Rule-based monitoring with event correlation for business process compliance

Ping Gong[1]*, Zaiwen Feng[2], Jianmin Jiang[1]
1. Computer science department, Fujian normal university,
2. State key laboratory of software engineering, Wuhan University
*pinggong@fjnu.edu.cn

Abstract

Business processes compliance monitoring checks whether running business processes comply with involved semantic constraints, i.e., compliance rules. Business processes in modern enterprise are rarely supported by a single and centralized workflow system, but instead implemented over different applications (e.g., CRM, ERP, WfMS, and legacy systems). The running data (i.e., event) about process executions are scattered across these applications. Under the circumstances, understanding the compliance of running processes entails the compliance monitoring enabling to correlate events from different applications and even different cases (event correlation herein is identifying events related to the same compliance rule instance). This paper introduces a framework named as bpCMon for business process compliance monitoring. bpCMon consists of an expressive compliance rule language ECL and a rule system ERS. ECL is a pattern-based formal language for specifying compliance rules of multiple process perspectives, and also allows for describing event-correlation condition. ERS, generated from compliance rules in ECL, in turn plays as a compliance monitor enabling to correlate events efficiently by means of an indexing structure created from event-correlation conditions. The applicability of bpCMon is demonstrated by experiments on a real-world data set. Overall, bpCMon enables business process compliance monitoring meeting real-world requirements.

Keywords: Business process compliance, compliance monitoring, event correlation

1. INTRODUCTION

Business process compliance (BPC) requires that business processes are executed in conformance with prescribed and approved sets of compliance rules. The latter may stem, for example, from norms, standards, and laws (Sadiq, 2011). In general, there are compliance checking approaches of various kinds taken on different phases of process life cycle to enforce the BPC, e.g., a-priori checking at design time or a-posteriori checking based on the event logs of completed process instances.

However, a-priori checking is not always sufficient, since process instances may deviate from prescribed process implementations (Schonenberg, 2008). Furthermore, in many enterprise systems, processes are not model-driven, but more or less hard coded in the respective system (de Lenoi, 2016). In turn, a-posteriori checking might be inapplicable for decision making when quick reaction is needed for compliance violations. These thus emphasize the need for run-time compliance checking, i.e., compliance monitoring.

1.1 Problem statement

Business processes in modern enterprise are rarely supported by only one centralized workflow system. Instead, business processes as well as related compliance rules may refer to activities whose executions are supported by different applications (e.g., CRM, ERP, WfMS, or legacy systems) (Reza Montahari-Nezhad, 2011). The information about processes executions, i.e., events, which refer to the facts about activity execution, are scattered across different applications, and also in many cases there does not always exist in-build mechanisms, with which events are collected and correlated to process instances (Perez-Castillo, 2014). In this context, understanding the compliance of the executions of processes with respective to involved compliance rules is a challenging issue.

Unlike existing works on BPC, which in general implicitly assume the input events are already correlated correctly to the same process instance (Ly, 2015), the compliance monitoring in this circumstance is required to enable to correlate events, which may be generated separately from different applications and even different process instances (also known as cases). Note that in this work, we assume that events are collected and fed into compliance monitors through relevant message oriented middleware (MOM, e.g., ActiveMQ or Kafka) as *Figure 1*. The event correlation here is identifying events related to the same compliance rule instance, which corresponds to one-time triggering of compliance rule. Considering *Example 1* from financial domain, it consists of compliance rules **B1**-**B3**, which arm at frauds prevention in the context of bank transactions. For the compliance rule B1, which concerns with bank transfer transactions, it was triggered by one *transfer* event, if there was another related *transfer* event with an amount exceeded €10,000 (suspicious transfer) occurred at maximum 30 days before. Here, these two *transfer* events, which occur from different cases, are correlated with respective to the rule if they satisfy a certain

DOI: 10.29268/iciot.2016.0006

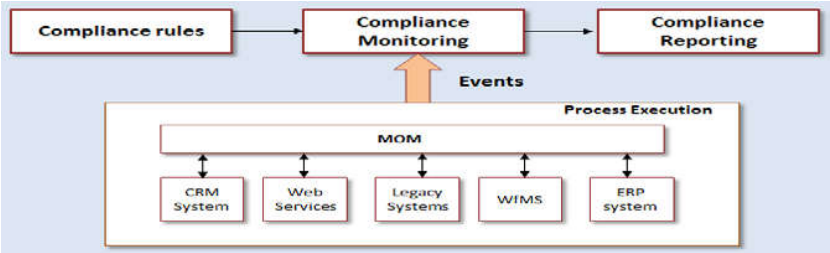

Figure 1. Business process compliance monitoring

Example 1. Following compliance rules address the prevention of frauds in banking domain [Basin, 2013, 2015]:
B1. *Every bank transfer of a customer, who was involved in a suspicious bank transfer (e.g., with an amount greater than €10,000) within the last 30 days , must be reported afterward within 2 days.*
B2. *The sum of withdraws from credit card account within 30 days, must not exceed the limit of €10,000.*
B3. *For each user, the number of withdrawing peaks over the last 30 days does not exceed a threshold of 5, where a peak is a value at least twice the average over some time window (30 days).*

condition, e.g., with the same customer and/or account. From then on, the *transfer* event, the trigger of the rule, was required to be followed by one report event, which was correlated for the triggered rule by the information of, e.g., customer and/or transaction ID, and may be generated from relevant reporting system.

On the other hand, compliance rules, as constraints for executions of business processes, usually refer to multiple process perspectives, including:

- Control flow perspective, which refers to the occurrence, absence, and temporal order of activities, e.g., compliance rule **B1** requires report activity after transfer activity.
- Data perspective, which refers to constraints on attributes of activities, e.g., compliance rule **B1** restricts the amount of transfer activity, i.e., greater than €10,000.
- Time perspective refers to time-interval relations between activities, e.g., compliance rule **B3** refers to the time interval of a period of 30 days as well as within 2 days.
- Resource perspective refers to roles and organizations who are related to perform activities, e.g., compliance rule **B2** concerns with withdraw activity performed by customer.

Altogether, to address the aforementioned challenge, the framework for compliance monitoring, which normally consists of the language for specifying compliance rules and the monitor for ensuring the BPC, needs to meet following requirements:

- (R1) Multiple process perspectives. It must allow for specifying and monitoring of compliance rules that refer to multiple process perspectives, including the control flow, data, time, and resource perspectives.
- (R2) Event correlation. As stated before, compliance monitoring in modern enterprise is required to enable correlating events, which may be generated from different process instances or even applications, for involved compliance rule. The compliance rule language hence needs to support specifying the correlation between events, and then the compliance monitor should have the capability of identifying those correlated events accordingly.
- (R3) Efficiency. Under the context of potentially large numbers of running process instances, efficient monitoring mechanism is needed to deal with correlating and reasoning over large volumes of events.

1.2 Contributions

This paper introduces the *bpCMon* [1] framework, which consists of two parts: event-based compliance rule language (*ECL*) and event reaction system (*ERS*). More precisely, major contributions made in this work are summarized as follows:

- Event-based compliance rule language (*ECL*): *ECL* builds on the notions of *event* and *event-relation* patterns. It allows for specifying compliance rules referring to multiple process perspectives. Furthermore, it provides the way to specify correlations between events.
- Event reaction system (*ERS*): *ERS* is a rule system that serves as compliance monitor for compliance rules in *ECL*. *ERS* is able to cope with multiple process perspectives. By a tree-based index structure, *ERS* allows for efficiently correlating events. *ERS* differs from existing rule systems (e.g., Drools (Drools, 2015), Jess (Friedman-Hill, 2003)), not only in the rule form, but also in the structure of working memory.
- Evaluations for *bpCMon*: we implemented a proof-of-concept prototype and applied it to a real-world data set from Dutch Academic Hospital.

[1] https://github.com/PingFair/bpCMon

The remainder of this paper is structured as follows: Related work is discussed in Section 2. Section 3 defines the language ECL. The ERS compliance monitor is presented in Section 4. Section 5 is the implementation and evaluation of *bpCMon*. The last section concludes the paper and also gives an outlook on future work.

Table 1. Overview on related works

Type	works	R1_c	R1_d	R1_t	R1_r	R2	R3
Graph-based	MonbuconEC(Montali,2010)	+	-	+/-	-	-	a
	BPMN-Q(Awad,2011)	+/-	+/-	-	-	-	n.a
	eCRG/CRG (Ly 2011, Knuplesch 2015)	+	+	+	+	-	n.a
Logic-based	MonbuconLTL(Maggi 2011)	+	-	-	-	-	n.a
	MonPoly(Basin 2015)	+	+	+	+	+	a
Pattern-based	Giblin, 2006	+/-	+/-	+/-	+/-	-	n.a
	Turetken, 2012	+/-	+/-	+/-	+/-	-	n.a
	Mulo, 2013	+	-	-	-	-	n.a
	bpCMon	**+**	**+**	**+**	**+**	**+**	**a**
Others	MOP (Chen, 2009)	+	+/-	-	+/-	-	a
	LOGFIRE/Drools (Havelund, 2015)	+/-	+	+	+	+	a

'+' full support, '+/-' partial support, '-' not supported; 'av.' available, 'n.a.' not available.

2. Related Work

The monitoring of business processes compliance requires first specifying compliance rules by languages in formal and unambiguous way. Based on the characteristics of adopted languages, existing works of this category can be classified into graph-based, logic-based, and pattern-based ones.

MonbuconEC (Montali, 2010) specifies compliance rules by the graph language Declare (Pesic, 2006), which is extended to support specifying metric time as well as activity life cycle. The rules in Declare are then translated into axioms of the event calculus, on which the monitoring mechanism of MonbuconEC depends. Based on the Declare as well, MonbuconLTL (Maggi, 2011) translates the rules in Declare into Linear Temporal Logic (LTL), and relies its monitoring on colored automata. However, these two works mainly aim at control-flow perspective, and have limited supports for the data perspective, and also consider neither event correlation nor aggregations. Works (Ly, 2011, Knuplesch, 2015) utilize the (extended) Compliance Rule Graph Language (eCRG) to support the control flow, data, time, and resource perspectives. They annotate eCRG with markings, colors, and texts to describe the states of compliance rules. These annotations hence enable to detect and highlight root causes of compliance violations. In addition, BPMN-Q (Awad, 2011) is also a graph language extended from BPMN and enables to specify compliance rules of control-flow as well as partly data perspectives.

MonPoly (Basin, 2015) aims at monitoring compliance rules in metric linear temporal logic (MLTL). By rewriting rules for MLTL formulas, MonPoly enables to support compliance monitoring for various perspectives, including event correlation. To specify compensations for violations, work (Giacomo, 2014) proposes the Linear Dynamics Logic (LDL) by combining LTL with regular expression. However, it only provides a theory without any further performance data.

The REALEM framework (Giblin, 2006) enables specifying compliance rules by three compliance rule patterns. These patterns are then translated into executable

rules for the specific infrastructures. Note that, our approach includes more patterns, besides these three patterns. Works (Turetken, 2012, Elgammal, 2014) proposed a pattern-based language CRL to capture compliance requirements. The patterns included in CRL range from control-flow to data as well as resource perspectives. However, these works mainly concern with introducing rich compliance patterns. In addition, works (Mulo, 2013) proposes a domain specific language (DSL) for specifying compliance rules and apply complex-event processing for compliance monitoring. However, these works are restricted to the control-flow perspective yet.

MOP (Feng, 2009) is claimed the fastest monitor for parametric finite state properties with formalism-independent specification. In MOP, approaches of trace slicing and indexing structure enable monitor to attain quite efficient performance. However, MOP does not yet support data aggregation. The work (Havelund, 2015) implements a rule system based runtime engine, i.e., LOGFIRE, by ramifying RETE algorithm. The ramifications include introducing a double-indexing among related nodes for speeding up token matching. However, once some fact in the node was updated, each related nodes as well as the indexing mappings would need to be updated synchronously. These thus might undermine engine efficiency. Besides, from the specification aspect, the work has limited supports for specifying control-flow perspective in high level.

To sum up, representative works from above different fields are selected and compared according to their supports for the monitoring requirements R1-R3, wherein, R1 is further divided into $R1_c$, $R1_d$, $R1_t$, and $R1_r$ to correspond to the perspectives of control-flow, data, time, and resource, respectively. As indicated by *Table* 1, *bpCMon*, together with MonPoly, fully supports the requirements R1-R3. But for the efficiency requirement, as shown in the evaluation section of our technical report (Gong, 2016), *bpCMon* is comparable to the known fastest monitor MOP, and outperforms MonPoly as well as rule engine Drools, which is selected as a representative for RETE-based rule systems, since LOGFIRE is not yet publicly assessable.

3. EVENT-BASED COMPLIANCE LANGUAGE

Basic concepts of this work include *event, event instance*, and *event matching*. In this work, events are the way of describing the useful and relevant facts about activities executions, e.g., for the bank transfer, suspicious transfer event refers to the executed transfers with amounts greater than €10,000. On the other hand, an event instance is used to describe the fact of one activity executed at certain time point, e.g., one transfer with amount €12,000 occurred at 12:00 01/09/2016, is an event instance, which in fact is belonged to the suspicious transfer event. There is then one relationship between event and event instance, i.e., event matching, to depict if one event instance is belonged to one event.

3.1 The definition of ECL

Compliance rule in this work is a constraint referring to the desired property of executions of processes. As implied in (Dwyer, 1999), most of finite-state system properties can be classified into two basic patterns, *Precedence* and *Response*, i.e., event *p* is always *preceded* (*followed*) by event *q*. They can also be equally rewritten as, whenever event *p* occurs, event *q* must occur *before* (*after*). Note that, within the patterns, there are some essential ingredients: the *trigger* for activating patterns, e.g., the occurrence of event *p*; the *target* constrained by patterns, e.g., event *q* which must occur before (after) when the rule was activated by *p*; the *scope* as pattern takes effects on, e.g., a trace scope specified by the qualifier always. These ingredients form the basis for control-flow patterns. However, as mentioned above, compliance rules usually refer to multiple perspectives, including control-flow, data, time, and resource. The patterns hence need to be extended to include more elements for other perspectives. More specifically, within *ECL*, for data and. resource perspective, introduced elements include the structure of event, and event-correlation condition; meanwhile the time constraint is also introduced for metric-temporal relations among events. We term as *event-relation patterns* for the patterns which are composed by these ingredients. Currently, the patterns introduced in the ECL are: *before, after, when, beforeSince,* and *afterUntil*. Syntactically, these patterns, as predicates together with events as variables, form a signature for *ECL* definition.

Definition 1. (*ECL*) For an event e and an integer t, the event-based compliance language ECL can be defined as follows:

ecl := [event]+ [tmf]+

tmf := always emf | exists emf | ! tmf | tmf1 && tmf2

emf := f | ! emf | emf1 & emf2

f := e | ors(e) | constr when f | before(tc, f, e, econ)
| after(tc, e, f, econ) | beforeSince(tc, f2, f1, e, econ)
| afterUntil(tc, e, f1, f2, econ)

econ := $e_1.attr_1 = e_2.attr_2$ | $econ_1$ && $econ_2$

tc := [t, right)

right := t [d | h | m | s]

From the definition, the structure of *ECL* consists of two parts, i.e., events part and rules part. The events part is for events definitions which form an alphabet for *ECL* formula, whereas rules in rule part are specified in trace-matching formulas (TMF for short) *tmf*, which is defined by extending

event-matching formulas (EMF) *emf* from time-point scope to the trace scope by *always* or *exists* qualifiers as well as negation and conjunction operators. An EMF *emf* is built on a set of atomic formulas, which correspond to event-relation patterns. Within the atomic formula, besides the control flow specified for the involved events, it also includes time constraints *tc* and events correlation condition *econ*.

Example 2. Every bank transfer of a customer, who has within last 30 days been involved in a suspicious transaction (e.g., with amount greater than 10,000), must be reported within 2 days.

```
//events part
  e1 = (1, 'transfer', [ customer, amount, tId ]) ;
  e2 = (2, 'transfer', [ customer, amount > 10000 ]) ;
  e3 = (3, 'report', [customer, tId]) ;
// policy part
rule1 =
always( before( [0,31d),e2,e1,e1.customer=e2.customer)->
        after( [0,3d),e1,e3, e1.customer=e3.customer &&
                                        e1.tId=e3.tId ) )
```

4. EVENT REACTION SYSTEM

To monitor the fully featured *ECL*, it is necessary to have a uniform and powerful analysis mechanism. In this section, event reaction system (ERS) is proposed, which in fact is the rule system plus working structure.

Definition 2. (Event Reaction System) Event reaction system is a 2-ary tuple *ers* = (*rs*, *ws*), where:

(1) *rs* is a rules system with reaction rules, where reaction is a sequence of operations over working structure;

(2) *ws* is a working structure in charge of organizing instances for their efficiently storing and assessing.

Different to the net-like working memory in RETE algorithm or other rule engines, ERS working structure is of tree structure including index and bounded queue.

4.1 The rule system of ERS

The rule system of ERS specifies relevance reactions when event instance is read and matched. It is therefore consisted of reaction rules in a form, $e \rightarrow c\_reaction$. Semantically, the reaction rule means, when the trigger *e* is matched, then reaction *c_reaction* is invoked and operations in *c_reaction* are executed as specified, where operations include various operators on working structure as *Table 2*. A rule system *rs* is *deterministic* if there do not exist any two rules in R with the same let hand, and ERS is *deterministic* if its rule system is *deterministic*. In this work, only deterministic rule system is considered.

For right hand of rule, i.e., *c_reaction*, it could be an operation *op* as well as a compositional reaction, which is composed by operations with sequential operator "**;**". For operation *op*, it could be an atomic operation, e.g., delete operation *#d*, write operation *#w*, etc., a conditional operation formed by conditional operator "**? :** ", or a composite operation by chaining operator "[illegible] Note that, the operators of sequential and chaining have different semantics and usages. The sequential operator is used to merge rule systems by connecting reactions with same trigger event to form one composite reaction, and when rule was invoked, each reactions within the composited reaction would be executed sequentially; whereas the chaining

Table 2. The descriptions for each operation in rule system

Operation	Description
#d(i, ta, tr)	when *tr* instance was occurred, delete all correlated *ta* instances from related value structure in *i* IIS.
#g(i,ta,tr,tc)	when *tr* instance was occurred, get all correlated *ta* instances within time interval *tc* from related value structure in *i* IIS.
#w(i,ta,tr)	write *ta* instance as a new fact into related value structure in *i* IIS for correlated *tr* instance.
#fail/succ(t,ta,e,tr)	when *tr* instance was occurred, create a new *t*-type failure/success.
#next	terminate the execution of current operation and go to next operation if there is, otherwise read the next event instance.
#ge(i,ta,tr,tc)	when *tr* instance was occurred, check whether there exist correlated *ta* instances within time interval *tc* in related value structure of *i* IIS.
#empty(i,ta,tr)	check whether it is empty for the value structure related to *ta* and *tr* in *i* IIS.
#eval(constr)	evaluate whether the attributes constraint *constr* is satisfied in current moment
#tcm(ta, tr)	compare the temporal order of *ta* instance with *tr* instance

where: *ta*, *tr* are events representing respectively the target and trigger of operation, *i* is a symbol for instances indexed structure IIS within working structure, *tc* is the time constraint, and *constr* is the attribute constraints for event.

operator is used to chain operations to one composite operation. The operations within the composite operation are normally executed sequentially, however, if there was one terminating operation among them, e.g., *#next*(), then the followed operations would be skipped. For the more formal and complete details, please refer to the technical report (Gong, 2016).

Example 3. For a TMF always (after(_, e6, e7, e6.caseID = e7.caseID)), which specifies that, for each trace, whenever e6 occurs, then e7 should occur after. Its rule system can then be specified as follows:

```
RS :
% when e6 instance occurs, writing e6 instance into the
% value structure for e6 and e7.
e6 -> #w(1,e6,e7)
% when e7 instance occurs, if there is related e6instance
% in the structure for e7 instance, delete all such e6
% instances and create success for such e6 instances
% and e7 instance; otherwise read the next.
e7 -> #ge(1, e6, e7) ? #d(1,e6,e7).#succ(2,e6,e7 ):#next()
% when end event occurred, if the structure of e6 for
% e7 is not empty, then create violation instances of
% type 3 for each such e6 instance; otherwise read the
% next.
end -> !#empty(1,e6,e7) ? #fail(3,e6,e7):#next()
```

4.2 Instances indexed structure IIS

Value structure is an essential part for executing operations. To efficiently assess the correlated target instances for certain trigger in executing an operation, for example, *#g*(0,*ta*, *tr*), the value structure, storing *ta* instances, is equipped with an index which is defined based on the event-correlation condition between events *ta* and *tr*. In this work, we term the instances value structure with index as instances indexed structure (IIS) and its structure is depicted as *Figure 3*, which is of four layers and with storing mechanism, i.e., queue used in this work, as its leaf nodes.

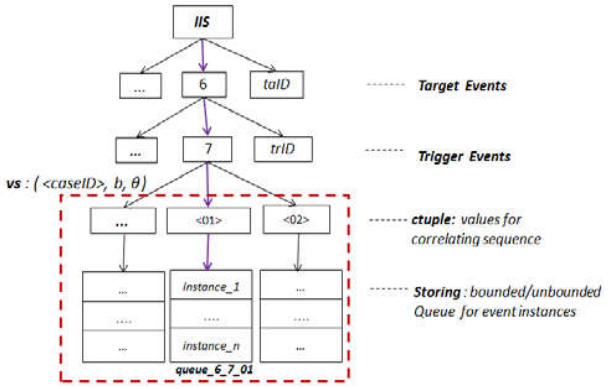

Figure 3. Instances Indexed Structure IIS

Within the IIS, its core component is the value structure, which is used to store target event instances by queues and further equips such queues with indexes, which are determined by correlating sequence. As depicted with dashed rectangle in *Figure 3*, a value structure, defined as (< *caseID* >, *b*,_), where <caseID> is the correlating attributes sequence and *b* is the bound of queue, semantically corresponds to a set of pairs of queue together with related index values (i.e., *ctuple*), for example a queue *queue_6_7_01* and a *ctuple* < 01 > in the figure.

In fact, the IIS is built on following two basic operations during the compliance monitoring for data perspective, i.e., get instances and write instance. However, as for write operation, after target instances stored, there are two subcases with subtle differences regarding to whether the stored instance requires the desired trigger to be occurred after. IIS is then divided into two types: *beforeIIS*, wherein the stored target instance does not require certain trigger instance must occur after, and *afterIIS*, where each stored target instance requires desired trigger instance must occur after. For example, for the operation *#w(1, e6,e7)*, it writes occurred *e6* instance into afterIIS, and for each stored *e6* instance, it requires desired *e7* to be occurred; on the other hand, if changed 1 to 0 in the operation, the stored *e6* instance in *beforeIIS* would not require *e7* instance to be occurred after. The working structure is then consisted of beforeIIS, afterIIS, failures container, and success container. During monitoring, it stores relevance data including event instances as well as successes and failures. The compliance states, i.e., *compliant*, *partially compliant*, and *violated*, are determined by the configurations of working structure, which also include contents of containers of failures and successes.

Due to the rule system and working structure, ERS gains the capability of computing the compliance of running trace with respect to related compliance rules. To leverage the power of ERS, compliance rules in ECL need to be translated into ERS-based monitors. Based on the structure of ECL formula, there are two translations for EMF and TMF formulas. They share similar steps, i.e., creating from formula working structures as well as rule systems. For the more detailed, please refer to the technical report.

5. IMPLEMENTATION AND EVALUATION

5.1 Implementation

The basic structure of ERS-monitor is consisted of three parts, working interface, working structure, and rule system:

- Working interface: acts as a data reader in charge of reading, filtering and matching instances from data source. It is implemented as Java interface `WorkingInterface`, including below essential methods:
  - `Instance next()`: read the instance from the data source;
  - `Event matchToEvent(Instance inst)`: match the instance to event;
  - `void init()` and `void close()`: initialize and close the working interface.
- Working structure: it is the core component of ERS, which is implemented as Java class, including beforeIIS,

Algorithm 1: basic monitoring procedure of ERS-monitor

```
1  var inst, event, assign;
2  var reaction;
3  while true do
4      inst = workingInterface.next();
5      event = workingInterface.matchToEvent(inst);
6      if event == null then
7          continue;
8      reaction = ers.rs.getReaction(event) ;
9      if reaction ≠ null then
10         assign = new Assignment() ;
11         assign.add(inst) ;
12         reaction.doReaction(ers.workingStructure, assign) ;
13         if ers.workingStructure.fCon.isNotEmpty() then
14             ers.workingStructure.responseF() ;
15         if ers.workingStructure.sCon.isNotEmpty() then
16             ers.workingStructure.responseS() ;
17     if event == endOfRuning then
18         break;
```

afterIIS, and success/failure containers as well as basic operations over queues.

- Rule system: it is also implemented as a Java class including methods for manipulating rules:
  - `Reactions getReaction(Event event)`: get the reaction for event, where Reactions is a interface including method `void doReaction(WorkingStructure ws, Assignment assign)` to execute the reaction.
  - `RuleMap rulesJoin(RuleMap other)`: merging two rule systems, wherein the sequential operator is used to connect reactions with the same trigger.

Note that, the real implementation is far more complicate than the above. But it is enough for presenting basic monitoring procedure of ERS-monitor.

As described in the Algorithm 1, ERS-monitor reacts on each matched instances until the end of running is reached. For each time of reaction, an assignment is created and also added the just matched instance. The main use of assignment is temporally storing the intermediate results generated during operations executing, e.g. *#ge* operation generates a set of instances after executed. After the reaction, in line 13-16, the monitor responses compliance situations by checking two containers in working structure.

5.2 Evaluations

To evaluate the applicability of *bpCMon* in case-by-case and also across-case context which requires monitor has the capability of event correlation, one test case is adopted which includes the real datasets from the hospital and 16 compliance rules of various perspectives. To read the data from the hospital logs in XES format (Gnther et al., 2014), class `XESWorkingInterface`, which implements the interface `WorkingInterface`, is developed based on the OpenXES libary. The `XESWorkingInterface` is in charge of generating interested event instances by parsing, selecting, and merging related event-scope and trace-scope attributes values. These instances also include instances of `endOfCase` and `endOfLogs` for the end of trace and logs respectively. After the rules are specified in ECL formulas, relevant ERS monitors are generated automatically from these formulas.

The evaluation is performed on Luna version of Eclipse IDE with jdk-1.8.0 40 in laptop with win7 64-bit OS, Intel(R) i5 CPU2.4G, and 8G RAM. It is consists of two

Table 3. Violations for incompliant rules in MCC and MAC

rules	R1	R2	R4	R5	R6	R7	R8	R9	R11	R12	R14
#violation(MCC)	855	192	352	660	2169	75	958	848	8	45	823
#violation(MAC)	855	192	352	660	2169	75	958	848	8	45	823

Table 4. Performance of bpCMon monitor running 5 times for the hospital logs

rules	#eventInstances	#violation	time(sec)	memory(MB)
R1-R16	151419	7085	5.39 / 5.49	378.3 / 350.2
			4.936 / 5.492	326.5 / 294.59
			5.143 / 5.406	377.53 / 335.19
			5.361 / 5.648	293.69 / 240.18
			5.370 / 5.471	385.51 / 328.45

phrases: at first phrase, the ERS monitor runs over the logs in case-by-case 100 times, i.e., monitoring in case-by-case (MCC). Within each running, a new composite ERS monitor is regenerated by merging each ERS monitors in randomly order. At second phrase, a composite ERS monitor runs over the across-case logs which is generated by mixing each case from the logs but keeping the order for each event instances from the same case, i.e., monitoring in across-case (MAC). After two phases tests, as shown in *Table 3*, no matter in MCC or MAC, the logs is compliant with five rules, R3, R10, R13, R15, R16, and for other rules, there are various numbers of violations. The findings are that: the independency of working structures for these formulas is demonstrated by the same violations number #violation in 100 times running from first phrase, and thanks to the event correlation of ERS, the applicability of bpCMon in across-case context is also indicated by the same violation numbers in MCC and MAC.

As for the running cost, it mainly consists of two parts: for ERS monitor running and for OpenXES caching all the event instances. Within *Table 4*, the symbol "/" is used to delimit the cost for ERS monitor (at the front) and OpenXES file caching (at the behind). From these cost data, it can be implies that, ERS-monitor is of practically efficient. For the factors influencing the performances of ERS-monitor, the event features and reaction rules length would be the main factors. Event features here refer to the event structure property and the sub-event relation among events. If the event consists of complex attributes constraints or there exist couples of events with sub-event relation, then the cost of event matching as well as reaction would be increased. In the test case, there are three pairs of events with sub-event relation. As for the memory cost of ERS monitor, it might be related to the working manner of OpenXES: loading all the data from logs file into the memory and then the data available for use. After data loaded, there is a memory overhead for ERS-monitor ranged from 8% to 20%.

6. CONCLUSIONS

Understanding the compliance of running processes in current enterprise is challenging since compliance monitoring needs to enable to correlate events from different applications and even different cases. This work presents a compliance monitoring framework *bpCMon*, which consists of: an event-based compliance language (ECL) for specifying compliance rules as well as event correlation condition, and event reaction system (ERS) as engine for compliance monitoring. Experiments on a real life hospital logs over 16 compliance rules indicate the applicability of *bpCMon* in case-by-case as well as across-case context, since its capability of correlating event.

As for the future works, from the practical view, a friendly interface is needed to support users specifying and managing their compliance rules; from the theoretical view, it is also important to further devise methods to resolve the rules conflict issue and the intersecting of working structures as well as their possible relations. In fact, such solutions would be the basis for addressing the scalable issue of the *bpCMon* when considering huge numbers of rules. Finally, further evaluations are also needed for the soundness of *bpCMon*.

7. ACKNOWLEDGMENT

This work was supported in part by NSFC 61100017 and NSF of Fujian Province No.2014J01221.

8. REFERENCES

Sadiq, S. (2011): A roadmap for research in business process compliance. In: W. Abramowicz, L. Maciaszek, K. Wecel (Eds.) BIS 2011 Workshps, LNBIP 97, pp. 1-4

Schonenberg H. Mans R., Russell N., Mulyar N., and van der Aalst W.M.P. (2008): Process flexibility: A survey of contemporary approaches. In: J.L.G. Dietz et al. (Eds.):CIAO 2008 and EOMAS 2008, LNBIP 10, pp. 16-30

de Leoni M., van der Aalst W.M.P., and Dees M. (2016): A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Information Systems, 56:235-257

Reza Montahari-Nezhad H. Saint-Paul R., Casati F., and Benatallah B. (2011): Event correlation for process discovery from web service interaction logs. The VLDB Journal 20:417-444

Perez-Castillo R., Weber B., Garcia-Rodriguez de Guzman I. (2014): Assessing event correlation in non-process-aware information systems. Softw Syst Model 13:1117-1139

Drools(2015):http://docs.jboss.org/drools/release/6.3.0.Final/droolsdocs/html_single/index.html

Friedman-Hill E. (2003): Jess in action: Rule based systems in Java. Manning publications.

Ly L.T., Maggi F.M., Montali M., Rinderle-Ma S., van der Aalst W.M.P. (2015): Compliance monitoring in business processes: functionalities, application, and tool-support. Information Systems. 54, 209–234

Elgammal A., Turetken O., van den Heuvel W. (2014): Formalizing and applying compliance patterns for business process compliance. Software & Systems Modeling, pp 1-28

Montali M., Maggi M.F., Chesani F., Mello P., and Van der Aalst W.M.P. (2010): Monitoring business constraints with the event calculus. ACM Transactions on Embedded Computing Systems, Vol.9, No. 4, 1-29

Pesic M. and Van der Aalst W.M.P. (2006): A declarative approach for flexible business processes management. In: J. Eder, S. Dustdar et al.(Eds.): BPM 2006Workshops, LNCS 4103, pp.169-180

Maggi F.M., Montali M., Westergaard M., Van der Aalst W.M.P (2011).: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: S.Rinderle-Ma, F. Toumani, and K. Wolf(Eds.) BPM 2011, LNCS 6896, pp.132–147

Ly L.T., Rinderle-Ma S., Knuplesch D., and Dadam P. (2011): Monitoring business process compliance using compliance rule graphs, In: R. Meersman, T.Dillon, and P. Herrero(Eds.): OTM 2011, Part I, LNCS 7044, pp. 82-99

Knuplesch D., Reichert M., and Kumar A. (2015): Visually monitoring

multiple perspectives of business process compliance. In: Hamid, R. M. N., Jan, R., and Matthias,W.(Eds.), BPM 2015. LNCS 9253,pp. 263-279

Awad A.,Weidlich M., andWeske M. (2011): Visually specifying compliance rules and explaining their violations for business processes. J. Visual Languages and Computing. 22(2011), pp. 30–55

Basin D., Klaedtke F., Mller S., and Zlinescu E.: Monitoring metric first-order temporal properties. In: Journal of ACM, 62(2),pp:1-38 (2015)

Basin D., Klaedtke F., Mller S, etc. (2013): Monitoring of temporal first order properties with aggregations. In: proceeding of RV 2013, LNCS 8174, pp. 40-58

Giacomo G., Masellis R.D., Grasso M., Maggi M.F., and Montali M. (2014): Monitoring business metaconstraints based on LTL and LDL for finite traces. In: Shazia W. S., Pnina S., Hagen V.(Eds.) BPM 2014. pp. 1–17

Giblin, C., Muller, S., Pfitzmann, B. (2006): From regulatory policies to event monitoring rules: towards model-driven compliance automation. Technical report RZ3662, IBM Research.

Turetken O., Elgammal, A., and van den HeuvelW.J. (2012): Capturing compliance requirements: a pattern-based approach. IEEE Software, IEEE Computer Society

Mulo, E., Zdun, U., and Dustdar, S. (2013): Domain-specific language for event-based compliance monitoring in process-driven SOAs. In: SOCA (2013) 7: 59-73

Mulo, E., Zdun, U., and Dustdar, S. (2009): Monitoring web service event trails for business compliance. In: IEEE International Conference on Service-oriented Computing and Applications(SOCA), IEEE.

Barbon, F., Traverso, P., Pistore, M., and Trainotti, M. (2006): Run-time monitoring of instances and classes of web service compositions. In: IEEE Inernational Conference on Web Services(ICWS'06), IEEE computer society.

Chen F., Jin D., Meredith P.O.N., and RoÅu G(2009).: Efficient formalism independent monitoring of parametric properties. In: ASE 2009, IEEE press, pp. 383-394

Havelund K. (2015): Rule-based runtime verification revisited. In: Journal of Software Tools Technology Transfer, 17:143-170

Gong P., and Knuplesch D. (2016): bpCMon: An Efficient Framework for Business Process Compliance Monitoring. Technical Report UIB-2016-03, Ulm University, In: https://github.com/PingFair/bpCMon.

Dwyer, M.B., Avrunin, G.S., Corbett, J.C. (1999): Patterns property specifications for finite-state verification. In: ICSE 99, Los Angeles CA, ACM, pp. 411-420

Gnther C.W., and Werbeek E. (2014): XES standard definition. In http://www.xes-standard.org/_media/xes/ xesstandarddefinition-2.0.pdf, March 28, version 2.0

Authors

Ping Gong received his PhD degree from SKLSE of Wuhan University in 2009. He now is a lecturer in Fujian normal University. His research interests include, stream computing, business process management, and service computing.

Zaiwen Feng received his PhD degree from SKLSE of Wuhan University in 2009. He now is a lecturer in Wuhan University. His research interests include, requirement engineering, business process management, and service computing.

Jianmin Jiang received his PhD degree from Chengdu computing institute of Chinese Academy of Sciences in 2006. He now is an associate professor in Fujian Normal University. His research interests include, software engineering.