(c) IEEE 2019
Int'l Conference on Process Mining 2019, DOI 10.1109/ICPM.2019.00026

Compliance Monitoring on Process Event Streams from Multiple Sources

Patrik Koenig
University of Vienna
Faculty of Computer Science
Vienna, Austria
Email: patrik.koenig@univie.ac.at

Juergen Mangler
Austrian Center of Digital Production
Vienna, Austria
Email: juergen.mangler@acdp.at

Stefanie Rinderle-Ma
University of Vienna
Faculty of Computer Science, ds:UniVie
Vienna, Austria
Email: stefanie.rinderle-ma@univie.ac.at

***Abstract*—Comprehensive and continuous compliance monitoring is crucial for many process scenarios, e.g., machine-spanning production processes. However, business process execution data is often scattered over several heterogeneous event sources and needs to be seamlessly incorporated into the data structure of the compliance checking system. The proposed COMS approach offers the possibility to define business rules based on process behaviour and the ability to handle events from different information system sources in an integrated way. COMS consists of a generic event data structure to store process information, a rule language, and matching concepts. COMS concepts are prototypically implemented and evaluated based on a real-world event stream from the manufacturing domain.**

I. Introduction

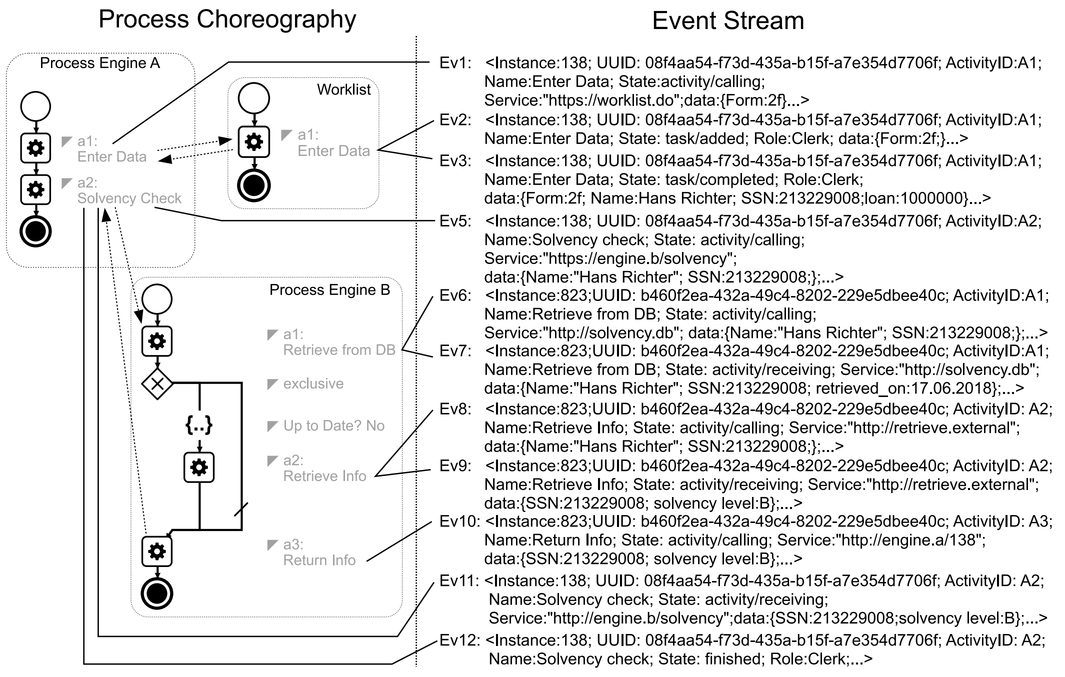
Compliance monitoring is a major challenge for business processes, particularly if the execution information is distributed over heterogeneous event sources [1], [2], for example, in distributed supply chain scenarios [3]. It requires to ensure and verify business rules over several event streams emitted at runtime. Most compliance monitoring approaches define a compliance language, but abstract from the data structure in which the process information is provided by the event stream [1]. This potentially causes uncertainty in matching process and rule information caused by e.g., heterogeneous labels or formats in [4]. Hence, an accessible integration format is crucial to overcome uncertainty.

Figure 1 displays a loan application process which is distributed across multiple systems. In this case a bank clerk receives a loan application and therefore triggers the process by creating a loan request. This process is then enacted on process engine A, where the first process activity requires the bank clerk to enter the customer data of the loan requester into the system. Even though the input process is pervasive for the bank clerk, the input is entered and processed on a worklist component. Hereby the contextual information is sent from process engine A to the worklist by invoking its service. After the information is correctly entered it is returned to process engine A. The same applies to the following solvency check activity which triggers the creation of a process instance on process engine B. In process engine B the solvency level is either retrieved from the database, if current information is available or from external sources.

Every step of the process execution generates multiple events which stem from multiple information systems with different characteristics. To monitor the process execution all of these events may need to be taken into account to assemble a holistic process context. Moreover, by now process information as represented in the event stream of Fig. 1 needs to be matched to a predefined data schema. This schema is mostly either too specialised for a specific process engine or too generic in a way that it holds only basic attributes of a process instance. In [5], for example, a relational data schema is specified, which considers a defined set of process attributes and characteristics. Every bit of information which cannot be matched to the data schema is omitted. While XES [6] is the de-facto standard input file-format for process mining tools, it is not well-suited for partial event streams from multiple sources, which have to be processed at run-time. While many XES concepts are universal, a generic event data structure (i.e. not a file format) optimized for allowing for efficient access to structured data sent and received by tasks is crucial.

Consider the following business rule imposed on the loan example in Fig. 1: *if the loan request is greater or equal to one million, the solvency level of the customer needs to be at least A, a manager needs to process the request, and the solvency information must not be older than two days.* In the case of Fig. 1 the information necessary to check this rule is distributed across multiple systems and hence requires the provision of an integrated and accessible information based on a generic event data structure. In modern compliance management system, rules are separated from the process model to foster re-usability and manageability of business rules. Hereby the rule concept needs to be matched to the process model. This matching has to be system independent to support the integration of different process aware information system. When matching rules and process models the event and activity labels are probably not a strong identifier. Often other aspects of an activity, e.g., the employed resources, can be more relevant. Based on the problem statement three research questions have been defined:

RQ1 What are the common properties of an event which is produced during process execution? Do events and their information adhere to a certain structure which is present



```
Ev1:  <Instance:138; UUID: 08f4aa54-f73d-435a-b15f-a7e354d7706f; ActivityID:A1;
      Name:Enter Data; State:activity/calling;
      Service:"https://worklist.do";data:{Form:2f}...>
Ev2:  <Instance:138; UUID: 08f4aa54-f73d-435a-b15f-a7e354d7706f; ActivityID:A1;
      Name:Enter Data; State: task/added; Role:Clerk; data:{Form:2f;}...>
Ev3:  <Instance:138; UUID: 08f4aa54-f73d-435a-b15f-a7e354d7706f; ActivityID:A1;
      Name:Enter Data; State: task/completed; Role:Clerk;
      data:{Form:2f; Name:Hans Richter; SSN:213229008;loan:1000000}...>
Ev5:  <Instance:138; UUID: 08f4aa54-f73d-435a-b15f-a7e354d7706f; ActivityID:A2;
      Name:Solvency check; State: activity/calling;
      Service:"https://engine.b/solvency";
      data:{Name:"Hans Richter"; SSN:213229008;};...>
Ev6:  <Instance:823;UUID: b460f2ea-432a-49c4-8202-229e5dbee40c; ActivityID:A1;
      Name:Retrieve from DB; State: activity/calling;
      Service:"http://solvency.db"; data:{Name:"Hans Richter"; SSN:213229008;};...>
Ev7:  <Instance:823;UUID: b460f2ea-432a-49c4-8202-229e5dbee40c; ActivityID:A1;
      Name:Retrieve from DB; State: activity/receiving; Service:"http://solvency.db";
      data:{Name:"Hans Richter"; SSN:213229008; retrieved_on:17.06.2018};...>
Ev8:  <Instance:823;UUID: b460f2ea-432a-49c4-8202-229e5dbee40c; ActivityID: A2;
      Name:Retrieve Info; State: activity/calling; Service:"http://retrieve.external";
      data:{Name:"Hans Richter"; SSN:213229008;};...>
Ev9:  <Instance:823;UUID: b460f2ea-432a-49c4-8202-229e5dbee40c; ActivityID: A2;
      Name:Retrieve Info; State: activity/receiving; Service:"http://retrieve.external";
      data:{SSN:213229008; solvency level:B};...>
Ev10: <Instance:823;UUID: b460f2ea-432a-49c4-8202-229e5dbee40c; ActivityID: A3;
      Name:Return Info; State: activity/calling; Service:"http://engine.a/138";
      data:{SSN:213229008; solvency level:B};...>
Ev11: <Instance:138; UUID: 08f4aa54-f73d-435a-b15f-a7e354d7706f; ActivityID: A2;
       Name:Solvency check; State: activity/receiving;
       Service:"http://engine.b/solvency";data:{SSN:213229008;solvency level:B};...>
Ev12: <Instance:138; UUID: 08f4aa54-f73d-435a-b15f-a7e354d7706f; ActivityID: A2;
       Name:Solvency check; State: finished; Role:Clerk;...>
```

Fig. 1. Loan Process Event Stream Emitted by Two Process Engines

in every event independent of a process engine?

RQ2 Which data structure allows for the most flexible representation of individual events from distributed systems? How can information from enacted business processes be represented efficiently within a runtime compliance monitoring system? Efficient means that the data should be retrievable without traversing the data structure.

RQ3 What language concept enables the simplest but flexible matching of business rules and process events? What is an efficient and flexible way to define such an matching?

For tackling these questions, we develop a Compliance Monitoring System (COMS). It consists, at first, of an accessible data structure to integrate process event streams from multiple information systems (cf. Sect. II) that avoids transformations as much as possible. This comes in hand with a concept for matching business rules and process models beyond label matching that exploits the data exchanged between the different event streams (cf. Sect. III) . The matching goes beyond label matching; instead it matches process rules to process instances based on semantics, e.g., based on the facts which service is invoked by a single process instance activity and the context of the invocation. The proposed artefacts are technically evaluated based on the COMS prototypical implementation and applied to a real-world data set from the manufacturing domain in Sect. IV. Related work is discussed in Sect. V and the paper concludes in Sect. VI.

II. Generic Event Data Structure

In order to define a generic event data structure for rule and process matching, first it needs to be determined how the information of a process instance is produced and collected. For a process engine, an event is typically produced when the state of a process instance changes, for example, from state "started" to state "finished". Though different granularities of possible states exist, the activities in every process engine follow at least a basic instance activity life-cycle model. This model particularly depends on how the underlying process engine processes single activities. The model of possible states is denoted as *instance activity life-cycle model*.

An instance activity life-cycle model describes the transactions and communication which need to take place to execute a single activity, e.g., a start event, assigning resources to an activity, or the end event. In the following common properties of 4 common life cycle models will be analysed and subsequently transferred into an accessible data model.

MXML [7] defines a process log concept. It includes an instance activity life-cycle concept, that does not define process execution states explicitly, but defines the transitions between them. The **Business Process Analytics Format (BPAF)** [8] focuses on the instance activity life-cycle and defines states rather than transitions. Additionally the BPAF model offers a higher granularity of different possible activity states compared to MXML. It also considers resource allocation as states in the instance activity life-cycle model, as well as seven possible end-states, of which MXML has two. Similar to MXML, **XES (eXtensible Event Stream)** [6] describes a language for process logging and aims to solve the problems encountered in MXML. XES references the BPAF life-cycle model as a possible model to describe an instance activity life-cycle, but introduces its own instance activity life-cycle transitions model. It is similar to BPAF, but also describes the transitions between states. These transitions are similar to those in the MXML life-cycle model. The **CPEE** [9] life-cycle model is a compound instance activity life-cycle model which represents the instance activity life-cycle of the CPEE process engine, including the life-cycle model of a worklist component. The latter is a separate logical system and handles the resource allocation of user based tasks. This makes the system less monolithic as multiple worklists can be connected to the process engine.

Besides the instance activity life-cycle, which is required to correctly address the context of a single events, attributes which need to be present in every event are a unique instance identifier and an identifier for the process activity the event relates to. This is required to correlate all events to a specific process instance and activity. Those two classifiers need to be included into every event otherwise an exact correlation of related events is hardly possible for complex instance activity life-cycle models.

In summary, a single event can be uniquely distinguished based on the following characteristics:

- **A unique instance identifier.** Based on this identifier the process instance can be uniquely identified.
- **An identifier for the activity.** A field or value which identifies a activity within a specific instance.
- **The source system.** As multiple information systems can be involved, this identifies the system the event stems from.
- **The event topic.** Defines the parent state in which context the event was raised.
- **The event name.** It defines the actual event such as calling, assigned, etc. and has to be unique in combination with the event topic within the instance activity life-cycle.

The definition of a source systems provides the possibility to correlate the events of multiple information systems into one process instance trace. Those information systems can either share the same instance activity life-cycle or operate on a different one. This leads to the hierarchic data structure shown in Fig. 2. Every hierarchy level is fully dependent on the combination of all hierarchy levels above. To represent this hierarchical structure, the data structure is defined as interleaved key-value pairs. Finally in the lowest level of the data structure, the actual **event data** is stored as arbitrary semi-structured information, i.e., no structural restriction to the event itself arises.

The example stream in Lst. 1 shows parts of an event stream based on the example in Fig. 1. Lst. 1 also depicts that the actual data is saved in the leaf called “Event Data”, as defined in Fig. 2. The other keys are primarily pointing to the next key which narrows down the specific classification of the event.

```
Instance ID
└ Activity ID
  └ Source System
    └ Event Topic
      └ Event Name
        └ Event Data
```

Fig. 2. Data Structure

Listing 1. Event Stream Represented in Generic Event Data Structure

```
:08f4aa54-f73d-435a-b15f-a7e354d7706f:
  :A1:
    :engine:
      :source: Process Engine A
      :activity:
        :calling:
          UUID: 08f4aa54-f73d-435a-b15f-a7e354d7706f
          label: Enter Data
          endpoint: https://worklist.do
          parameters:
            arguments:
              form: http://worklist.do/form/2f.html
              role: Clerk
              user: Hans Richter
        :receiving:
          label: Enter Data
          endpoint: https://worklist.do
          received:
            - Form: http://worklist.do/form/2f.html
            - Name: Hans Richter
            - SSN: 213229008
            - loan: 1000000
    :Worklist:
      :task:
        :completed:
          UUID: 08f4aa54-f73d-435a-b15f-a7e354d7706f
          activityName: Enter Data
          parameters:
            Form: http://worklist.do/form/2f.html
            Name: Hans Richter
            SSN: 213229008
            loan: 1000000
:b460f2ea-432a-49c4-8202-229e5dbee40c:
  :A1:
    :engine:
      :source: Process Engine B
      :activity:
        :calling:
          UUID: b460f2ea-432a-49c4-8202-229e5dbee40c
          label: Retrieve from DB
          endpoint: http://solvency.db
          parUUID: 08f4aa54-f73d-435a-b15f-a7e354d7706f
          parameters:
            arguments:
              Name: Hans Richter
              SSN: 213229008
        :receiving:
          UUID: b460f2ea-432a-49c4-8202-229e5dbee40c
          label: Retrieve from DB
          endpoint: http://solvency.db
          parUUID: 08f4aa54-f73d-435a-b15f-a7e354d7706f
          received:
            - Name: Hans Richter
            - SSN: 213229008
            - retrieved_on: '2018-03-17T11:21:37.213+02:00'
```

III. Matching Business Rules and Processes

Section III-A starts with a discussion of different approaches for rule and process matching. The rule schema used for flexible matching to processes is described in Sect. III-B. Section III-C finishes with the checking mechanisms.

A. What to match?

A first idea could be to base rule and process matching on unique IDs as every process activity has a unique id based on which it can be distinguished. The mayor drawback in this case is that this unique identifier is hard to maintain within the business rule as well as the process repository. Moreover, in case of process evolution it is hard to determine if the rule still applies since a unique id neither represents the context of an activity nor its execution.

Process matching, mining, and compliance are widely based on label equivalence [10], i.e., two process activities or an event and a process activity are considered as equal if their labels are equivalent. More flexibility was added by going from label equivalence to label similarity based on string similarity metrics such as string edit distance [11]. However, label equivalence and similarity are not useful in the context of multiple and heterogeneous data sources. Hence, in literature it is proposed to shift from label equivalence to attribute equivalence [10], i.e., qualifying a set of attributes of a process activity or event that signifies the equivalence and hence the matching. This is suitable for runtime evaluation as the context of the activity can be determined even though the full execution trace is not available. Examples for attributes comprise resources, service endpoints, and service parameters. In connection with service endpoints it has to be noted that different web services URIs can offer identical web services. Service parameters are sent to a web service during the service call. As discussed in [10], these service parameters provide attributes which describe the context of a service, e.g., the use of a specific form in a specific web service.

The COMS matching approach works with attribute equivalence. For this, in the data structure defined in Fig. 2 every characteristics of an activity, which is included in the events, is represented as an attribute. Due to the hierarchical structure, instead of referring to a fixed attribute set, different attribute levels can be considered for matching. More precisely, to match the process rules to process activities a single identifier or a combination of the above can be used. Moreover, the matching semantics can be defined specifically for each rule as described in the next section, i.e., based on a novel Match-Condition-Action rule structure.

B. Rule Schema

In order to incorporate the flexible definition of the rule matching into the rule the well-known principle of Event-Condition-Action rules as, for example, used in previous work [12] is adapted to a **Match-Condition-Action** rule. Here the **Match** part defines how events are matched to process activities contained in the rule, i.e., based to which attribute set a matching can be successfully conducted. Consider the rule (notation: YAML) as shown in Lst. 2. The **Match** part of the rule defines on which business process activity the rule applies. Multiple activities can be defined and then are internally associated with a symbol, which is later used to describe conditions which relate to multiple activities. Such symbols are represented in Rows 2, 6, and 12 in Lst. 2. The next section of the rule schema contains the conditions. They are checked if an activity pattern (e.g., "A precedes B" [1]) matches the conditions in the match section of the rule. If a match pattern is sufficiently fulfilled, a statement concerning the compliance of the instance can be made. In the match part as well as in the conditions part, logical operations can be used to describe the connection between the different clauses. If the connection is not explicitly defined, an "AND" connection is assumed. A clause can span across multiple rows, but also can be defined within one row. Depending on how the conditions evaluate, the actions defined in the *"if"* or in the *"ifnot"* part are executed. The *"if"* part applies if the outcome of the evaluation of the conditions is true. The *"ifnot"* part is applied if the conditions evaluate as false. The actions are enacted as soon as a definitive outcome can been determined, this can be the case even before every condition is evaluated. For example if all conditions are conjunct, as soon as one evaluates to FALSE, the outcome of the compliance rule is FALSE.

TABLE I
CLAUSE STRUCTURE

left hand side	**operator**	**right hand side**
system > topic > name > data	$==$	"http://example.org/service"
system > topic > name > data	$<=$	300
system > topic > name > time	$before$	system > topic > name > time

Listing 2. Example Compliance Rule
```
 1 match:
 2   a :
 3    − ["engine > activity > calling > endpoint","==","https://worklist.do"]
 4    − ["engine > activity > calling > parameters > arguments > form",
 5       "==","http://worklist.do/form/2f.html"]
 6   b :
 7    − ["engine > activity > calling > endpoint","==","http://engine.b"]
 8    − ["engine > activity > calling > parameters > arguments",
 9       "include?","SSN"],
10    − ["engine > activity > calling > UUID",
11       "==", "engine > activity > calling > parUUID"
12   c:
13    − ["engine > activity > calling > endpoint",
14       "==","http://solvency.db"]
15    − ["engine > source","==","Process Engine B"],
16    − ["engine > activity > calling > UUID",
17       "==", "engine > activity > calling > parUUID"
18 condition:
19   − ["a > engine > activity > receiving > received > loan","<=",1000000]
20   − "IMPLIES"
21   − "("
22    − ["a > worklist > user > take > Role","==","Manager"]
23    − ["b > engine > activity > receiving > received > solvency_level",
24       "==","A"]
25    − ["c > engine > activity > calling > parameters > retrieved_on",
26       "withindays",2]
27   − ")"
28 if: []
29 ifnot: [notify_warn]
```

1) Rule Clauses.: The clauses express compliance conditions. As seen in Lst. 2 clauses are defined in both the Match and Condition section, as for example in Rows 3 or 17. In accordance with Compliance Monitoring Functionality 2 on "data" as set out in [1] we support *unary* and *extended* conditions of the form "$d \odot v$" where d is a process data element and $\odot$ is a comparison operator. For unary conditions, "v is some value of d's domain" [1]. Examples are Clauses 1 and 2 in Tab. I. For extended conditions v refers to paths which reference values within the data structure. Then basically two values are processed against each other whereby one of them constitutes a reference to the data structure. An example is Clause 3 in Tab. I.

2) Referencing data structure values.: Based on the underlying instance activity life-cycle model, COMS monitors specific events which are defined in the rule clause. These references are therefore paths which seek for the relevant event data. The paths follow the same structure as shown in Fig 2 except for the first two hierarchy levels. They are abstracted in the reference, as they identify the process activity instance within the data structure.

TABLE II
RULE REFERENCE STRUCTURE – EXAMPLES

source	event topic	event name	event data structure		
engine	activity	calling	endpoint		
engine	activity	calling	parameters	arguments	form
engine	dataelements	change	changed	schadenssumme	
worklist	user	take	user		
worklist	user	giveback	cpee_activity		

a) Path.: The basic syntax for reference data values, described by a regular expression, is as follows: `/^(\w|\s|\>)+\w+\$/`

A sequence of characters is defined by a word as a start, followed by a “>”, which operates as separator. This pattern can be repeated multiple times and finally the query must end with a word. Every element between the “>” character is hereby an entity which is looked up. The first entity hereby defines the element with the highest hierarchy, going deeper with each given entity. Compared to the first three hierarchy levels as displayed in Fig. 2, the level ”Event Data“ can have an arbitrary depth. An example for this rule reference structure is represented in Tab. II: the hierarchy elements in the three leftmost columns relate to the event. The elements in the two rightmost columns relate to the event data attributes.

C. Rule Checking

If the process matches the semantics defined in the match section of a rule, the activity is mapped to a symbol. For the rule example in Lst. 2, the symbols “a”, “b” and “c” are later used in the conditions section, to define which condition should be applied to which activity.

In order to express structural compliance conditions, currently, COMS supports selected LTL patterns [13], i.e., (eventual) existence of an activity, absence of an activity, and a follows relations between activities. We have evaluated the expressiveness of COMS rules along the “Compliance Monitoring Functionalities framework (CMFF)” defined in [1]. COMS exhibits full coverage of CMF1 – 3 regarding data, time, and resources. Due to the explicit support of the instance activity life cycle, the execution CMF 4 – 6 are well-supported. Regarding the user-oriented CMF 7 – 10 COMS particularly supports root cause analysis as the source (system) of compliance violations can be explicitly determined.

1) Conditions.: Rule queries within the conditions must also include a reference to the matched activities. This reference is represented in the first entity of the rule query. For example the rule query “$a > worklist > user > take$” within Lst. 2 would reference the activity “a” defined in the *“match”* part. Therefore the value will be retrieved from the activity which matched the defined pattern in “a”. A rule file can include multiple conditions. Conditions are, by default joined with an ”AND“ operator. Additionally they can be connected with an ”OR“ or an ”IMPLIES“ operator.

2) DSL.: To be able to define more complex conditions, in the operator of a clause, a DSL is introduced. Besides the basic operators ”==“, ”<=“, ”!=“, additional operators can be used. These operators depend on the data type and allow all operators for this type based on the ruby programming language as well as the possibility to define custom operators.

3) Actions.: Depending on the outcome of the conditions actions are executed. In case the overall condition evaluates as true the ”if“ part is executed, otherwise the ”ifnot“ part will be invoked. In the example rule in Lst. 2, the actions are defined in Rows 28 and 29.

Actions are code which is supplied with the rules to the compliance management system. It uses a blackboard concept [14] to share data. As actions are executed they may persist data on the blackboard (e.g. store an average of certain values). Blackboard data may either be shared between all event sources, or between all tasks/event sources inside an instance (cmp. to Lst. 1):

Listing 3. Example Data represented in YAML
```
:08f4aa54−f73d−435a−b15f−a7e354d7706f:
  :Enter Data: ...
  :Retrieve from DB: ...
  :Blackboard:
    event: changed
    values:
      somevalue: 1
:Blackboard: (see above)
```

Changing a value on a blackboard generates an additional event, that can be caught by the compliance management system. Thus, as seen in Lst. 3, the blackboard is treated just like any event source, being either equal to an activity (e.g. ”Enter Data”) or an instance (e.g. :08f4aa54. . .).

D. Matching Performance

When looking at the way the data and the events are structured, and the examples above, it becomes apparent that the complexity of the rule-base and the events is fairly limited (see also CRISP rule-base [15]). Thus we can optimise the algorithm that compares the rules with the events.

To evaluate the defined rules, during initialisation all rules are parsed and common clauses are joined into one matching set. For example all rules with the clause [”engine > activity > calling > endpoint”,”==”,”https://worklist.do”] will be joined, as shown in Lst. 2. The resulting list of clauses point to each rule the are contained in, and how many clauses each originating rule contains.

Listing 4. Joining rules to matching sets
```
def join_rules(rulesets)
  matchbase = new Hash
  foreach rule in ruleset
    foreach clause in rule
      matchbase[clause] << [rule_id,number_of_clauses]
    end
  end
end
```

It is important to note, that there is no differentiation between clauses in events and clauses in condition necessary, as they all have to match in order for an action to be triggered.

The resulting match-base is a hash that can be accesses with O(1) if the key is known. In an incoming event, which is used to prepare a blackboard view, each value is a leaf, and the path to each leaf is the key that can be used to access the the condition and information about matching rules in the ruleset.

Listing 5. Store matches of rules

```
def on_event(event)
  foreach leaf in event
    if clause = matchbase[leaf] then
      foreach ref in clause
        matches[instance + task][ref−>rule_id] << leaf
        if count(
          unique(
            matches[instance + task][ref−>rule_id]
          )
        ) == ref−>number_of_clauses
          ACTION
        end
      end
    end
  end
end
```

As can be seen above, access to the match-base is constant, the runtime of the algorithm is mostly dependent on the pieces of information in each event.

IV. Evaluation

The *feasibility* of COMS is demonstrated based on a prototypical implementation, which is made available here[1], also including the logs used for the evaluation.

A. Evaluation of Applicability

The approach is also evaluated with respect to its *applicability* based on a real world event stream of a manufacturing process operated at the Austrian Competence Center for Digital Production (CDP)[2]. We analyze the event stream emitted by a EMCO ”MaxxMill 500“ during the execution of a process instance which orchestrates the milling of a part. After triggering the manufacturing of the part, the process instance running on the Cloud Process Execution Engine (CPEE) [9] collects data from the mill from multiple sources/sensors simultaneously. Through one sensor connected by the MT-Connect[3] [16] protocol it is possible to detect the overall power consumption of the machine. The goal is, based on energy consumption, to find out about (1) explicit machine errors during milling and (2) to monitor the power level during milling. While the mill has some explicit error states, it cannot detect (a) when a milling tool breaks and (b) when the edge of a milling tool rapidly deteriorates.

For the approach presented it this paper, it makes no difference if the event stream origins from one source, or many, as it is collected into one generic **event data** structure.

We created a set of rules (including custom actions) that detect Errors (a) and (b) from the power levels during milling. If the milling tool breaks (and falls off) the power levels show a rapid decline, as there is no more resistance. If the milling tools' edge becomes suddenly blunt the power levels spike. For both cases we want the process (and thus the milling) to stop, so that the machine can be fixed. This minimises the time wasted due to faulty milling tools. The realisation of this scenario is based on events such as depicted in Lst. 6. The unit for the value (power consumption) is watt.

[1]https://github.com/pakoe/coms
[2]http://www.acdp.at/
[3]http://www.mtconnect.org/

Listing 6. Aggregated Event Split Into Single Events

```
event:
  cpee:lifecycle:transition: activity/receiving
  list:
    data_receiver:
      message:
        mimetype: application/json
        content:
          ID: pac51_65
          value: 1956.174
```

The realization was achieved by two simple rules. The rule depicted in Lst. 7 monitors the operation of the milling machine. In Rows 3 to 4 of Lst. 7, the milling task ”MaxxMill 500“ is identified based on the service invoked by the process instance. All activities, in all instances which make use of the ”MaxxMill 500“ are monitored.

Listing 7. Sensor Data Rule 1

```
1 match:
2   − :a :
3     − [ ”engine > activity > done > endpoint”, ”==”, ”https://centurio.work/data
          /mm500/signals/”],
4     − [ ”engine > source”, ”==”, ”MaxxMill 500”]
5 condition: [ [ ”a > engine > activity > receiving > received > message >
      content > pac52_65 > value”, ”exits?” ] ]
6 if: [blackboardize_power_value]
7 ifnot: []
```

The condition evaluates if the power value is present in the respective activity. If yes the value is saved to a blackboard as described in Sec. III-C3 (see Lst. 8), which is accessible across all process instances.

Listing 8. Action blackboardize_power_value

```
def blackboardize_power_value(ctx)
  value = ctx.a.engine.activity.receiving.received.message.content.pac52_65.value
  ctx.blackboard.values.last_value = value
end
```

The change of a value within the blackboard then triggers the rule depicted in Lst. 9.

Listing 9. Sensor Data Rule 2

```
match:
  − :a : [ [ ”blackboard > event”, ”==”, ”changed” ] ]
condition:
  − [ ”a > blackboard > values > last_value”, ”<”, ”a > blackboard > values
      > average_low” ]
  − [ ”a > blackboard > values > last_value”, ”>”, ”a > blackboard > values
      > average_high” ]
if: [engine_stop]
ifnot: [calculate_average]
```

Its condition evaluates if the latest sensor is out a certain power consumption band (lower bound, upper bound). If the value is outside of the band, the machine is stopped. If the value is within the defined band, the value contributes to the average power consumption (see action depicted in Lst. 10).

Listing 10. Action calculate_average

```
def calculate_average(ctx)
  ctx.blackboard.values.average =
    ( ctx.blackboard.values.average ∗ ctx.blackboard.values.average_count +
      ctx.blackboard.values.last_value) /
    ( ctx.blackboard.values.average_count + 1 )
  ctx.blackboard.values.average_count += 1
  ctx.blackboard.values.average_low = ctx.blackboard.values.average − 10
  ctx.blackboard.values.average_high = ctx.blackboard.values.average + 10
end
```

There are various benefits to this approach. As mentioned above, errors can be detected and fixed early, which leads to higher Overall Equipment Effectiveness (OEE) [17]. While this could be done through explicit process modelling, this would require explicit tasks for sharing the average values, which is not part of the business logic and complicates the process model. Thus a specific advantage of our approach is, that it is automatically affecting all current and future process models which invoke the milling machine.

B. Evaluation of Computational Complexity

As elaborated above, based on the example rule-set we make the following assumptions.

- Rule-sets are large, individual events are small.
- Actions are not allowed to trigger rules.

We then designed the data-structures around these decisions to allow for an algorithm that can yield optimal performance for the given problem. In comparison to more generic algorithms like RETE, which support deeply intertwined rule-bases, this allows for better runtime characteristics. In Tab. III, we outline some popular solutions, which could achieve the same results as our solution, and their runtime characteristics.

In the RETE Algorithm [18], the matchset is denoted as productions C (see Lst. 2), while the event data (see Lst. 1) is denoted as working memory W. Each individual line, in each event is thus a working memory entry *WME*. In the simplest case, whenever a new *WME* occurs, the runtime is C. However in RETE, things might get worse, as for C it is assumed that all elements of the matchset are strictly independent, basically forming one rule. If they are not, then $P = NP$.

Our algorithm is designed with the following properties (speaking in RETE terms):

- Each *WME* should be compared to C only once.
- Each element in C is independent, thus the problem is not NP hard, no trigger cascades can occur.
- Matching C is O(1) as we can utilise a hash structure, and do not have to deal with true conditions in the sense of RETE.

The problem we tackled, could be solved in RETE, but the runtime would be W^C, whereas the runtime for the algorithms shown in Lst. 5 is in the worst case $W * C$ (i.e. multilinear). As concluded by [19] in Chapter 6.1, we assume restrictions on W and C, which will lead to polynomial runtime in W and C, while the RETE algorithm is able to tackle much more complex problems, which can be linear in P.

A comparison of the algorithms shown Lst. 5 to techniques utilised in Prolog, seems fairer. Tabling should yield similar results, to our approach, whereas standard search and backtrack should be similar to the results given by RETE networks.

In Tab. III a summary of this discussion is shown, including the names of popular RETE based engines. For the case of Drools we assume that the Lazy RETE algorithm (asserting possible solutions in the network) can lead to significantly improved runtime, although its remains unclear if it can come close to $W * C$, as this step adds additional in relation to C

So a good solution in Prolog should perform as well as any implementation that relies on the algorithm shown in Lst. 5. As

TABLE III
RUNTIME CHARACTERISTICS

Prolog		Drools		Jessy	COMS
Tabling	Search & Backtrack	Rete	Lazy Rete	Rete	Hash Access
$W * C$	W^C	W^C	$< W^C$	W^C	$W * C$

mentioned before, our prototypical implementation is a micro-service with a REST interface relying on the Cloud Process Execution Engine for the data-streams, which is tedious in Prolog, but easy in any modern programming language.

V. RELATED WORK

The compliance of process models can be monitored online based on on event streams [1]. Online conformance checking [20] measures the conformance of process models with some behaviour of interest (e.g., expressed by behavioural patterns) based on events streams during runtime. COMS concepts can be used to integrate event streams for compliance monitoring and online conformance checking. As stated in [21] *“[s]tate-of-the-art conformance analysis techniques are typically optimised and devised for one-time use”.* While [21] advocates performance gains by conformance approximation, COMS designs its rules less expressive as declarative approaches such as MobuconLTL and MobuconEC (cf. pattern-based analysis in [1]), but “expressive enough” to be rapidly checked.

[22] proposes techniques “for aligning business process compliance and monitoring requirements in dynamic [business networks]”. However, the focus is not on the process and rule matching at the data level. Choreography compliance has been analysed in [23], [3]; the proposed compliability criterion refers to the property that a collaboration can comply with global and local compliance rules as well as assertions. Again the data aspect has not been considered here. Event-based compliance checking in business processes comprises approaches for Complex Event Processing (CEP). In [24] the events in the stream can be aggregated to match activities in the process model. [25] works in a similar way for event stream integration for manufacturing processes. Both approaches provide valuable mechanisms for aligning the process and event stream level, however, do not directly address uncertainty in the context of activity labels and different formats. The same holds for previous work on matching rules and processes [26], [12]. Several approaches address the matching between process models, e.g., [27] and process similarity (see survey in [11]). Here similarity between process models is based on label equivalence or similarity measures such as string edit distance. As suggested in [10] the presented approach abstracts from label comparison and uses attribute equivalence instead. The observation that a process activity consist out of more than one event and therefore the consideration of an instance activity lifecyle is also present in [28]. [28] applies a m:n mapping of events to predefined business processes based on the activity labels and textual description. Our work in contrast (1) operates at runtime and (2) utilises a multitude of aspects, while in the context of this paper relying on non-fuzzy properties of process execution, such as task IDs,

endpoints and exchanged data.

COMS can be also compared to existing business rule engines such as Drools [29]. In contrast to, e.g., Drools, our approach models the process execution domain itself, with multiple instances, activities, instance activity life-cycle stages and event attributes. Therefore the domain does not have to be modelled to save the data or make the information processable. The structure of the rule clauses is to some point similar to XPATH [30], this similarities arise as they both operate on tree structured data. COMS processes the information not as a whole, but incrementally, with every incoming event.

VI. Conclusion

The presented COMS framework supports flexible matching of business rules and process events from multiple sources. This, in turn, enables integrated compliance checking together with resolving uncertainty caused by heterogeneous labelling and different formats. COMS consists of a generic event data structure for storing events, a generic rule language, and concepts for matching process tasks and events. The evaluation shows that it is applicable to real-world scenarios and the concept is mature enough to be adapted to newly emerging requirements. COMS creates an holistic approach, by defining a fitting data structure for the process event streams and offers a rule language which is designed to operate on this structure. Future work will develop a graphical user interface and investigate on how to involve users into process and rule matching most efficiently. Moreover, we aim at conducting further case studies, e.g., in the logistics domain.

Acknowledgements: This work has been partly funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-072 and the Austrian Research Promotion Agency (FFG) via the "Austrian Competence Center for Digital Production" (CDP) under the contract number 854187.

References

[1] L. T. Ly, F. M. Maggi, M. Montali, S. Rinderle-Ma, and W. M. P. van der Aalst, "Compliance monitoring in business processes: Functionalities, application, and tool-support," *Information Systems*, vol. 54, pp. 209 – 234, 2015.

[2] M. Wang, K. Y. Bandara, and C. Pahl, "Distributed aspect-oriented service composition for business compliance governance with public service processes," in *Internet and Web Applications and Services*, 2010, pp. 339–344.

[3] W. Fdhila, S. Rinderle-Ma, D. Knuplesch, and M. Reichert, "Change and compliance in collaborative processes," in *Services Computing*, 2015, pp. 162–169.

[4] A. Artikis, A. Gal, V. Kalogeraki, and M. Weidlich, "Event recognition challenges and techniques: Guest editors' introduction," *ACM Trans. Internet Techn.*, vol. 14, no. 1, pp. 1:1–1:9, 2014.

[5] J. Schiefer, B. List, and R. M. Bruckner, "Process data store: A real-time data store for monitoring business processes," in *Database and Expert Systems Applications*, 2003, pp. 760–770.

[6] IEEE, "Ieee standard for extensible event stream (xes) for achieving interoperability in event logs and event streams," *IEEE Std 1849-2016*, pp. 1–50, Nov 2016.

[7] B. F. van Dongen and W. M. Van der Aalst, "A meta model for process mining data." *EMOI-INTEROP*, vol. 160, p. 30, 2005.

[8] M. zur Muehlen and K. D. Swenson, "BPAF: A standard for the interchange of process analytics data," in *Business Process Management Workshops*, 2010, pp. 170–181.

[9] J. Mangler, G. Stuermer, and E. Schikuta, "Cloud process execution engine - evaluation of the core concepts," CoRR, Tech. Rep., 2010.

[10] S. Rinderle-Ma, M. Reichert, and M. Jurisch, "On utilizing web service equivalence for supporting the composition life cycle," *Int. J. Web Service Res.*, vol. 8, no. 1, pp. 41–67, 2011.

[11] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling, "Similarity of business process models: Metrics and evaluation," *Inf. Syst.*, vol. 36, no. 2, pp. 498–516, 2011.

[12] J. Mangler and S. Rinderle-Ma, "Rule-based synchronization of process activities," in *Commerce and Enterprise Computing*, 2011, pp. 121–128.

[13] F. M. Maggi, M. Westergaard, M. Montali, and W. M. P. van der Aalst, "Runtime verification of ltl-based declarative process models," in *Runtime Verification*, 2011, pp. 131–146.

[14] D. D. Corkill, "Blackboard systems," *AI expert*, vol. 6, no. 9, pp. 40–47, 1991.

[15] W. Fdhila, M. Gall, S. Rinderle-Ma, J. Mangler, and C. Indiono, "Classification and formalization of instance-spanning constraints in process-driven applications," in *International Conference on Business Process Management*. Springer, 2016, pp. 348–364.

[16] A. Vijayaraghavan, W. Sobel, A. Fox, D. Dornfeld, and P. Warndorf, "Improving machine tool interoperability using standardized interface protocols: Mt connect," in *Flexible Automation*, 2008.

[17] Õ. Ljungberg, "Measurement of overall equipment effectiveness as a basis for tpm activities," *International Journal of Operations & Production Management*, vol. 18, no. 5, pp. 495–507, 1998.

[18] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," in *Readings in Artificial Intelligence and Databases*, J. Mylopolous and M. Brodie, Eds. San Francisco (CA): Morgan Kaufmann, 1989, pp. 547 – 559.

[19] R. B. Doorenbos, "Production matching for large learning systems." Carnegie-Mellon Univ., Dept. of Computer Science, Tech. Rep., 1995.

[20] A. Burattin, S. J. van Zelst, A. Armas-Cervantes, B. F. van Dongen, and J. Carmona, "Online conformance checking using behavioural patterns," in *Business Process Management*, 2018, pp. 250–267.

[21] P. M. Dixit, H. M. W. Verbeek, and W. M. P. van der Aalst, "Fast conformance analysis based on activity log abstraction," in *Enterprise Distributed Object Computing*, 2018, pp. 135–144.

[22] M. Comuzzi, "Alignment of process compliance and monitoring requirements in dynamic business collaborations," *Enterprise IS*, vol. 11, no. 6, pp. 884–908, 2017.

[23] D. Knuplesch, M. Reichert, W. Fdhila, and S. Rinderle-Ma, "On enabling compliance of cross-organizational business processes," in *Business Process Management*, 2013, pp. 146–154.

[24] E. Mulo, U. Zdun, and S. Dustdar, "Domain-specific language for event-based compliance monitoring in process-driven soas," *Service Oriented Computing and Applications*, vol. 7, no. 1, pp. 59–73, 2013.

[25] S. Appel, P. Kleber, S. Frischbier, T. Freudenreich, and A. P. Buchmann, "Modeling and execution of event stream processing in business processes," *Inf. Syst.*, vol. 46, pp. 140–156, 2014.

[26] C. Indiono, J. Mangler, W. Fdhila, and S. Rinderle-Ma, "Rule-based runtime monitoring of instance-spanning constraints in process-aware information systems," in *On the Move to Meaningful Internet Systems*, 2016, pp. 381–399.

[27] H. Leopold, M. Niepert, M. Weidlich, J. Mendling, R. M. Dijkman, and H. Stuckenschmidt, "Probabilistic optimization of semantic process model matching," in *Business Process Management*, 2012, pp. 319–334.

[28] T. Baier, C. Di Ciccio, J. Mendling, and M. Weske, "Matching events and activities by integrating behavioral aspects and label analysis," *Software & Systems Modeling*, vol. 17, no. 2, pp. 573–598, 2018.

[29] M. Proctor, "Drools: A rule engine for complex event processing," in *Applications of Graph Transformations with Industrial Relevance*, 2012, pp. 2–2.

[30] J. Robie, J. Snelson, D. Chamberlin, and M. Dyck, "XML path language (XPath) 3.0," W3C, W3C Recommendation, Apr. 2014, http://www.w3.org/TR/2014/REC-xpath-30-20140408/.