Solution Pattern for Anomaly Detection in Financial Data Streams

Maciej Zakrzewicz[1(✉)], Marek Wojciechowski[1], and Paweł Gławiński[2]

[1] Poznan University of Technology, Poznan, Poland
{maciej.zakrzewicz, marek.wojciechowski}@cs.put.poznan.pl
[2] Softman SA, Piaseczno, Poland
pawel.glawinski@softman.pl

Abstract. Anomaly detection in versatile financial data streams is a vital business problem. Existing IT solutions for business anomaly detection usually rely on explicit Complex Event Processing or near-real time Business Activity Monitoring. In this paper we argue that business anomaly detection should be considered an implicit infrastructural BPM service and we propose a corresponding Solution Pattern. We describe how a Business Anomaly Detector can be architectured and designed in order to handle fast dynamic streams of business objects in BPM environments. The presented solution has been practically verified in Oracle SOA/BPM Suite environment which handled real-life financial controlling business processes.

Keywords: Service oriented systems · Design patterns · Risk management

1 Introduction

Business anomaly detection is a technique used to identify business items or events which do not conform to valid data patterns. Typical business anomalies include credit card frauds, purchase card frauds, telecommunication subscription fraud, phone call fraud, financial reporting fraud, insurance fraud, fraudulent claims for health care, credit applications fraud, credit transactional fraud, etc. Since fraud is a multi-million dollar business, efficient IT solutions are demanded by the business sector to timely detect anomalous/fraudulent activities attempted by performers of business processes fed with streams of complex data. Anomalies in business process execution may exhibit themselves in different manners, e.g., as an unusual process object state, as an unusual process execution path, as an unusual performer-to-activity assignment.

For a long time, SOA-based Business Process Management Systems (BPMS) have been used to design and execute services (typically SOAP Web Services) orchestrated into complex business processes performed by humans and applications. However, despite the maturity of business process modeling techniques and efficiency of execution platforms, effective monitoring of the flow of such business processes still lacks usability and flexibility, especially in the area of detecting anomalous business behavior, e.g., representing fraudulent actions. Existing solutions rely either on explicit

T. Welzer et al. (Eds.): ADBIS 2019, CCIS 1064, pp. 77–84, 2019.
https://doi.org/10.1007/978-3-030-30278-8_10

calls to rule evaluation systems (Complex Event Processing) in order to validate business data, engage external tools for near-real time business reporting (Business Activity Monitoring) or simply assume that anomaly detection is outside functional requirements. We argue that monitoring capabilities of BPMSs should be expanded with functions to automatically monitor every single business process instance to detect anomalous activities and report their findings to other components/processes.

Architectural, Design, and Solution Patterns are known as generalized, formalized descriptions of reusable solutions to common problem classes within a given context, supposed to transfer knowledge about successful designs and implementations. Examples of patterns that may partially support business anomaly detection include Complex Event Processing (CEP) Design Pattern and Business Activity Monitoring (BAM) Design Pattern. Unfortunately, the diverse nature of business processes and their business data objects make it challenging to develop a universal CEP or BAM framework for business anomaly detection.

In this paper we describe our proposal for the Business Anomaly Detection Solution Pattern and we present its successful implementation in the form of an asynchronous Java EE service which can be easily injected into existing Business Process Management (BPM) environments, allowing business processes to benefit from automated detection of anomalous behavior.

The remainder of this paper is organized as follows. Section 2 contains related work. In Sect. 3 our Business Anomaly Detector Solution Pattern is characterized. Section 4 describes architecture and design of a real-life Business Anomaly Detector. Section 5 summarizes results of selected experimental validation tests that we have conducted to justify our design decisions. Conclusions are presented in Sect. 6.

2 Related Work

General anomaly detection methods have been covered by numerous papers and surveys. Statistical outlier detection techniques have been described in [3]. Machine learning anomaly detection methods have been surveyed in [9]. A review of anomaly detection techniques for numerical and symbolic data has been provided in [2].

SOA best practices and technologies have been covered in [7]. Surveys on Web Service composition methods can be found in [8]. Implementation best practices and messaging patterns for SOA have been described in [5]. In [6] the authors discussed Business Activity Monitoring functional requirements and applications. Patterns for business processes have been extensively studied in [1]. In [10] the motivation for design patterns for Complex Event Processing has been presented and a foundation for them has been provided.

3 Problem Definition

In this section we will follow a usual three-parted scheme [4] to describe our Solution Pattern.

3.1 Context: SOA, BPM, Dynamic Data Streams

The background is a SOA-based Business Process Management environment, where complex business processes orchestrate both software services and user tasks. The business processes are fed with data entered by users as well as by data stream sources like sensors, POS terminals, automatic document feeder scanners, IoT devices, etc. Software services are invoked either directly or through an Enterprise Service Bus (ESB).

3.2 Problem: Anomaly Detection Based on Rules and Learning Models

Automatic identification of business data objects or events which do not conform to patterns. The patterns can be static in nature (defined by an operator) or dynamically discovered (by using machine learning methods). The business data objects/events to be analyzed can be any business process objects created, retrieved or transmitted by business process tasks. The business data objects/events will be delivered explicitly or implicitly intercepted in-flow during a business process execution. The solution should be platform-agnostic to integrate with various BPM environments.

3.3 Solution: Infrastructural Service

A new architectural component of BPMS, a form of an infrastructural service which performs on-line monitoring of business data objects being transmitted between activities of a business process. The captured business data objects are validated against the static and discovered patterns in order to detect anomalies. When an anomaly is detected, a BPM message or signal is generated to notify other processes or applications about the finding (Fig. 1).

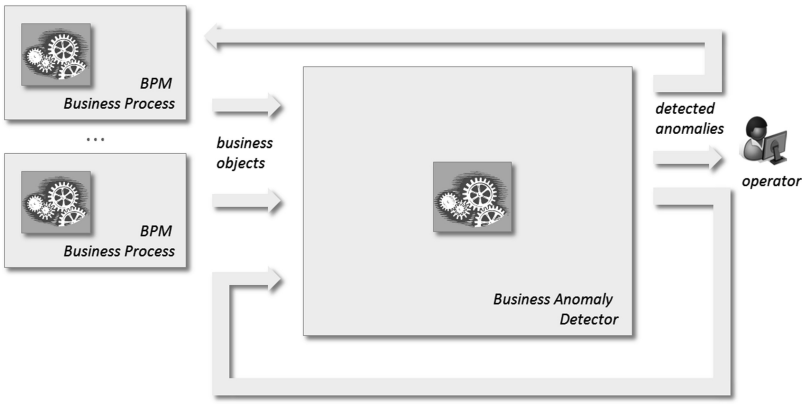

Fig. 1. Business Anomaly Detector as an infrastructural service

We have successfully developed a prototype of a Business Anomaly Detector based on the Java EE platform, using SOAP Web Service messaging, JMS/Kafka message buffering, Drools Rule Management System, Weka Machine Learning, and ESB

message interception. The prototype has been validated in an Oracle SOA/BPM Suite environment which handled real-life financial controlling business processes.

4 Software Architecture and Design

4.1 Overview

An overview of the architecture of the Business Anomaly Detector is shown in Fig. 2. Business objects are delivered to SOAP Web Service interfaces as XML documents. JAXB converts the XML documents into Java objects, which are sent to a throttling JMS queue or Kafka topic. The objects in the queue/topic are periodically propagated by the Controller to the embedded JBoss Drools rule engine, which then executes business anomaly detection rules on the received objects. The business anomaly detection rules are designed by a business user using a visual rule editor (part of our solution). The rules can be based on the expert's knowledge or rely on statistical models obtained through machine learning. The models are learnt using Weka algorithms integrated into the Business Anomaly Detector. When anomalies are detected, new business objects are created and delivered to an output JMS queue or Kafka topic. The Dispatcher splits the queued objects into classes and delivers them to external consumers (SOAP Web Services) based on defined allocation schemes. All management tasks are handled by a Web-based administration console. A database repository is used as a persistence store to protect the state of the Business Activity Detector in case of failures or planned unavailability. MongoDB has been selected for that purpose in our implementation due to its permissive license, simple but adequate data model, and small write overhead.

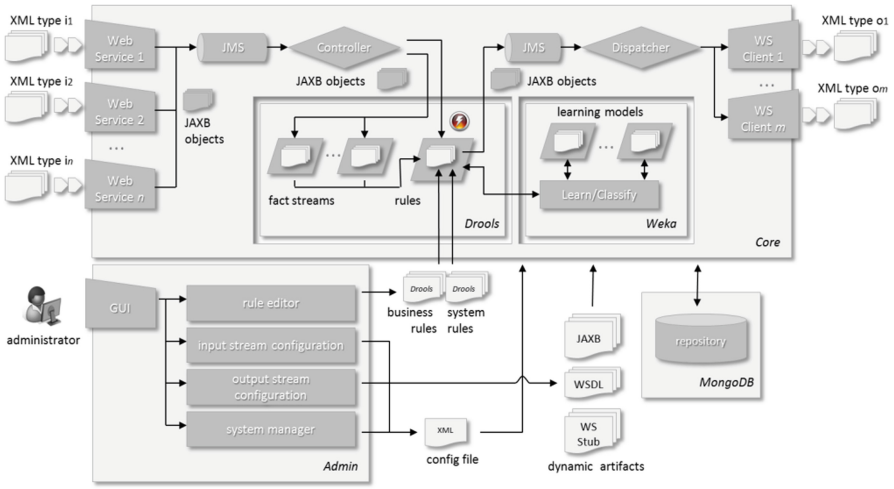

Fig. 2. Business Anomaly Detector software architecture

Business objects need to be delivered to the Business Anomaly Detector in order to be processed using static or dynamically discovered anomaly detection rules. Several scenarios can be considered for this action: (1) a business process can explicitly invoke the Business Anomaly Detector, providing objects to be analyzed, (2) a service call can be intercepted on ESB level and its object can be delivered to the Business Anomaly Detector, (3) an external application (e.g., a database table trigger, a network firewall) can invoke the Business Anomaly Detector and pass business object data.

Whenever an anomaly has been detected, a new business object is generated asynchronously. The object can be then consumed by an event-based business process to perform standard anomaly-related activities (e.g., notifications and alerting) or it can be processed by an existing Business Activity Monitoring tool in order to visualize the findings to the operator.

4.2 Input and Output Configuration

Business Anomaly Detector receives input data in the form of streams of XML documents. Each input stream corresponds to a particular class of analyzed business objects, e.g., invoices, bank transfers, credit card payments, etc. In order to define a new input stream a user has to upload an XML Schema document describing the structure of XML documents that will form the stream. Based on the uploaded XML Schema, the system will automatically complete the process of input stream definition by performing the following steps: (1) generation and compilation of JAXB classes, (2) generation of a SOAP Web Service with one operation, called "Receive", (3) deployment of the created Web Service. The automatically generated code of the Web Services' "Receive" operation forwards the JAXB object corresponding to the received XML document to the input JMS queue or Kafka topic.

The output of Business Anomaly Detector is a collection of streams of XML documents representing discovered anomalies, sent to registered external consumers. It is assumed that a consumer provides a SOAP Web Service to receive XML messages from Business Anomaly Detector. When defining an output stream, a user has to upload an XML Schema describing the outgoing XML documents and provide parameters identifying the consuming SOAP Web Service and its operation. As part of the output stream definition process, the system will automatically perform the following steps: (1) generation and compilation of JAXB classes, (2) adding a record to the system configuration associating the generated JAXB class with the external Web Service.

4.3 Rule-Based Anomaly Detection

The heart of the Business Anomaly Detector is its rule engine. As this crucial component we incorporated Drools since it was the only open source rule engine with a permissive license that supported all types of rules included in our functional requirements. Drools is implemented using Java, has a well-defined API conforming to the JSR-94 specification, and therefore seamlessly integrates with Java EE applications.

Business rules in Drools are expressed in a textual format called DRL and have a WHEN-THEN syntax. Coding business rules directly in DRL can certainly be

beneficial for advanced users due to its flexibility and expressive power. However, relying on free-text rule editing only could make the system too difficult to use for novice users, especially those without a technical background. Based on this observation, we developed a GUI to define rules of low or moderate complexity in the form of an interactive dialog window, still allowing advanced users to switch into the direct DRL editing mode.

Four types of rules are supported by the Business Anomaly Detector and handled by its Drools rule engine component:

1. Simple rules based on the current business object only. Example: When a credit card payment exceeds 1500 EUR, then raise an alert.
2. Aggregation rules based on moving window aggregates calculated from collections of business objects received in the past. Example: For each new bank transfer calculate the number of bank transfers within the last 24 h for the given customer and when this number is greater by at least 50% than the average daily number of bank transfers for this customer from the last 30 days, then raise an alert.
3. Calendar rules based on schedules to validate business objects received recently. Example: Every day at 23:59 calculate a balance for each customer for the time window of the last 30 days and when the balance is below -10000 EUR, then raise an alert.
4. Learning model rules based on patterns learnt from business objects received recently. Example: When the actual decision for a loan application is different than predicted by the statistical model, then raise an alert.

For the rules of type 2 we had to extend the out-of-the box Drools engine with aggregate materialization functionality to achieve satisfactory performance of the system on large amounts of data (see Sect. 5). For the rule of type 4 we integrated the Weka data mining library to build prediction models and implemented a solution for keeping the prediction models up to date (see Sect. 4.4).

4.4 Discovery-Based Anomaly Detection

Data mining subsystem within our Business Anomaly Detector serves two purposes: (1) generates statistical models based on event history from a defined time window, (2) based on the built statistical model, discovers anomalies in incoming new business events. The system reacts to discovered anomalies (e.g., by raising an alert) according to defined business rules.

Despite the fact that the input to our system can be regarded as a data stream, we decided against using a dedicated data stream mining library due to immaturity of available solutions. Instead, we incorporated Weka, which is a well-established and flexible library. From the collection of algorithms offered by Weka for anomaly prediction we selected two classification algorithms: J48 decision trees and Naïve Bayes. To keep the anomaly detection models up to date we periodically rerun the prediction algorithms on recent data according to a user-defined time window. We also enable operators to define derived or calculated facts/attributes in order to provide for anomaly detection in correlated parallel data streams.

As building prediction model takes significant time, our system performs this task in the background. Each statistical model is learnt in one separated execution thread. When a new prediction model is ready, it is put into production, and immediately afterwards the new model is being learnt from the most recent time window including the data that came during the previous model learning cycle. Thus, each statistical model exists in two versions: the completely learnt Foreground Model which is used for scoring incoming data and the Background Model being learnt that will become the new Foreground Model when completed. Expired statistical models (ones replaced by new Foreground Models) can be archived (kept in the MongoDB repository) for the sake of future analytics or auditing.

The detailed procedure of machine learning and prediction within Business Anomaly Detector consists of the following steps: (1) A new business event object is retrieved from the input JMS queue or Kafka topic. (2) The business event object is placed in two Drools streams. The first stream is temporary, i.e., the object is removed from it after being processed by all defined business rules. The second stream maintains an event time window on which statistical models are to be built. Objects are removed from that stream when they fall out of the time window. (3) Drools business rules are called. (4) Business rules apply predictive functions based on statistical models to the business object.

5 Testing and Evaluation

In order to validate the proposed solution pattern and to verify its architectural and design approaches, we have implemented the Business Anomaly Detector in Oracle SOA/BPM Suite environment which handled real-life financial controlling business processes. The prototype implementation served as a proof of concept and was used as a testbed to verify the effect of aggregate materialization on the performance of aggregation rules processing.

The impact of our aggregate materialization techniques has been experimentally verified using a 4-CPU, 16 GB RAM Linux machine. Our test business anomaly detection rule alerted when an invoice exceeded the 30-day moving average for a customer. The number of customers was set to 100, the daily number of invoices per a customer varied between 1 and 1000 (uniform distribution). The performance metric was the number of invoices processed per second. Throughout the experiments aggregate materialization resulted in almost constant throughput of 10000 invoices processed per second. With materialization turned off, the performance degraded to an unacceptable level (i.e., the system unable to process invoices in real time) already around 100 invoices per a customer, which clearly showed that aggregate materialization is a must for aggregation-based rules.

6 Summary

We have proposed a new solution pattern called Business Anomaly Detector and provided good practices for its implementation. The Business Anomaly Detector can be perceived as an infrastructural service, intercepting (explicitly or implicitly) business

objects from SOA BPM business process flows in order to detect anomalous behavior. The good practices include asynchronous architecture, four types of anomaly detection rules, aggregate materialization, and off-line learning of discovery-based business rules. A prototype system has been implemented according to the proposed solution pattern and has been validated in a real-life environment.

The current deployment architecture is based on Java EE, JMS/Kafka, SOAP technology stack. Our future work will focus on development of alternative integration interfaces and runtime platforms, including application containerization and RESTful Web Services (with business events represented in JSON).

References

1. Van der Aalst, W., Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distrib. Parallel Databases **14**(1), 5–51 (2003)
2. Agyemang, M., Barker, K., Alhajj, R.: A comprehensive survey of numeric and symbolic outlier mining techniques. Intell. Data Anal. **10**(6), 521–538 (2006)
3. Barnett, V., Lewis, T.: Outliers in Statistical Data, 3rd edn. Wiley, Hoboken (1994)
4. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture Volume 1: A System of Patterns. Wiley, Hoboken (1996)
5. Chaterjee, S.: Messaging patterns in service-oriented architecture (parts 1 and 2). Microsoft Archit. J. (2, 3) (2004)
6. DeFee, J., Harmon, P.: Business activity monitoring and simulation. In: Fischer, L. (ed.) Workflow Handbook, pp. 53–74. Future Strategies Inc., Lighthouse Point (2005)
7. Dodani, M.: Where's the SOA Beef? J. Object Technol. **3**(10), 41–46 (2004)
8. Dustdar, S., Schreiner, W.: A survey on web services composition. Int. J. Web Grid Serv. **1**(1), 1–30 (2005)
9. Hodge, V., Austin, J.: A survey of outlier detection methodologies. Artif. Intell. Rev. **22**(2), 85–126 (2004)
10. Paschke, A.: Design patterns for complex event processing. In: Proceedings from Distributed Event-Based Systems Symposium (2008)