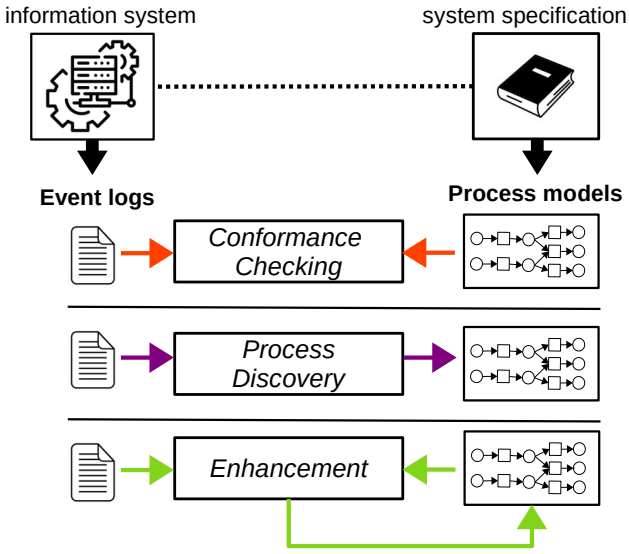

Figure 1: Methods in process mining: *discovery*, *conformance*, and *enhancement*.

Process mining methods particularly focus on the *control-flow* perspective of processes, i.e., activities that a process executes and their causal dependence. For example, "the execution of an activity `trade` must be preceded by an activity `order submission`". Process models allow to capture different control-flow patterns such as sequences, choices, parallel executions, and loops. Nowadays, a plethora of process modeling notations are available: Petri nets [33], UML Activity Diagrams [14], Business Process Modeling Notation (BPMN) [47] and Event-driven Process Chains (EPCs) [31], among many others.

A traditional application domain for process mining has been Business Process Management (BPM) [2]. Various organizations such as Siemens, Uber and BMW have used process mining to improve their business processes in areas ranging from finances and customer support to manufacture [43]. Nonetheless, process mining have been also applied in innovative case studies outside the sphere of BPM, such as human habits in smart spaces [27], delays in railway networks [29], and gaming user behavior [42].

Notably, the use of process mining to analyze *software processes* in computer systems is an area which has gained attention by researchers and practitioners [45, 46]. In [45], process mining techniques are applied to logs extracted from software systems supporting the touristic domain in order to get insights regarding software usage by users and to suggest improvements. Conversely, in [46], process mining is used on event logs from a booking system to discover violations on the system's architectural guidelines.

Process mining can be seen as a suitable passive analysis approach to validate the behavior of processes in trading systems. The management of orders inside a trading platform, or the communication management between agents and the platform are examples of processes that can be analyzed with this approach. Albeit there are no research works that have studied before the use of process mining for validating trading systems, its potential has been acknowledged by domain experts [21]. Figure 2 illustrates how process mining can be used to validate a process in a trading system. A process model (e.g., a Petri net) can be designed based on the specification of the process to diagnose. The model describes the expected behavior that the process under evaluation should comply. Conversely, an event log recording real behavior of the process can be obtained by pre-processing data sources of the trading system. The data can be extracted from network messages exchanged between agents and a trading platform. Network messages indicate the activities executed by agents or a platform, as well as they inform the status of buy and sell orders. Then, a conformance checking method can be used to search for differences between the process's real behavior recorded in the event log and the process model. As output, a list of deviations can be obtained, revealing concrete points where the real process did not comply with the specification. Deviations may reveal that some orders where not correctly handled by a platform, or that some agents did not interact with the platform according to the specification. Finally, this output can be leveraged by engineers to opportunely fix the process under evaluation.

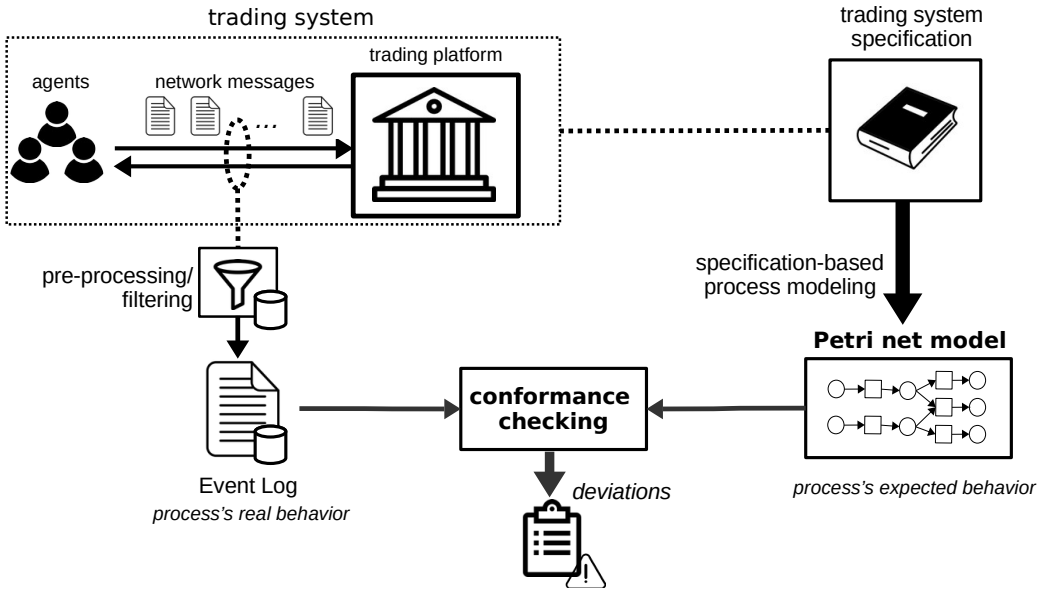

Figure 2: An approach to validate real processes in trading systems via process mining.

As illustrated in Figure 2, *Petri nets* are chosen as the formalism to model the expected behavior of processes in trading systems, whereas *conformance checking* is used to validate the real behavior of such processes, by comparing event logs with Petri nets. In what follows, we briefly introduce Petri nets and conformance checking, motivating our choice of using both for modeling and analysis of trading systems.

Petri Nets: Modeling *Expected* Behavior

Petri nets are one of the best investigated formalisms for modeling concurrent distributed systems [33]. The formalism was introduced by Carl Adam Petri in 1962, although their graphical representation was introduced later. Petri nets have proved to be useful in tasks such as system design, simulation and verification [17]. The majority of methods in process mining are grounded on the use of Petri nets, which can be seamlessly mapped to other process modeling notations.

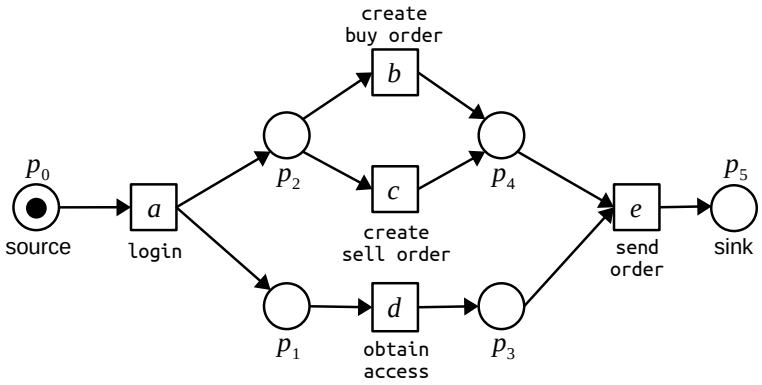

Figure 3: A Petri net describing the process of a "one-time trader": a trader that connects to a trading system to exchange only one order.

A Petri net is a bipartite graph that consists of two kinds of nodes: *places* and *transitions*. Places (drawn as circles) may represent buffers of resources, virtual or physical locations, logical conditions, etc. Transitions (drawn as boxes) may denote tasks, jobs, activities of a process, etc. Places may store tokens (drawn as black dots), where tokens may abstractly represent signals, control threads, resources, individual objects, etc. A distribution of tokens across the places of a Petri net is called a *marking*, abstractly representing a current state of a system. For instance, the Petri net in Figure 3 models a "one-time trader" process. This process corresponds to a class of participants that logins in a trading system, submits exactly one order (either a buy order or a sell order), and finally he logouts from the system once the order is placed. In particular, the model in Figure 3 is a *workflow net* (WF-net) [1], a class of Petri nets widely used in process mining to model processes. A WF-net consists of two distinguished places, a *source* and a *sink*, such that all nodes are in a path from the source to the sink. A token in the source place models the initial state of a process, whereas a marking where only the sink place has a token corresponds to the process's final state.

Similar to other modeling notations used in process mining, Petri nets allow to model the control-flow perspective of processes, that is, activities of a process and their causal dependence. In Figure 3, the Petri net specifies an expected ordering between activities. This ordering must be complied in every run of the real process: the activity `login` must be executed first; then, the trader choices to create a `new buy order` or a `new sell order`, while in parallel he may `obtain access` to the trading system; finally, after obtaining access to the system, the participant sends his order for trading.

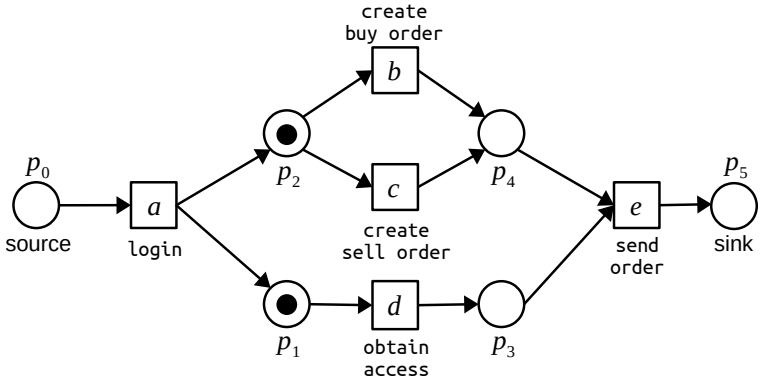

Figure 4: The Petri net of Figure 3 in a new marking after the firing of transition a (activity `login`)

Petri nets are "executable": the *firing* of a transition t consumes a token from each input place of t, and produces a token in each output place of t; as a result, the marking of the Petri net is updated. The firing of a transition abstractly represents the execution of an activity in a system, which consequently changes the current system's state. In Figure 3, the firing of transition a (activity `login`) consumes a token from the source (place p_0) and produces a token in place p_1 and another token in place p_2. The new marking resulting after the firing of transition a is depicted in Figure 4. In this new marking , transition d (activity `obtain access`) may fire as there is a token to be consumed from input place p_1. Similarly, either transition b or c may fire as a token is ready to be consumed from place p_2. The marking depicted in Figure 4 relates to the state of a trader that, after sending a login request, he can create an order, while in parallel he waits to obtain an access approval from the trading platform.

Petri net extensions can be used when it becomes crucial to model and validate other aspects beyond the process's control-flow. For instance, when analyzing the management of orders inside a trading system, it may be required to check the correct transformation of orders' data attributes such as their price, quantity of stocks, etc. Conversely, when analyzing the communication of traders with a platform, a model should be able to properly represent the concurrent interaction of multiple agents. Extensions of Petri nets provide means to properly model specific aspects of these processes, by enriching ordinary Petri nets with concepts of data types, hierarchy and modules. Petri nets with identifiers [20], Colored Petri nets [24], Nested Petri nets [28], and DB-nets [32] are some examples of Petri net extensions. Thus, one may choose a certain extension of Petri nets depending on the application-specific aspects that must be analyzed in a process.

Conformance Checking: Validating *Real* Behavior

We consider conformance checking to compare real behavior of processes in trading systems against their specification. Conformance checking is a family of process mining methods that search for differences between real behavior of processes, as observed in *event logs*, and expected behavior described in models [7]. Event logs consists of cases, where each case is associated to a particular run of the process. Each case has an ordered sequence of events which is called a *trace*. Events indicate activities that were executed by the system. Then, to validate a specific case, conformance checking methods seek to relate its trace with a process model. There exist two well-known types of conformance checking methods: *replay* and *alignments*. Replay comprises the execution of a process model (e.g., a Petri net) according to the information available in events of a trace [44]. A deviation, that is, a precise difference between a real process and its model, is detected when the model cannot be executed as an event indicates. Conversely, alignments seek to directly match a trace with a valid execution sequence specified in the model (e.g., a sequence of transition firings in a Petri net) [5]. In alignments deviations are detected when a trace only partially matches with a valid execution sequence in the model.

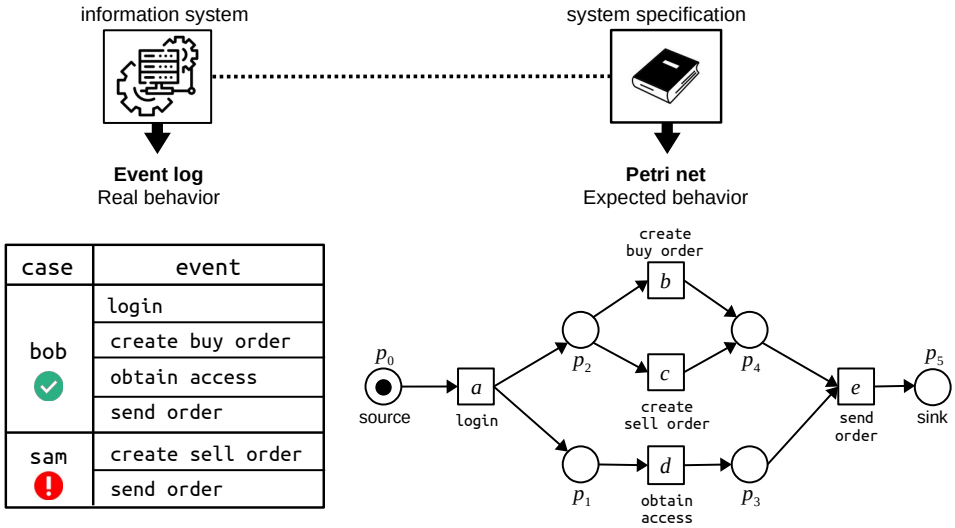

Figure 5: Example of replay-based conformance checking between a Petri net and an event log.

Figure 5 illustrates the use of replay-based conformance checking for comparing an event log and a Petri net. The event log records two cases related to individual runs of traders, `bob` and `sam`. The trace of each case is depicted. In this example, events only contain the activities executed by each trader. However, events in real-life event logs can be equipped with additional attributes. For each trace, the method executes the Petri net according to each event: a transition labeled with the event's activity is selected for firing. If the transition is not enabled for firing (i.e., there is not a token in each input place of the transition) then a deviation is detected, implying that the real process violated the causal ordering described by the model.

For instance, in Figure 5, for `bob`'s trace, transition a (labeled with activity `login`) is enabled, so it can fire, mimicking the execution of activity a in the real system by `bob`. It is easy to see that the Petri net can be executed according to each of the following events in `bob`'s trace, implying that the behavior of this trader complies with the model.

Conversely, if we replay `sam`'s trace, two deviations can be detected: the trader executed activity `create buy order` without executing before executing activity `login`. Also, he executed activity `send order` before the execution of activity (`obtain access`) which grants him approval to send the order. These are examples of deviations that can be systematically detected when executing the Petri net as indicated by every event in the trace.

Finally, note that the given example considers only to validate the process's control-flow as it makes use of ordinary Petri nets. In this sense, for evaluating other aspects such as data one needs to consider extensions of Petri nets.

Research Problem and Aim

The **research problem** that this thesis addresses is about how to properly model trading systems using Petri nets and how to use Petri net models on process mining, in particular conformance checking, for validating the real behavior of such systems. Petri nets and process mining have been employed in a plethora of application domains. Yet, their usage on trading systems has not been addressed before. In trading systems, there are *application-specific aspects* that modeling and validation techniques must be aware of, in such way to properly assess the correct behavior of these systems and for detecting errors. In particular, two important aspects must be considered:

- *Interaction of distinguishable objects such as orders and agents.* Trading systems are characterized by a high degree of interaction between distinguishable objects such as orders or traders. On the one hand, it is required to check whether multiple agents are interacting with a trading platform according to the communication protocol established in the system's specification. On the other hand, once traders submit buy orders and sell orders to a trading platform, it is needed to check whether orders are being handled according to the system's specification.

- *Transformation of data attributes of objects in trading systems.* In trading systems, it becomes necessary not only to validate the control-flow of processes, but also to check whether data attributes are transformed as intended. For example, for an order submitted in a platform, it is required not only to check a correct ordering between the activities related to such an order, but also if attributes of the order such as its stock quantity, its price, and other attributes are being transformed according to the system's specification.

The **research aim** of this thesis is to define proper models based on extensions of Petri nets that can describe application-specific aspects of processes in trading systems. Rather than seeking to develop a universal model that can be used to analyze all aspects of a process, we shall consider how different models, built with different Petri net extensions, can be used to evaluate individual aspects of a trading system. For example, a certain model may be used for analyzing the transformation of data attributes in orders, whereas another model can be used to describe the interaction of traders with a trading platform. Then, this work aims to develop process mining methods that can validate the behavior of real trading systems, including the detection of errors in these systems, by comparing their event logs with the proposed models of Petri net extensions.

Main Results of This Thesis

The **main research contributions** of this thesis are summarized as follows:

I. Models based on Petri net extensions have been defined for modeling application-specific aspects of trading systems. Petri net with identifiers and Colored Petri nets are used for modeling the interaction of buy/sell orders. In particular, the latter extension allows to describe the transformation of data attributes of orders. Also, Nested Petri nets are used for modeling individual behavior of traders and their interaction with a trading platform. Syntactical restrictions are defined for these models in such a way that the models can faithfully describe specific characteristics of these systems.

II. *New* conformance checking methods have been developed to validate trading systems. The methods seek for concrete deviations in a system by comparing the proposed models with event logs from the system. Different types of deviations can be detected depending on the system's aspect that is being validated: *control-flow* deviations, *priority rule* violations, *data* corruptions, and non-proper termination are examples of deviations that can be detected.

III. Prototypical implementations of the conformance methods were developed, and their use for validating a real trading platform is reported in this thesis. Experimental evaluations with artificial data are also reported.

Research Publications and Conference Presentations

Research Publications

The results of this thesis have been prepared in the following scientific articles, which have been published in international conference proceedings and peer-reviewed journals.

According to the rules of the HSE Dissertation Council in Computer Science*, articles are classified in two types. *First-tier* publications include papers indexed in the Web of Science or Scopus databases

*https://www.hse.ru/en/science/disscoun/council_computerscience/

(quartile Q1), as well as in peer-reviewed collections of conferences that appear in CORE rankings (ranks A and A*). *Other publications* are papers published in journals included in HSE's list of recommended journals[†], or indexed in the Web of Science or Scopus databases (quartile $\geq$ Q2 or without quartile), as well as in peer-reviewed collections of conferences appearing in CORE rankings (rank B). For each article, we provide its status and the proportion of the author's personal contribution measured in editor's sheets[‡]

Other Publications

- Carrasquel, J.C., Mecheraoui, K., Lomazova, I.A. *Checking Conformance Between Colored Petri Nets and Event Logs. Analysis of Images, Social Networks and Text* (*AIST* 2020), Lecture Notes in Computer Science, vol. 12602, Springer, 2021, pp.435-452 [13]. (*Scopus*, *Q2*; *author contribution: 0.84*).
- Carrasquel, J.C., Chuburov, S.A., Lomazova. *Pre-processing Network Messages of Trading Systems into Event Logs for Process Mining. Tools and Methods of Program Analysis* (*TMPA* 2019), Communications in Computer and Information Science, vol. 1288, Springer, 2021, pp.88-100 [8]. (*Scopus*, *Q4*; *author contribution: 0.67*).
- Carrasquel, J.C., Mecheraoui, K. *Object-Centric Replay-Based Conformance Checking: Unveiling Desire Lines and Local Deviations. Modeling and Analysis of Information Systems*, vol. 28, 2021, pp.146-168 [12]. (*HSE's list of recommended journals*; *author contribution: 1.18*).
- Mecheraoui, K., Carrasquel, J.C., Lomazova, I.A. *Compositional Conformance Checking of Nested Petri Nets and Event Logs of Multi-Agent Systems. Modeling and Analysis of Complex Systems and Processes* (*MACSPro* 2020), CEUR, vol. 2795, 2020 [30]. (*Scopus*; *author contribution: 0.3*).
- Carrasquel, J.C., Lomazova, I.A, Rivkin A., *Modeling Trading Systems using Petri Net Extensions. International Workshop on Petri Nets and Software Engineering* (*PNSE 2020*), CEUR, vol. 2651, 2020 [11]. (*Scopus*; *author contribution: 0.75*).
- Carrasquel, J.C., Lomazova, I.A. *Modeling and Validation of Trading and Multi-Agent Systems: An Approach Based on Process Mining and Petri Nets. Proceedings of the ICPM Doctoral Consortium*, CEUR, vol. 2432, 2019 [9]. (*Scopus*; *author contribution: 0.56*).
- Carrasquel, J.C., Lomazova, I.A., Itkin, I.L. *Towards a Formal Modeling of Order-driven Trading Systems using Petri Nets: A Multi-Agent Approach. Modeling and Analysis of Complex Systems and Processes* (*MACSPro* 2019), CEUR, vol. 2478, 2019 [10]. (*Scopus*; *author contribution: 0.67*).

[†] `https://scientometrics.hse.ru/goodjournals`

[‡] 1 editor's sheet is equal to 16 pages (or 40000 characters).

Conference Presentations

The results of this thesis have been presented and discussed at the following conferences and workshops:

- 6th International Conference on Software Testing, Machine Learning and Complex Process Analysis (*TMPA* 2021). *Online*. 25-27 November 2021. Talk: Searching for Deviations in Trading Systems: Combining Control-Flow and Data Perspectives.
- 12th Workshop on Program Semantics, Specification and Verification (*PSSV* 2021). *Online*. 4-5 November 2021. Talk: Validating Real Behavior of Agents in Trading Systems using Nested Petri Nets.
- 2nd Conference on Modeling and Analysis of Complex Systems and Processes (*MACSPro* 2020). *Online*. 20-24 October 2020. Talk: Compositional Conformance Checking of Nested Petri Nets and Event Logs of Multi-Agent Systems.
- 9th International Conference on Analysis of Images, Social Networks and Texts (*AIST* 2020). *Online*. 15-16 October 2020. Talk: Checking Conformance Between Colored Petri Nets and Event Logs.
- International Workshop on Petri Nets and Software Engineering (*PNSE* 2020) co-located with the 41stInternational Conference on Application and Theory of Petri Nets and Concurrency (*Petri Nets* 2020). *Online*. 24 June 2020. Talk: Modeling Trading Systems using Petri Net Extensions.
- 5th International Conference on Software Testing, Machine Learning and Complex Process Analysis (*TMPA* 2019). *Tbilisi, Georgia*. 7-9 November 2019. Talk: Pre-processing Network Messages of Trading Systems into Event Logs for Process Mining.
- Doctoral Consortium at the 1st International Conference on Process Mining (*ICPM* 2019). *Aachen, Germany*. 23 June 2019. Talk: Modelling and Validation of Trading and Multi-Agent Systems: An Approach Based on Process Mining and Petri Nets.
- 1st Conference on Modeling and Analysis of Complex Systems and Processes (*MACSPro* 2019). *Vienna, Austria*. 21-23 March 2019. Talk: Towards a Formal Modelling of Order-driven Trading Systems using Petri Nets: A Multi-Agent Approach.

Thesis Outline

The thesis consists of *eight* chapters. **Chapter 1** corresponds to the **Introduction**. A brief introduction on trading systems and process mining is presented. Petri nets and conformance checking are introduced as the techniques that have been chosen in the thesis for modeling and validation of processes in trading systems. The chapter introduced the research problem, the main research contributions and finally the structure of the thesis.

Chapter 2 describes the core elements of **trading systems**, on which we focus on this thesis: orders, order books and the Financial Information Exchange (FIX) protocol.

Orders are records submitted by agents into a trading platform in order to buy or sell securities; orders are equipped with different attributes that are configured according to the intentions of traders, e.g., how many stocks of a security to buy or sell, at which price per stock, etc.

Order Books are two-sided lists inside a trading platform where orders trading the same security are placed and trade according to certain priority rules [19]. Each order book has a buy side and a sell side, where buy orders and sell orders are respectively placed. Buy orders with highest prices and sell orders with lowest prices have highest priority in their sides and trade first. If two orders have the same price, then the order submitted earlier trades first. Table 1 shows how orders are prioritized in an order book. In the example, Bif's buy order and Sol's sell order have highest priority in their sides. A trade is executed iff the price of the buy order with highest priority is greater or equal than the price of the sell order with highest priority; the latter means that the best buyer is willing to pay at least as much as what the seller demands. In the current state of the order book in Table 1, a trade of 1 stock is executed between Sol's sell order and with Bif's order that buys 4. Sol's order is *filled* (traded all stocks), and Bif's order is *partially filled* (the order traded only some stocks) with a remainder of 3.

Table 1: Buy orders (a) and sell orders (b) are submitted into a trading platform; the platform places and ranks these orders into the corresponding order book according to a *price-time policy* (c).

(a) submitted buy orders

time	trader	size	price
10:01	Bea	3	20
10:08	Ben	2	20
10:15	Bif	4	market
10:18	Bob	2	20.1
10:29	Bud	7	19.8

(b) submitted sell orders

time	trader	size	price
10:05	Sam	2	20.1
10:08	Sol	1	19.8
10:10	Stu	5	20.2
10:20	Sue	6	20.0

(c) order book

buyer	size	price	size	seller
Bif	4	market		
		20.1	5	Stu
Bob	2	20.1	2	Sam
Bea	3	20.0		
Ben	2	20.0		
		20.0	6	Sue
Bud	7	19.8	1	Sol

The *Financial Information Exchange* (FIX) protocol is a communication standard widely used in modern trading platforms for the exchange of messages between agents and a trading platform [15]. Using FIX channels, agents can submit orders to a platform, and they can receive periodical reports about the status of their submitted orders. Hence, by parsing FIX messages exchanged between agents and a platform is possible to extract the real behavior of their associated processes.

FIX messages exchanged by agents and the platform consists of ASCII-encoded `tag-value` pairs, where every pair is separated by the ASCII control character `0x01`. *Tags* are integers that are associated to the different attributes that can be sent in a FIX message. Figure 6 exemplifies a FIX message sent from an agent to a trading platform. The message corresponds to an agent who is submitting a `day limit order` configured to buy 40 stocks (tag 38) of the security `VTB24` (tag 55) at a price per stock of 100 (tag 44). Thus, it can be seen how FIX messages inform the activities that agents may execute in a trading system. The complete set of FIX messages and all possible tag-value pairs can be found in [16].

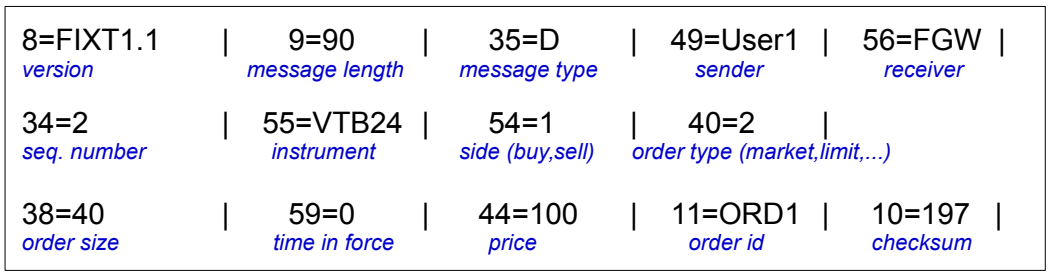

Figure 6: Example of a FIX message: *tag-value* pairs inform the activity that is executed in a system, as well as the data attributes related to the order. Tag 35 with value `D` indicates that the agent sending this message submits an order in the platform.

The next part of the thesis addresses the problem about how to properly model and validate processes in trading systems that manage the mentioned core elements of trading systems.
Two **processes** in trading systems are considered:

(1) *The management of* multiple *orders in order books inside a trading platform.* Multiple orders are submitted in different order books of a trading platform. Orders have different attributes that may change according to the execution of certain activities in a platform. The execution of these activities must comply a certain ordering. A trading platform must correctly implement priority rules in such a way to establish which orders should trade first, as well as it must execute as expected the trades between orders with highest priority.

(2) *The management of the connections between multiple agents and a trading platform.* Agents interacting with a trading platform must comply with the specification established by the FIX protocol.

Depending on the process to model and the specific aspects that must be analyzed, a certain Petri net extension model and a validation method can be chosen. Table 2 shows the models and methods that are proposed for modeling and validating each of these processes, including the chapters and sections where they are addressed.

Table 2: Models and methods presented in the thesis for analysis of processes in trading systems.

process	**aspects to model & validate**	**Petri net extension** (Chapter 4)	**Event log** (Chapter 5)	**Conformance checking methods** (Chapter 6)
(1) management of orders	interaction of *multiple* orders (control-flow)	Petri nets with identifiers (PNIs) (Section 4.1)	Event logs with object identifiers (Section 5.1)	Trace replay of an event log with object identifiers on a PNI model (Section 6.1)
	interaction of *multiple* orders (control-flow) and transformation of orders *data* attributes	Colored Petri nets (CPNs) (Section 4.2)	Event logs with object data attributes (Section 5.2)	Trace replay of an event log with object data attributes on a CPN model (Section 6.2)
(2) management of agents	interaction of *multiple* agents and their autonomous behavior	Nested Petri nets (NP-nets) (Section 4.3)	Event logs of multi-agent systems (Section 5.3)	Trace replay of an event log of a multi-agent system on a NP-net model (Section 6.3)

Chapter 3 introduces basic concepts about Petri nets and conformance checking which are the basis for the models and methods presented in this work.

Chapter 4 presents a study on **modeling trading systems using Petri net extensions**. These extensions come to enrich the base formalism of Petri nets with notions that resemble those of high-level programming languages such as data types, functions, modules and hierarchy. Depending on the process to model and the concrete aspects to evaluate, a certain Petri net extension may be chosen. In particular, the extensions of Petri nets considered on the thesis are: Petri nets with identifiers, colored Petri nets, and nested Petri nets.

Petri nets with identifiers (PNIs) are defined in Section 4.1 for modeling the management of orders in a trading platform. In this class of Petri nets, tokens carry distinct values, so buy orders and sell orders can be abstractly represented as tokens. Hence, this class particularly focuses on modeling the interaction of distinguishable objects, and omits other aspects such as the transformation of data attributes. Figure 7 exemplifies a PNI model describing the management of orders in a simple order book, where individual orders (represented as tokens) either trade once or they are canceled.

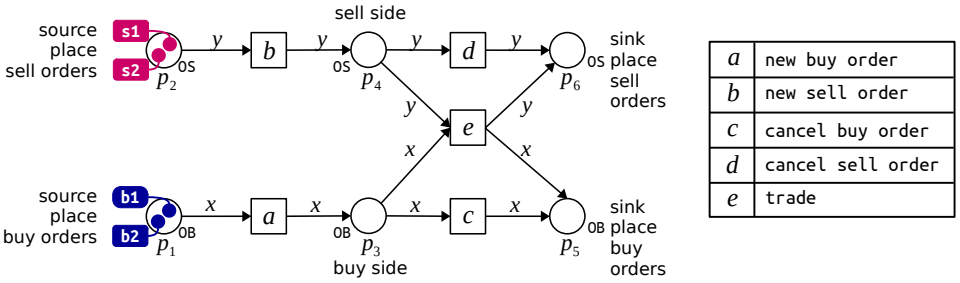

Figure 7: Petri net with identifiers (PNI) modeling the management of orders in a simple order book.

Colored Petri nets (CPNs) are presented in Section 4.2. CPNs are an extension of Petri nets where tokens carry values that belong to certain domains [24]. These values can be n-ary tuples. Tokens in CPNs can represent individual objects, which carry identifiers and also data attributes. In Figure 8, token (`b1`, `1`, `22.0`, `5`) represents an order with identifier `b1`, submitted in time `1`, to buy `5` stocks at price `22.0` per stock. Arcs in CPNs are labeled with expressions that may modify data attributes attached to tokens. Thus, CPNs allow us to describe the process of managing orders in order books of a platform, and how attributes of orders can be transformed upon the execution of certain activities.

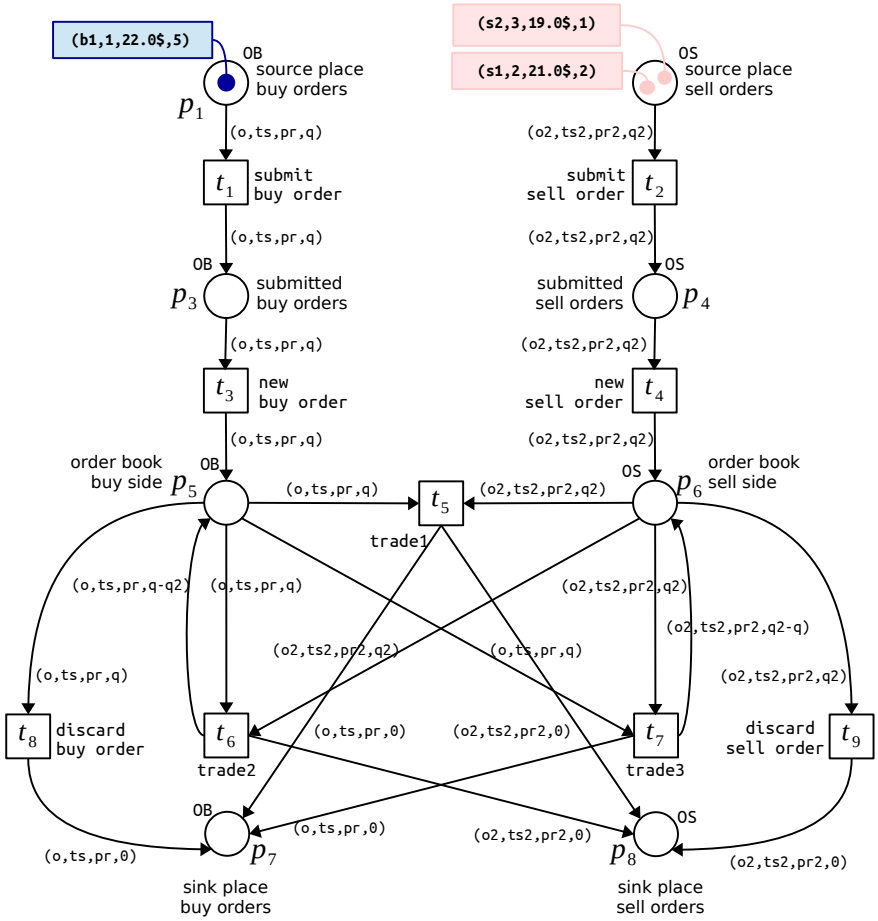

Figure 8: Colored Petri net modeling the management of orders in an order book.

Figure 8 shows a CPN modeling the management of orders in an order book. The CPN can be seen as a refinement of the *PNI* shown in Figure 7. The source places p_1 and p_2 are locations for incoming orders; places p_3 and p_4 are buffers for submitted orders; places p_5 and p_6 model the buy side and the sell side of the order book, whereas p_7 and p_8 are the sink places for filled or canceled orders. Transitions t_1 and t_2 model submission of orders by agents; transitions t_3 and t_4 model insertion of orders in the order book. Transitions t_5, t_6 and t_7 model variants of a trade: transition t_5 (activity `trade1`) models a trade where both orders are filled (all stocks were bought/sold); t_6 and t_7 (activities `trade2` and `trade3`) model the cases where only one order is *filled*, whereas the other is *partially filled* (returning with its stock remainder to a side of the order book). Finally, transitions t_8 and t_9 represent cancellation of orders.

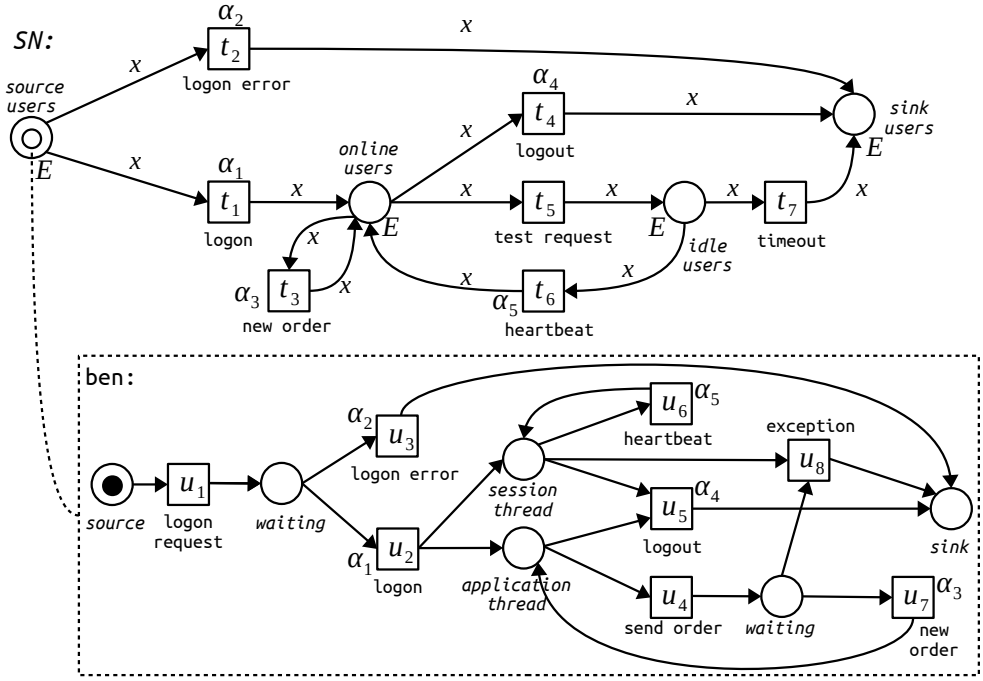

Figure 9: Nested Petri net describing agents autonomous behavior and their interaction in a platform.

Nested Petri nets (NP-nets) are presented in Section 4.3. NP-nets are an extension of Petri nets where tokens can be Petri net themselves (referred to as "net tokens") [28]. Every net token in a nested Petri net model will account for an individual agent whose autonomous behavior is defined by the inner Petri net of the token. Figure 9 depicts a NP-net model describing the expected behavior of multiple agents in a trading platform. In particular, the model focuses on the management of connections between agents and the platform, as well as the activities that each agent can execute depending on his connection state. The model abstractly describes the interaction between agents and a trading platform using the FIX protocol, as described in Chapter 2 of the thesis.

The *system net* SN of the NP-net in Figure 9 represents the trading platform. Places in the SN represent agents' connection states (also referred to as "locations"). Net tokens (drawn as white dots) stand for individual agents, which can be located in any of these places. Transitions in the SN represent activities that a platform executes to handle connection states of agents. For example, transition t_1 (activity

`logon`) transfers agents from the source place p_1 to place p_2 meaning that the platform approved the logon request of an agent. Conversely, each net token is associated with a different Petri net which allows to describe his autonomous behavior. For an inner Petri net of a net token, places represent local states of an agent, whereas transitions abstractly denote the activities that the agent may execute.

Synchronization labels labeled to some transitions allow the simultaneous firing of transitions on a system net and in net tokens. For trading systems, synchronization labels provide a way for modeling interaction and responses between the trading platform and agents. Execution steps in NP-nets are as follows: (*i*) An *element-autonomous step* which corresponds to the firing of a transition t without synchronization label in a net token, according to the usual rules for Petri nets; (*ii*) a *system-autonomous step* is the firing of a transition t in the system net according to the firing rule described above; and (*iii*) a *synchronization step* which is the simultaneous firing of a transition t in the system net with a synchronization label α, and the firing of enabled transitions (also labeled with α) in net tokens residing in input places of t.

For each of the presented classes of Petri net extensions, syntactical restrictions are formally defined in the thesis. These restrictions guarantee that the models that are constructed with these classes will comply with elemental characteristics of trading systems. For example, objects such as orders and agents should be unique, so all tokens should be distinguished. On the other hand, models should have the structure of workflows with distinguished sources and sinks. We defined the models that comply with these restrictions as conservative-workflow nets.

Chapter 5 presents definitions of different classes of **event logs** for describing real behavior of processes in trading systems. The following classes of event logs are defined: (1) *event logs with object identifiers* which record the interaction of individual objects (e.g., orders); (2) *event logs with object data attributes* which record how data attributes of different objects (e.g., orders) are transformed when activities are executed; and (3) *event logs of multi-agent systems* which record activities executed by a given trading platform and by multiple agents that concurrently interact with such a platform.

In what follows, the definitions of events logs stated in Chapter 5 are presented, as well as their corresponding examples in Tables 3-5.

Definition (**Event Log with Object Identifiers**). *Let $\mathfrak{D}$ be a finite set of types, and let $\mathcal{A}$ be a finite set of activities. An event log with object identifiers is a multiset of traces L. For every trace σ in L, each event in σ is a tuple of the form $e = (a, R(e))$, where $a \in \mathcal{A}$ is an activity, and $R(e)$ is a finite set of elements. For each $r \in R(e)$, we say that r_j is the identifier of an object of type D, involved in the execution of activity a, such that $r \in D \ \wedge \ D \in \mathfrak{D}$.*

Table 3: Event log with object identifiers with two traces related to different runs in a trading platform.

trace	**event** (e)	**activity** (a)	**objects** ($R(e)$)
σ_1	e_1	`new buy order`	`b1`
	e_2	`new sell order`	`s1`
	e_3	`new sell order`	`s2`
	e_4	`trade`	`b1 , s1`
	e_5	`cancel sell order`	`s2`
σ_2	e_1	`new buy order`	`b1`
	e_2	`trade`	`b1 , s1`
	e_3	`trade`	`b2 , s1`
	e_4	`new sell order`	`s2`

Definition (**Event Log with Object Data Attributes**). *Let $\mathfrak{D}$ be a finite set of types, let Σ be a set of colors defined over $\mathfrak{D}$, and let $\mathcal{A}$ be a finite set of activities. An event log with object data attributes is a multiset of traces L where: for every trace σ in L, each event in σ is a tuple of the form $e = (a, R(e))$ s.t. $a \in A$ and $R(e)$ is a set where $\forall r \in R(e)$, $r \in \mathcal{C}$ and $\mathcal{C} \in \Sigma$.*
For each element r in the set $R(e)$ of an event e, r is a tuple representing an object involved in the execution of activity a in event e.

Table 4: Example of one trace σ in an event log L with object data attributes.

event (e)	**activity** (a)	**objects** ($R(e)$)
e_1	`submit buy order`	(`b1`, 1, `22.0`, `5`)
e_2	`new buy order`	(`b1`, 1, `22.0`, `5`)
e_3	`submit sell order`	(`s1`, 2, `21.0`, `2`)
e_4	`new sell order`	(`s1`, 2, `21.0`, `2`)
e_5	`new sell order`	(`s2`, 3, `19.0`, `1`)
e_6	`trade2`	(`b1`, 1, `22.0`, `4`), (`s1`, 2, `21.0`, `0`)

Definition (**Event Log of a Multi-Agent System**). *Let $\mathcal{O}$ be a finite set of agent identifiers, and let $\mathcal{A}$ be a finite set of activities. An event log of a multi-agent system L is a multiset of traces where, for every trace σ in L, each event e in σ can be either:*

(1) a system-autonomous event, where $e = (a, \{r_1, ..., r_n\})$ such that $\{r_1, ..., r_n\} \subseteq \mathcal{O}$ are identifiers of agents involved in the execution of system activity $a \in \mathcal{A}$;

(2) an element-autonomous event, where $e = (a, r)$ such that $a \in \mathcal{A}$ and $r \in \mathcal{O}$, corresponding to the autonomous execution of activity a by agent r;

(3) a synchronous event, where $e = (a, \{(r_1, b_1), ..., (r_n, b_n)\})$, corresponding to the simultaneous execution of activity $a \in A$ in the system, and activities $b_1, ..., b_n$ by agents $r_1, ..., r_n$; for each $(r, b) \in \{(r_1, b_1), ..., (r_n, b_n)\}$ we have that $r \in \mathcal{O} \wedge b \in \mathcal{A}$.

Table 5: Example of an event log of a multi-agent system L with two traces, corresponding to runs in a trading system.

trace	event		trace	event	
σ_1	e_1	(`logon request`, `sam`)	σ_2	e_1	(`send order`, `ben`)
	e_2	(`logon request`, `bob`)		e_2	(`new order`, {(`new order`,`ben`)})
	e_3	(`logon`, {(`logon`, `sam`)})		e_3	(`test request`, {`ben`})
	e_4	(`logon`, {(`logon`, `bob`)})		e_4	(`heartbeat`, {(`heartbeat`, `ben`)})
	e_5	(`logout`, {(`logout`, `sam`)})			
	e_6	(`logout`, {(`logout`, `bob`)})			

Finally, *criteria of syntactical correctness* are presented in the chapter in order to properly map events and objects in each of these event logs with activities and markings in models of Petri net extensions.

Chapter 6 presents **conformance checking methods** for validating the real behavior of processes in trading systems. The methods are based on the replay of traces of the event logs defined in Chapter 5 on top of the models of Petri net extensions presented in Chapter 4. Each of the methods return a list of deviations, indicating the precise differences between a real system (as observed in traces) and a model. Depending on the process and aspect to validate a certain method can be chosen.

In Section 6.1, a method is presented for **validating the interaction of orders** in order books by replaying a trace of an event log with object identifiers on a Petri net with identifiers. Also, this method allow to compute *local conformance diagnostics*, which allow to identify deviations in concrete parts of a system.

In Section 6.2, a method is presented for **validating the transformation of data attributes** of orders by replaying a trace of an event log with object data attributes on a colored Petri net model. By leveraging semantics of CPNs, different data-related deviations can be detected such as violation of priority rules or resource corruption (i.e., attributes that were not correctly updated).

In Section 6.3, a method was presented for **validating the interaction of agents in a trading platform**. The method replays a trace of an event log of a multi-agent system on top of a nested Petri net model. Then, deviations can relate to errors that occur in a specific agent or in the trading platform.

Chapter 7 presents **experimental evaluations**. A software tool is presented, which implements the conformance checking methods proposed for validating trading systems. The tool was developed in Python programming language, and with the support of SNAKES [40] — a library that facilitates the construction and simulation of Petri net extensions. Using SNAKES, models of Petri net extensions presented in Section 5 are built as Python scripts. The tool is freely available at [18]. In addition, an independent routine to generate artificial event logs was also developed and it can be found in the project repository.

Experimental evaluations with artificial event logs are reported in Section 7.2. Particularly, we demonstrate how the replay method using Petri net with identifiers presented in Section 6.1 can be used for

detecting and visualizing deviations in precise parts of a process. Three event logs with object identifiers were artificially generated from different trading system models S_1 , S_2, and S_3. Each event log was replayed on the PNI of Figure 7. These models represent replicas of a trading system, according to the specification, but each of them with undesired behavior increasingly added. For instance, system S_2 is a variation of S_1, but with a subtle modification to slightly increase its difference with the specification model. Table 6 describes each replica and its event log generated. Each event log consists of 100 traces and 20 resources (i.e., 10 buys orders and 10 sell orders). The aim of this experiment is two-fold: to showcase the use of local conformance measures presented in Section 6.1., computed with all traces of an event log, and to study the stability of the proposed measures, e.g., how much the metrics are affected by subtle increases of undesired behavior.

Table 6: Experimental settings with artificial event logs

event log	*system source*	*number of events* (*total, avg. per trace*)	*system description (undesired behavior)*
L_1	S_1 (Fig. 10(a))	(2610, 26)	sell orders and buy orders skip activity b and a.
L_2	S_2 (Fig. 10(b))	(2726, 27)	S_1 + activity e may return sell orders to place p_4.
L_3	S_3 (Fig. 10(c))	(2575, 26)	S_2 + activity b may take sell orders to a deadlock place p_7.

Figure 10 presents the model of Figure 7 extended with *local conformance diagnostics* (defined in Section 6.1), that are computed when checking conformance in each variant, and which are associated to model components. Input arcs and dotted lines representing jumps indicate the rounded average number of transferred/jumped tokens that flowed through them, considering all traces of a log variant. The introduction of certain undesired behavior in each variant is unveiled by our method as a jump, showing how such undesired behavior induces more deviations in a precise component. The latter impacts on the conformance-related measures, used to paint the model to clearly identify where deviations occurred more.

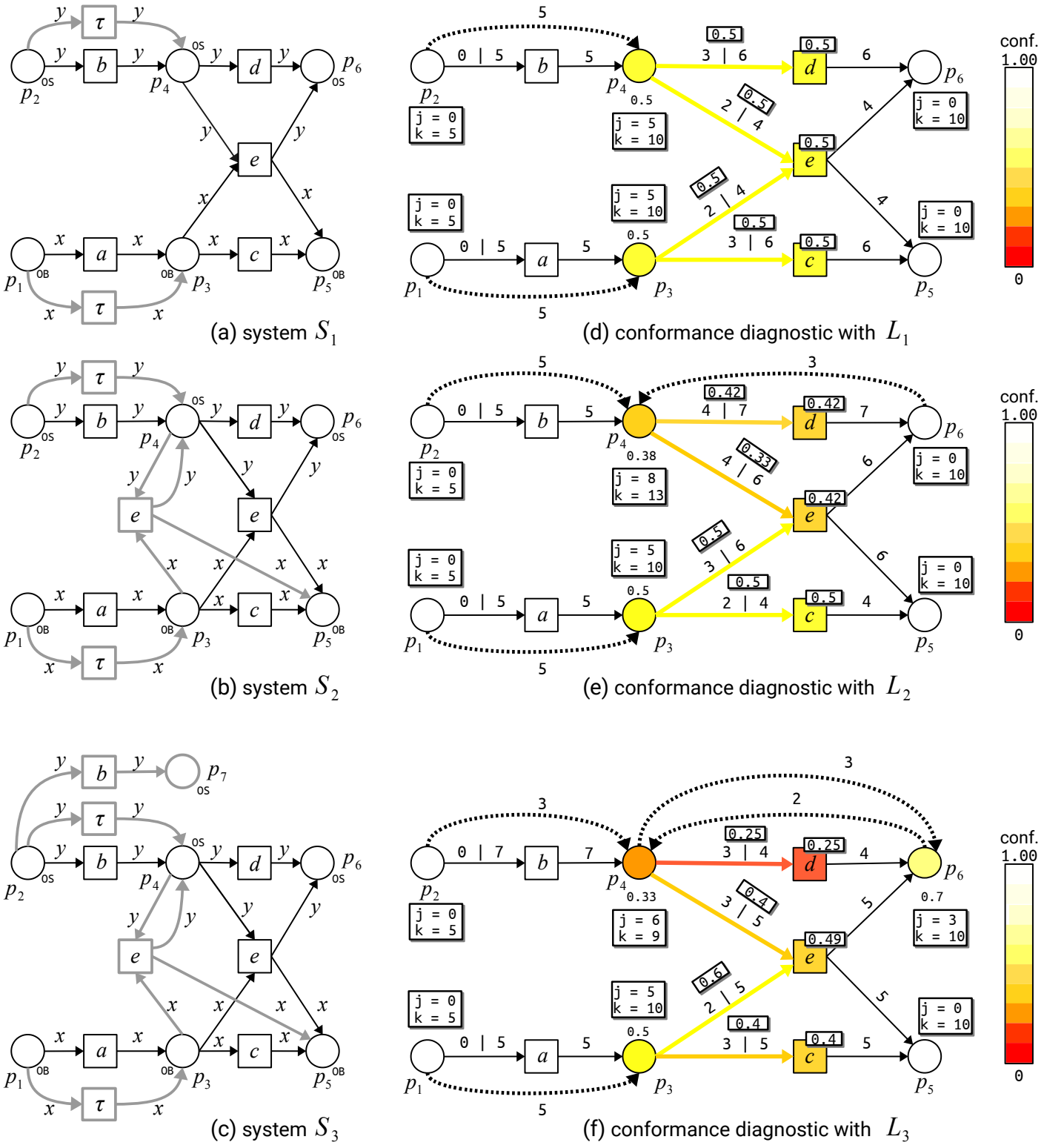

Figure 10: Unveiling deviations in concrete parts of a real system via conformance checking.

An experimental evaluation with a real event log is reported in Section 7.4. Using a real event log, we aimed at detecting deviations within a subset of order books in a real trading system. We considered order books with only *day limit orders*, orders that trade stocks at a fixed price, and that must trade or cancel by the end of a day. The orders considered are not amended after their submission. On the one hand, the system's expected behavior is described by the CPN of Figure 8. On the other hand, the event log consists of 73 traces (each trace is associated to a different order book) and 2259 events, with a mean of 30.94 events per trace. Figure 11 shows a fragment of this event log. The event log was obtained by pre-processing a set of FIX messages that were exchange between agents and a trading platform. The pre-processing methods is described in Section 7.3. Note that we only extracted traces that correspond to order books whose expected behavior is described by the CPN of Figure 8.

	A	B	C	D	E	F	G	H	I	J	K
1	CASE	TIMESTAMP	ACTIVITY	ID1	TSUB1	PRICE1	QTY1	ID2	TSUB2	PRICE2	QTY2
2	5169112	18-02-2019T03:00:23.804	submit buy order	00d0Phn2Svpv	18-02-2019T03:00:23.804000	105	100				
3	5169112	18-02-2019T03:00:23.805	new buy order	00d0Phn2Svpv	18-02-2019T03:00:23.804000	105	100				
4	5169112	18-02-2019T03:00:23.817	submit sell order	00d0Phn2Svpw	18-02-2019T03:00:23.817000	105	100				
5	5169112	18-02-2019T03:00:23.818	new sell order	00d0Phn2Svpw	18-02-2019T03:00:23.817000	105	100				
6	5169112	18-02-2019T03:01:24.280	trade1	00d0Phn2Svpw	18-02-2019T03:00:23.817000	105	0	00d0Phn2Svpv	18-02-2019T03:00:23.804000	105	0
7	5169113	18-02-2019T03:01:32.697	submit buy order	00d0Phn2SwNq	18-02-2019T03:01:32.697000	105	100				
8	5169113	18-02-2019T03:01:32.698	new buy order	00d0Phn2SwNq	18-02-2019T03:01:32.697000	105	100				
9	5169113	18-02-2019T03:01:32.708	submit sell order	00d0Phn2SwNr	18-02-2019T03:01:32.708000	105	100				
10	5169113	18-02-2019T03:01:32.709	new sell order	00d0Phn2SwNr	18-02-2019T03:01:32.708000	105	100				
11	5169113	18-02-2019T03:02:34.679	trade1	00d0Phn2SwNq	18-02-2019T03:01:32.697000	105	0	00d0Phn2SwNr	18-02-2019T03:01:32.708000	105	0
12	201701021	18-02-2019T05:05:44.541	submit buy order	00d0Phn2SzMt	18-02-2019T05:05:44.541000	12	10				
13	201701021	18-02-2019T05:05:44.544	new buy order	00d0Phn2SzMt	18-02-2019T05:05:44.541000	12	10				
14	201701021	18-02-2019T05:05:44.554	submit sell order	00d0Phn2SzMu	18-02-2019T05:05:44.554000	12	10				
15	201701021	18-02-2019T05:05:44.554	new sell order	00d0Phn2SzMu	18-02-2019T05:05:44.554000	12	10				
16	201701021	18-02-2019T05:05:44.555	trade1	00d0Phn2SzMu	18-02-2019T05:05:44.554000	12	0	00d0Phn2SzMt	18-02-2019T05:05:44.541000	12	0
17	201701021	18-02-2019T05:06:24.636	submit buy order	00d0Phn2SzPq	18-02-2019T05:06:24.636000	10	2000				
18	201701021	18-02-2019T05:06:24.642	new buy order	00d0Phn2SzPq	18-02-2019T05:06:24.636000	10	2000				
19	201701021	18-02-2019T05:06:24.650	submit buy order	00d0Phn2SzPr	18-02-2019T05:06:24.650000	10	2000				
20	201701021	18-02-2019T05:06:24.654	new buy order	00d0Phn2SzPr	18-02-2019T05:06:24.650000	10	2000				
21	201701021	18-02-2019T05:06:24.660	submit buy order	00d0Phn2SzPs	18-02-2019T05:06:24.660000	10	2000				
22	201701021	18-02-2019T05:06:24.663	new buy order	00d0Phn2SzPs	18-02-2019T05:06:24.660000	10	2000				

Figure 11: Fragment of an event log extracted from a real-life set of FIX messages.

A fragment of the deviations file computed by the method is shown in Figure 12. The file lists deviations detected in events of different traces of the input log. Each line describes precise information of a deviation in the real system: the trace (order book), event number, timestamp, and activity where the error occurred, the object affected, the kind of deviation detected, and an automatically generated description. In this experiment, most of the deviations relate to corruption of orders when executing trades: the prices of some orders changed upon the execution of trades, e.g., in event 1781 the price of the order with id. `bSovX` changed from 105 to 100 after trading, and such transformation is not described in the CPN. Thus, this information about deviations can be used by experts to confirm if this is a failure in the system, or instead the model should be slightly refined.

TRACE	EVENT	TIMESTAMP	ACTIVITY	OBJECT	DEV.	DEVIATION DESCRIPTION
1488058	1781	05:52:58.18	trade2	bSovX	RC	resource has event-state: ('b00d0PhqYSovX' 1550491266 100.0 100) ,but model-state is: ('b00d0PhqYSovX' 1550491266 105.0 100)
1488058	1782	05:52:58.18	trade1	bSovX	RC	resource has event-state: ('b00d0PhqYSovX' 1550491266 101.0 0) ,but model-state is: ('b00d0PhqYSovX' 1550491266 100.0 0)
1488061	1792	05:53:23.38	trade1	sSowK	RV	resource with id: s00d0PhqYSowK did not have priority ,over other resources in the same place.
1488061	1792	05:53:23.38	trade1	sSowK	RC	resource has event-state: ('s00d0PhqYSowK' 1550490938 101.0 0) ,but model-state is: ('s00d0PhqYSowK' 1550490938 101.0 -100)
1488061	1792	05:53:23.38	trade1	bSowJ	RC	resource has event-state: ('b00d0PhqYSowJ' 1550490919 101.0 0) ,but model-state is: ('b00d0PhqYSowJ' 1550490919 105.0 100)
1488061	1793	05:53:23.38	trade2	bSowJ	CF	resource with id: b00d0PhqYSowJ was not in location p5 but in p7
1488061	1793	05:53:23.38	trade2	bSowJ	RC	resource has event-state: ('b00d0PhqYSowJ' 1550490919 105.0 100) ,but model-state is: ('b00d0PhqYSowJ' 1550490919 101.0 -100)
1488061	1793	05:53:23.38	trade2	sSowL	RC	resource has event-state: ('s00d0PhqYSowL' 1550490947 105.0 0) ,but model-state is: ('s00d0PhqYSowL' 1550490947 100.0 0)
1488061	end	-	-	bSowJ	NT	resource with id: b00d0PhqYSowJ was not in final location p7 but in p5
1488062	1803	05:53:31.38	trade2	bSowN	RC	resource has event-state: ('b00d0PhqYSowN' 1550490899 100.0 100) ,but model-state is: ('b00d0PhqYSowN' 1550490899 105.0 100)
1488062	1804	05:53:31.38	trade1	bSowN	RC	resource has event-state: ('b00d0PhqYSowN' 1550490899 101.0 0) ,but model-state is: ('b00d0PhqYSowN' 1550490899 100.0 0)
9088012	end	-	-	bmkq9	NT	resource with id: b00d0PiS3mkq9 was not in final location p7 but in p5
9088012	end	-	-	smkqA	NT	resource with id: s00d0PiS3mkqA was not in final location p8 but in p6
9088015	end	-	-	sSSZd	NT	resource with id: s00d0Pi88SSZd was not in final location p8 but in p6

Figure 12: Fragment of deviations detected (DEV): resource corruptions (RC), priority rule violations (RV), control-flow deviations (CF), non-proper termination (NT)

.

Finally, the last chapter presents the **conclusions** of the thesis. The main contributions are summarized in the chapter. Also, it discusses directions for future research on the subject of this thesis.

Bibliography

[1] van der Aalst, W.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems, and Computers **8**(01), 21–66 (1998)

[2] van der Aalst, W.: Using Process Mining to Bridge the Gap between BI and BPM. Computer **44**(12), 77–80 (2011)

[3] van der Aalst, W.: Process Mining: Data Science in Action. Springer, 2nd edn. (2016)

[4] Abouloula, K., Krit, S.: Using a Robot Trader for Automatic Trading. In: International Conference on Engineering & MIS. pp. 1–9. ACM (2018)

[5] Adriansyah, A.: Aligning Observed and Modeled Behavior. Ph.D. thesis, Eindhoven University of Technology (2014)

[6] Averina, A., Itkin, I., Antonov, N.: Special Features of Testing Tools Applicable for Use in Trading Systems Production. In: Tools and Methods of Program Analysis (TMPA 2013). pp. 39–43. IEEE (2013)

[7] Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer (2018)

[8] Carrasquel, J.C., Chuburov, S., Lomazova, I.A.: Pre-processing network messages of trading systems into event logs for process mining. In: Tools and Methods of Program Analysis. CCIS, vol. 1288, pp. 88–100. Springer (2021)

[9] Carrasquel, J.C., Lomazova, I.A.: Modelling and Validation of Trading and Multi-Agent Systems: An Approach Based on Process Mining and Petri Nets. In: van Dongen, B., Claes, J. (eds.) Proc. of the ICPM Doctoral Consortium. CEUR, vol. 2432 (2019)

[10] Carrasquel, J.C., Lomazova, I.A., Itkin, I.L.: Towards a Formal Modelling of Order-driven Trading Systems using Petri Nets: A Multi-Agent Approach. In: Lomazova, I.A., Kalenkova, A., Yavorsky, R. (eds.) Modeling and Analysis of Complex Systems and Processes (MACSPro). CEUR, vol. 2478 (2019)

[11] Carrasquel, J.C., Lomazova, I.A., Rivkin, A.: Modeling Trading Systems using Petri Net Extensions. In: Köhler-Bussmeier, M., Kindler, E., Rölke, H. (eds.) Int. Workshop on Petri Nets and Software Engineering (PNSE). CEUR, vol. 2651 (2020)

[12] Carrasquel, J.C., Mecheraoui, K.: Object-centric replay-based conformance checking: Unveiling desire lines and local deviations. Modeling and Analysis of Information Systems **28**(2), 146–168 (2021)

[13] Carrasquel, J.C., Mecheraoui, K., Lomazova, I.A.: Checking conformance between colored petri nets and event logs. In: Analysis of Images, Social Networks and Texts. LNCS, vol. 12602, pp. 435–452. Springer (2021)

[14] Dumas, M., ter Hofstede, A.: UML Activity Diagrams as a Workflow Specification Language. In: UML 2001 — The Unified Modeling Language. Modeling Languages, Concepts, and Tools. LNCS, vol. 2185, pp. 76–90. Springer (2001)

[15] FIX Community - Standards: `https://www.fixtrading.org/standards/`

[16] FIX Dictionary Reference - All Message Types: Onix: `https://www.onixs.biz/fix-dictionary.html`

[17] Girault, C., Valk, R.: Petri Nets for Systems Engineering: a Guide to Modeling, Verification, and Applications. Springer (2013)

[18] Github: Conformance Checking using Petri net extensions - Project Repository. `https://github.com/jcarrasquel/hse-uamc-conformance-checking`

[19] Harris, L.: Trading and Exchanges: Market Microstructure for Practitioners. Oxford University Press (2003)

[20] Hee, K., Sidorova, N., Voorhoeve, M., van der Werf, J.: Generation of Database Transactions with Petri Nets. Fundamenta Informaticae **93**, 171–184 (2009)

[21] Itkin, I.: Mind the Gap Between Testing and Production: Applying Process Mining to Test The Resilence of Exchange Platforms [Accessed: January 24, 2022]. `https://tinyurl.com/y55sndcv` (2019)

[22] Itkin, I., Gromova, A., Sitnikov, A., Legchikov, D., Tsymbalov, E., Yavorskiy, R., Novikov, A., Rudakov, K.: User-Assisted Log Analysis for Quality Control of Distributed Fintech Applications. In: IEEE International Conference On Artificial Intelligence Testing (AITest). pp. 45–51. IEEE (2019)

[23] Itkin, I., Yavorskiy, R.: Overview of Applications of Passive Testing Techniques. In: Lomazova, I., Kalenkova, A., Yavorsky, R. (eds.) Modeling and Analysis of Complex Systems and Processes (MACSPro 2019). CEUR Workshop Proceedings, vol. 2478, pp. 104–115 (2019)

[24] Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer, 1st edn. (2009)

[25] Johannes, P.: From National Marketplaces to Global Providers of Financial Infrastructures: Exchanges, Infrastructures and Structural Power in Global Finance. New Political Economy **26**(4), 574–597 (2021)

[26] Kriger, A.M., Pochukalina, A., Isaev, V.: Reconciliation Testing Aspects of Trading Systems Software Failures. In: Spring/Summer Young Researchers Colloquium on Software Engineering. pp. 125–129 (2014)

[27] Leotta, F., Mecella, M., Mendling, J.: Applying Process Mining to Smart Spaces: Perspectives and Research Challenges. In: International Conference on Advanced Information Systems Engineering. LNBIP, vol. 215, pp. 298–304. Springer (2015)

[28] Lomazova, I.A.: Nested Petri Nets - a Formalism for Specification and Verification of Multi-Agent Distributed Systems. Fundamenta Informaticae **43**, 195–214 (2000)

[29] Mannhardt, F., Arnesen, P., Landmark, A.: Estimating The Impact of Incidents on Process Delay. In: International Conference on Process Mining (ICPM 2019). pp. 49–56. IEEE (2019)

[30] Mecheraoui, K., Carrasquel, J., Lomazova, I.: Compositional conformance checking of nested petri nets and event logs of multi-agent systems. In: Shapoval, A., Popov, V., Kakarov, I. (eds.) Conference on Modeling and Analysis of Complex Systems and Processes (MACSPro 2020). pp. 34–45. CEUR Workshop Proceedings, CEUR-WS.org (2020)

[31] Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. Ph.D. thesis, Vienna University of Economics and Business Administration (2011)

[32] Montali, M., Rivkin, A.: DB-Nets: On the Marriage of Colored Petri Nets and Relational Databases". In: Koutny, M., Kleijn, J., Penczek, W. (eds.) Transactions on Petri Nets and Other Models of Concurrency XII. LNCS, vol. 10470, pp. 91–118. Springer (2017)

[33] Murata, T.: Petri nets: Properties, Analysis and Applications. Proceedings of the IEEE **77**(4), 541–580 (1989)

[34] NASDAQ — Business Solutions for Public and Private Companies: `https://www.nasdaq.com/solutions`

[35] Nuti, G., Mirghaemi, M., Treleaven, P., Yingsaeree, C.: Algorithmic Trading. Computer **44**(11), 61–69 (2011)

[36] Parameswaran, S.: Fundamentals of Financial Instruments: An Introduction to Stocks, Bonds, Foreign Exchange, and Derivatives. John Wiley & Sons (2011)

[37] Patrick Eha, B.: Is Knight's $440 Million Glitch The Costliest Computer Bug Ever? [Accessed: January 24, 2022]. `https://money.cnn.com/2012/08/09/technology/knight-expensive-computer-bug/` (August 9, 2012)

[38] Pepitone, J.: Facebook: IPO Debacle Was NASDAQ's Fault [Accessed: January 24, 2022]. `https://money.cnn.com/2012/06/15/technology/facebook-ipo-lawsuit/` (June 15, 2012)

[39] Petram, L.: The World's First Stock Exchange: How the Amsterdam Market for Dutch East India Company Shares Became a Modern Securities Market, 1602-1700. Ph.D. thesis, University of Amsterdam (2011)

[40] Pommereau, F.: SNAKES: A flexible high-level petri nets library. In: Devillers, R., Valmari, A. (eds.) Application and Theory of Petri Nets and Concurrency. LNCS, vol. 9115, pp. 254–265. Springer (2015)

[41] Protsenko, P., Khristenok, A., Lukina, A., Alexeenko, A., Pavlyuk, T., Itkin, I.: Trading Day Logs Replay Limitations and Test Tools Applicability. In: Tools and Methods of Program Analysis (TMPA 2014). pp. 46–56 (2014)

[42] Ramadan, S., Baqapuri, H.I., Roecher, E., Mathiak, K.: Process Mining of Logged Gaming Behavior. In: International Conference on Process Mining (ICPM 2019). pp. 57–64. IEEE (2019)

[43] Reinkemeyer, L.: Process Mining in Action: Principles, Use Cases and Outlook. Springer (2020)

[44] Rozinat, A., Aalst, van der, W.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems **33**(1), 64–95 (2008)

[45] Rubin, V., Mitsyuk, A., Lomazova, I., van der Aalst, W.: Process Mining Can Be Applied to Software Too! In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM (2014)

[46] Shershakov, S., Rubin, V.: System Runs Analysis with Process Mining. Modeling and Analysis of Information Systems **22**, 818–833 (2015)

[47] Wohed, P., van der Aalst, W., Dumas, M., ter Hofstede, A., Russell, N.: On the Suitability of BPMN for Business Process Modelling. In: International Conference on Business Process Management (BPM 2006). LNCS, vol. 4102, pp. 161–176. Springer (2006)