Reconciliation Testing Aspects of Trading Systems Software Failures

Anna-Maria Kriger
Kostroma State Technological University
anna-maria.kriger@exactpro.com

Vladislav Isaev
Yuri Gagarin State Technical University of Saratov
vladislav.isayev@exactpro.com

Alyona Pochukalina
Obninsk Institute for Nuclear Power Engineering
alyona.pochukalina@exactpro.com

***Abstract* - This paper describes the concept of reconciliation testing - a process of using data reconciliation tools to validate the system in parallel with other activities. The authors studied information about two major software failures in electronic trading area: Facebook IPO on NASDAQ and Knight Capital runaway algorithms. This paper contributes to the subject matter by identifying aspects related to data reconciliation during these two events. The authors discuss the balance between automated and manual reactions to discrepancies reported by reconciliation tools and analyze the necessity of introducing reconciliation testing as part of system development life cycle for complex transactional processing systems.**

Keywords - data reconciliation, software testing, electronic trading

I. Introduction

Reconciliation is a process of finding discrepancies in data obtained from different sources. In accounting, reconciliation refers to the process of ensuring that two sets of records, usually account balances, match to each other. In financial markets, data reconciliation systems help asset managers to reconcile trades, cash and security flows, balances and positions between different systems, e.g. internal data stored by the trading participant vs. external data received from counterparties, brokers, clearers, custodians, etc. [1]. Data reconciliation packages are often used by middle- and back-office teams to identify breaks in post-trade data stored in relational databases. Most of data reconciliation research is also focused on various database related techniques [2]. General purpose extract, transform, load (ETL) products such as Informatica PowerCenter can be used as the basis for reconciliation tools implementations [3]. The financial services industry also uses specialized solutions such as UnaVista [4] from the London Stock Exchange. Data reconciliation can be implemented as:

a) End of day process

b) Periodic process

c) Real-time process

The optimal implementation approach depends on balance between time exposure risks of less frequent solutions and footprint requirements of more frequent solutions.

Slower solutions delay the delivery of critical information to the operational team, but require less hardware resources compared to faster solutions. Due to latency requirements, relational databases were removed from the main transactional path in most of the trading systems [5]. This fact, along with the desire to limit time exposure, is likely to be reflected in the next generation of reconciliation products that will move away from databases and focus more on real-time matching.

In the next part of the paper, Reconciliation Testing concept is described. Parts III and IV cover two samples of major software malfunctions in the electronic trading area. The first one describes events related to a broken Knight Capital algo that submitted millions of uncontrolled orders into the US markets and acquired a huge loss position. The second one describes problems with determining the uncrossing price and sending confirmations to members during Facebook IPO on NASDAQ exchange. The last part contains the analysis of similarities between Knight Capital and Facebook IPO events from data reconciliation and testing points of view.

II. Reconciliation Testing

Reconciliation testing is a process of using data reconciliation tools to validate the system in parallel with other testing activities. The term rarely appears in research papers. Data reconciliation tools can be viewed as passive testing tools due to their ability to check data consistency across the system without initiating any additional message flows [6]. The ability of data reconciliation tools to report errors in data consistency makes them useful test oracles for both functional and non-functional testing activities.

By their very nature, production data reconciliation tools satisfy the requirements for test tools that can be used in trading systems production environments [7]. Thus, data reconciliation tools can support the requirements of High Volume of Test Automation (HiVAT) methods:

- their impact on the system under test is acceptable for both production and test environments;
- tools can collect and process data regarding events in the system under test at production rates/volumes;
- they can highlight discrepancies in large data sets in a form that can be analyzed by the operational or QA teams;
- tools stability and resilience are sufficient to run high volumes of automated tests.

It is important to use data reconciliation tools during negative tests execution. The Quality Assurance team should

check whether negative scenarios can be picked by the data reconciliation tools or not. Whenever a negative test scenario leads to a discrepancy highlighted by the data reconciliation tool, the Quality Assurance team should validate whether it is possible to use the information from the tool to identify the source of the problem. This way, the operational team will have the necessary insight to take action if the problem ever occurs in production environment.

Reconciliation testing requires the presence of data reconciliation tools in the test environments. In some cases it can lead to an additional license costs and other expenses. Yet, the absence of production-like instrumentation limits the coverage of operational testing.

In order to perform reconciliation tests one needs to have data reconciliation tools available. The Quality Assurance team should strongly consider the possibility of implementing test tools capable of running passive data consistency checks. These tools should be implemented with a potential opportunity in mind that they will be also used in production environments.

In summary, the main aspects of data reconciliation tools are:

a) they are passive test tools

b) they serve as test oracles

c) they can be used with HiVAT methods

d) they should be used during negative test cycles

III. Knight Capital

Knight Capital was one of the most successful high-frequency trading (HFT) companies and represented approximately 10% of listed US equity securities in 2011-2012. Knight operated ultra-fast order router software named SMARS. A technical glitch in the system that happened on the 1st August 2012 led to an uncontrolled order submission into the market and accumulated a loss position of $460 million within 45 minutes period [8].

Smart Order Router (SOR) software is intended to execute orders in the current fragmented financial markets landscape [9]. Figure 1 below shows a simplified view of SOR system architecture.

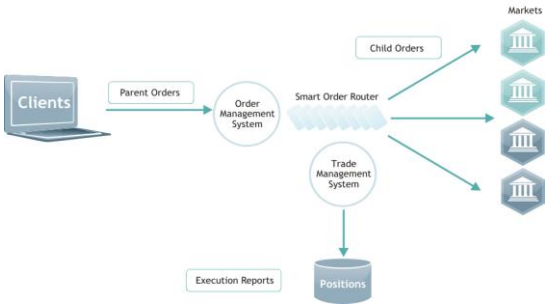

Fig. 1. Simplified SOR architecture

The Order Management System (OMS) receives orders from clients (parent orders) and after validation checks and controls passes them to the SOR subsystem. The latter creates market orders (child orders) for every parent order and sends them to different exchanges depending on the state of the markets and the internal business logic of the system. The information about trades is stored into Trade Management System (TMS) and trading positions and accounts are updated.

A set of reconciliation controls is necessary to protect the system as shown in Figure 2.

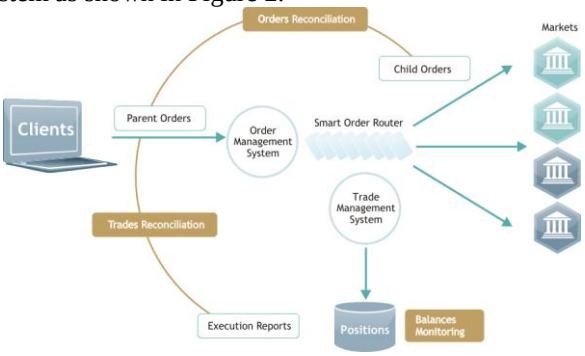

Fig. 2. Reconciliation controls in a SOR system

It is necessary to check the discrepancies between parent and child orders. When child orders are executed in the market it is necessary to reconcile market execution reports vs. parent orders. Whenever discrepancies are detected, they should be reflected in the error accounts.

The SMARS system contained the necessary reconciliation controls. However it appeared that they were not properly configured or tested. Reconciliation control to validate parent orders vs. child orders appeared to be higher in the source code and the SMARS system's logic was not affected by the check. All other risk controls were located at the OMS level and were not suitable to block problems should they happen in the SMARS system. Knight Capital had a special monitoring system called PMON to view positions accumulated in the error account, but its output was not linked to any kill-switch mechanism and did not provide sufficient information to operational teams to understand the source of the problem.

Knight Capital implemented changes related to New York Stock Exchange (NYSE) Retail Liquidity Platform in the SMARS system and put them live on the date of the events. Due to a human operator's error, the changes were deployed on seven servers instead of eight. The newly introduced switch triggered a piece of legacy code on that server, and the result was an uncontrolled flow of child orders into the market. The system continued to send child orders even though client parent orders were already filled. Broken real-time reconciliation controls were not able to halt erroneous run-away trading algorithm and post-trade controls were not designed to affect real-time flow. The Securities and Exchange Commission (SEC) executive order highlighted the lack of technical supervision in the firm and issued additional fine of $12 million. Knight Capital was not able to recover from these events, its share price dropped and the company was later acquired by one of its competitors.

IV. Facebook IPO

Facebook is the most widely used social network in the world. Its audience grew substantially over the years and exceeded one billion users. In 2012 company announced that

it selected NASDAQ market as its listing exchange. Facebook Initial Public Offering (IPO) was one of the largest IPOs in history. Many retail and institutional investors were going to participate and acquire company shares. On the day of IPO, 18th May 2012, the trading activities in the stock were disrupted by a set of technical malfunctions that lasted for several hours, had substantial financial impact on some of the market participants and led to SEC investigation [10].

The NASDAQ system is one of the most advanced trading platforms used by many national and alternative exchanges in many countries. The system has resilient and scalable distributed architecture and a set of built-in reconciliation controls targeted to validate internal data consistency. The trading system can operate in two different modes – continuous trading and auctions. Continuous trading is a very efficient way to organize markets with sufficient liquidity. Whenever a price of the buy order exceeds or equals the price of the sell order, a trade will happen during continuous trading. Market participants have immediate access to price discovery and trading opportunities. Continuous trading is a self-maintained process. However, is it not the most effective way of starting a new trading day or maintain an orderly market after significant material events. The reason for that is that every participant is afraid that others have information that is not reflected in the share price, as trading had not started yet and thus waits for others to submit their orders. Collectively, this behavior results in limited available liquidity. The problem can be resolved by the auction trading mode. For a designated period of time, the participants can submit, amend and cancel their orders, they can also view prices offered by other participants, but no trades will happen until a particular moment. Auction trading mode gives investor sufficient confidence to decide whether they really have an intention to trade at the price accepted by the market. At the end of the auction call period the exchange system identifies uncrossing price that will result in maximum volume of traded shares and the trading goes into the continuous mode [11]. Secondary trading in the NASDAQ markets usually starts with a special auction called "Display Only Period" (DOP). NASDAQ uses a separate component called "IPO Cross Application". It processes all orders to define the price at which the largest number of shares will trade and then the matching engine crosses eligible buy and sell orders at that price.

The NASDAQ system has a reconciliation control to validate that the list of orders presented in the matching engine is identical to the one used by the Cross Application to determine the price. This control directly affects the trading system and results in a request to recalculate the price whenever any discrepancy is located. One of the reasons for the reconciliation check to fail was that NASDAQ allowed participants to cancel orders even during a short period of uncrossing price calculation that usually takes 1-2 milliseconds.

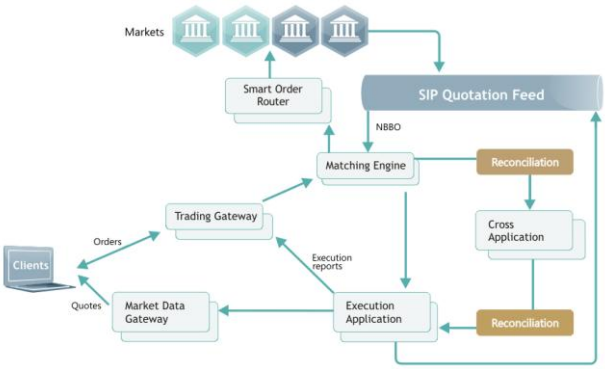

Fig. 3. Conceptual exchange trading system architecture

Information about the NASDAQ platform components and its architecture is not available in the public domain. We presented a generic view of the simplified architecture in Figure 3 based on information from the SEC report and our overall experience with similar systems. Clients submit orders into the trading gateways and orders are matched inside the matching engine. According to the report, the system uses separate components to calculate the uncross price during auctions and another application to send confirmation reports to members and publish quote updates called Execution Application. Similar to the Cross Application, the Execution Application also had associated reconciliation controls in place to make sure that its view of the orders match to the one available in the Cross Application.

On the day of Facebook IPO, the NASDAQ platform received unusually high number of orders from participants desiring to participate in market opening auction. The IPO Cross Application process took around 20ms to determine the uncrossing price and a single order was canceled during this period. The application repeated the calculation and reconciliation check, but more orders were canceled. The NASDAQ matching engine and the IPO Cross Application went into an infinite loop. Every attempt to recalculate the uncrossing price was followed by failed reconciliation check. Within the next 25 minutes, technical and executive teams determined that the reconciliation check prevented the system from opening the market and agreed on a so called Failover Proposal. Software update switching of the check was deployed on the secondary server and the primary one was killed, enabling the system to stop the cycle. Unknown at the time, due to the ongoing cycle the system's ability to process additional inbound order instructions was limited, and an extra 38k orders were stuck in the processing queue and did not participate in the cross. This led to the failure of reconciliation check in the Execution Application. Many market participants were not able to receive confirmations for their orders and trades until NASDAQ performed the second failover and switched off the reconciliation control in the Execution Application. Figure 4 shows the state of the system after both failover proposals were executed.

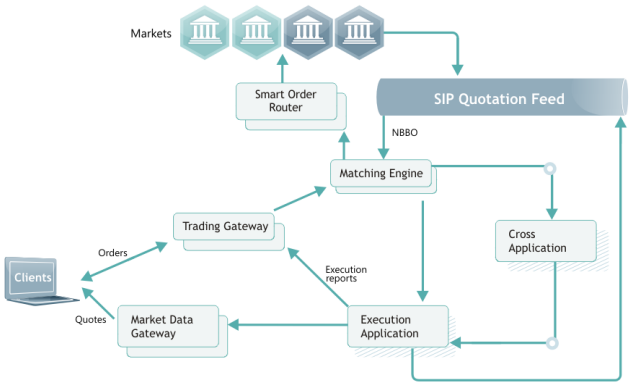

Fig. 4. NASDAQ system state after executing the second failover proposal

The events around Facebook IPO resulted in significant loss of investors' confidence, the NASDAQ operator was censured by SEC and had to pay an administrative penalty of $10 million and set aside a $62 million-worth fund to compensate firms harmed by the glitch.

V. Reconciliation Testing Analysis

Large-scale technology disasters are rarely a consequence of a single factor. Mostly, they result from a set of flaws in software development and maintenance processes. Data reconciliation controls serve as an independent additional protection mechanism for complex systems and therefore should be considered as a necessary part of production infrastructures. Reconciliation testing is an activity that not only helps to deliver systems that will behave correctly in production, but also provides additional confidence that operational teams will have sufficient information to take action if things unexpectedly go wrong.

In both the Knight Capital and the Facebook IPO cases, the trading systems had a reasonable set of reconciliation controls. In both cases, the impact of problems might have been significantly reduced had these controls worked properly. This section covers distinctions and similarities between the two considered events.

The correctness of real-time reconciliation control matching parent orders vs. child orders had not been tested by Knight Capital for several years. A negative scenario that resulted in a discrepancy between these two data sets could have highlighted that the risk control was longer active after being moved into another part of the source code. On the other hand, it is clear that reconciliation controls had been functionally tested by NASDAQ and proved to work as expected. However, the exchange team had never tested the course of actions if the reconciliation control failed permanently. The team executed the failover proposal without validating in detail first what impact it would have on other components and reconciliation utilities.

Both companies had a monitoring view that highlighted the problem to their operational teams. In both cases, the team was able to correctly interpret the extent of the events. The Knight Capital team erroneously decided to roll-back the changes and effectively made the things worse. The NASDAQ team was not aware of 38k orders stuck in the processing queue for some time, even though the reconciliation control in the Execution Application had picked up the problem immediately and marked the cross as invalid.

The Knight Capital reconciliation tools were not linked to any facilities to halt the trading. In the NASDAQ case, failed reconciliation immediately blocked further processing. Upon reflection, it is clear that neither of these two behaviors is ideal. It is necessary to have balance between automated stop-switches and the operators' ability to control reconciliation checks.

In both cases, real-time data reconciliation controls were built into the main transactional part. It might be a good idea to use tools separated from the main flow, e.g. surveillance sub-systems, to perform the data reconciliation function.

The following figure shows market surveillance system usages as the test tool.

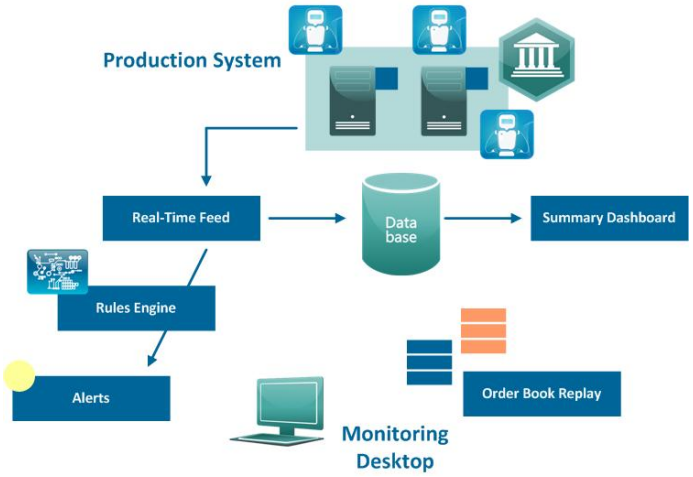

Fig. 5. Market Surveillance System used as reconciliation testing tool.

The primary task of a market surveillance system is to support the analytics gathered and analyzed by departments responsible for recognition of possible market abuse [12]. A surveillance system must collect the information pertaining to all incoming orders, system responses, data from external sources and relevant internal states of the trading platform.

It is possible and beneficial to use market surveillance system as a reconciliation testing tool for the following reasons:

- all required data is collected from the system and available both real-time and in the database;
- most of surveillance systems are configured as a downstream component and do not affect the main transactional path;
- rules engine allows creating data reconciliation checks and raise alerts when they fail;
- order book replay allows studying the exact source of the discrepancy.

VI. Conclusion

The examples of high-profile software failures presented in the paper show that incorrectly functioning data reconciliation controls in electronic trading systems can cause substantial financial losses. Validation of these controls needs to be incorporated into the software development life cycle for such systems.

A comprehensive test library should cover various potential discrepancies reported by data reconciliation tools. Operational teams should provide responses to each of these scenarios. Quality Assurance teams should verify that the tools provide enough information to identify the source of a discrepancy. The system itself should have controls to halt and resume trading both automatically and manually if a breakdown occurs in production environment.

Apart from being a critical part of production infrastructures, data reconciliation tools can provide additional test oracles for both functional and non-functional testing activities and enable more efficient testing of complex transactional processing systems.

The authors plan to proceed with researching data reconciliation tools applicability in software testing and developing a reference implementation of a scalable real-time tool for reconciliation testing based on the proprietary market surveillance platform.

References

[1] W. Wheatley Financial Systems, *Reconciliation Best Practice*, http://www.watsonwheatley.com/literature.html

[2] M, Cochinwala, V. Kurien, G. Lalk, D. Shasha, "Efficient data reconciliation", *The Journal of Information Science*, vol.137, issue 1-4, Sep. 2001

[3] R. Nolan, The Informatica Blog, *Even 'The Most Interesting Man In The World' Won't Do This…* http://blogs.informatica.com/perspectives/2012/03/06/even-the-most-interesting-man-in-the-world-wont-do-this/

[4] London Stock Exchange. *How UnaVista Works*: http://www.londonstockexchange.com/products-and-services/matching-reconciliation/how-unavista-works/index.html

[5] I. Itkin, *Highload trading systems and their testing*, Highload++ 2012

[6] A. Matveeva, N. Antonov, I. Itkin, *"The Specifics of Test Tools Used in Trading Systems Production Environments"*, Tools & Methods of Program Analysis 2013

[7] A. Alexeenko, P. Protsenko, A. Matveeva, I. Itkin, D. Sharov, *"Compatibility Testing of Protocol Connections of Exchange and Broker Systems Clients"*, Tools & Methods of Program Analysis 2013

[8] SEC Release No. 70694. *In the Matter of Knight Capital Americas LLC*

[9] Foresight: *The Future of Computer Trading in Financial Markets* (2012) Final Project Report

[10] SEC Release No. 69655. *In the Matter of THE NASDAQ STOCK MARKET, LLC*

[11] *NASDAQ Stock Market Rules* http://nasdaq.cchwallstreet.com/

[12] D. Diaz, M. Zaki, B. Theodoulidis, P. Sampaio, *A Systematic Framework for the Analysis and Development of Financial Market Monitoring Systems,* Annual SRII Global Conference 2011