Finding Non-compliances with Declarative Process Constraints through Semantic Technologies

Claudio Di Ciccio[2], Fajar J. Ekaputra[1], Alessio Cecconi[2],
Andreas Ekelhart[1], Elmar Kiesling[1]

[1] TU Wien, Favoritenstrasse 9-11, 1040 Vienna, Austria,
{firstname.lastname}@tuwien.ac.at
[2] WU Vienna, Welthandelsplatz 1, 1020 Vienna, Austria
{claudio.di.ciccio,alessio.cecconi}@wu.ac.at

Abstract. Business process compliance checking enables organisations to assess whether their processes fulfil a given set of constraints, such as regulations, laws, or guidelines. Whilst many process analysts still rely on ad-hoc, often handcrafted per-case checks, a variety of constraint languages and approaches have been developed in recent years to provide automated compliance checking. A salient example is DECLARE, a well-established declarative process specification language based on temporal logics. DECLARE specifies the behaviour of processes through temporal rules that constrain the execution of tasks. Existing approaches typically report on compliance at the aggregate level of binary evaluations of constraints on execution traces. Consequently, their results lack granular information on violations and their context, which hampers auditability of process data for analytic and forensic purposes. To address this challenge, we propose a novel approach that leverages semantic technologies for compliance checking. Specifically, our proposed approach provides not only global compliance metrics, but reports on particular non-compliant cases by identifying and contextualising violating events. Our approach proceeds in two stages. First, we translate DECLARE templates into statements in SHACL, a graph-based constraint language. Then, we evaluate the resulting constraints on the graph-based, semantic representation of process execution logs. We demonstrate the feasibility of our approach by testing its implementation on real-world event logs. Finally, we discuss its implications and future research directions.

Keywords: Process mining, compliance checking, SHACL, RDF, SPARQL

1 Introduction

The rationale of declarative approaches to business process specification is the normative description of processes' behaviour by means of inviolable rules. Those rules, named *constraints*, exert restrictions on the process behaviour. DECLARE [1], Dynamic Condition Response Graphs (DCR Graphs) [24], and the Case Management Model and Notation (CMMN)[3] are examples of such declarative process specification languages. Declarative process discovery is the branch of the process mining discipline [2] aimed at extracting the constraints that specify a process by their verification over execution data, namely

[3] https://www.omg.org/spec/CMMN/1.1

event logs. To date, the majority of research endeavours in the area has been devoted to the devising of efficient algorithms for the mining of constraints verified in the event log [9,32,14,11]. All such approaches return constraints that are never (or rarely) violated in the event log, thus implicitly assuming that rarity corresponds to noise. Similarly, approaches have been proposed to verify the compliance and conformance of event logs with declarative process specifications [39,30,29,28], indicating whether, and in how far, constraint sets are satisfied by process executions.

However, it is often not sufficient to solely assess whether constraints are satisfied or not. Especially in cases with rare exceptions, knowing when and why certain constraints are violated is key. Those circumstances apply in many scenarios: To mention but a few, inspecting the root cause of enactment errors [15], deriving digital evidence of malicious behaviour [25], finding the initiation of drifts in the process [31], or identifying illicit inflows into digital currency services [26]. Metaphorically, we claim that finding the needle in the haystack is more informative than confirming that its composition is 99.999% compliant to the norms.

Against this background, we leverage semantic technologies for the verification and querying of event logs, in an approach that *(i)* checks whether those data structures comply with provided declarative process specifications, and *(ii)* if not, reports on the cause of violations of events at a fine-grained level. We evaluate our technique by applying our prototype on real-world event logs. We empirically show the insights into detected violations provided by detailed reports. The unprecedented opportunities brought about by the employment of semantic technologies go well beyond the promising results presented in this vision paper. We thus endow the discussion on the evaluation of our approach with considerations on potential improvements over the current achievements, especially in regard to further semantic analyses of the current results.

The remainder of the paper is structured as follows. Section 2 overviews the literature about declarative process mining and semantic technologies. Section 3 describes our approach, its workflow and components, from the bare event log to its semantic representation and reasoning. Section 4 evaluates and discusses the feasibility of the approach against real-life logs. Section 5 closes the paper highlighting the future paths.

2 Background and State of the Art

Our approach draws on two main strands of research: on the one hand, it is grounded in state-of-the-art process mining techniques for declarative specifications; on the other hand, it leverages validation concepts developed in the semantic web research community for fine-grained compliance checking.

Declarative process mining. A *declarative process specification* defines a process behaviour through a set of temporal rules (constraints) that collectively determine the allowed and forbidden traces. Specifically, constraints exert conditions that allow or forbid *target* tasks to be executed, depending on the occurrence of so-called *activation* tasks. A declarative specification allows for any process executions that do not violate those constraints. Each constraint is defined using a template that captures the semantics of the constraint, parametric to the constrained tasks. The number of parameters denote its cardinality. DECLARE [1] is a well-established declarative process specification language. Its semantics

Table 1: A collection of DECLARE constraints with their respective natural language explanation, and examples of accepted (✓) or violating (×) traces.

Constraint	Explanation	Examples			
PARTICIPATION(a)	a occurs at least *once*	✓ bcac	✓ bcaac	× bcc	× c
RESPONSE(a,b)	If a occurs, then b occurs eventually after a	✓ caacb	✓ bcc	× caac	× bacc
ALTERNATERESPONSE(a,b)	Each time a occurs, then b occurs eventually afterwards, and no other a recurs in between	✓ cacb	✓ abcacb	× caacb	× bacacb
PRECEDENCE(a,b)	b occurs only if preceded by a	✓ cacbb	✓ acc	× ccbb	× bacc
ALTERNATEPRECEDENCE(a,b)	Each time b occurs, it is preceded by a and no other b can recur in between	✓ cacba	✓ abcaacb	× cacbba	× abbabcb
NOTSUCCESSION(a,b)	a can never occur before b	✓ bbcaa	✓ cbbca	× aacbb	× abb

are expressed in Linear Temporal Logic on Finite Traces (LTL_f) [10]. It offers a repertoire of templates extending that of the seminal paper of Dwyer et al. [16]. Table 1 shows some of those. RESPONSE, for example, is a binary template stating that the occurrence of the first task imposes the second one to occur eventually afterwards. RESPONSE(a,b) applies the RESPONSE template to task a and b, meaning that each time that a occurs in a trace, b is expected to occur afterwards. ALTERNATERESPONSE(a,b) is subsumed by RESPONSE(a,b) as it restricts the statement by adding that a cannot recur before b. PRECEDENCE(a,b) switches activation and target as well as the temporal dependency: b can occur only if a occurred before. NOTSUCCESSION(a,b) establishes that after a, b cannot occur any more. Finally, PARTICIPATION(a) is a constraint stating that in every process execution, a must occur. It applies the PARTICIPATION unary template.

The fundamental input for process mining is the event log, i.e., a collection of recorded process executions (traces) expressed as sequences of events. For the sake of simplicity, we assume events to relate to single tasks of a process. The eXtensible Event Stream (XES) format is an IEEE standard[4] for the storage and exchange of event logs and event streams, based on XML. In *declarative process mining*, constraints are verified over event logs to measure their *support* (i.e., how often they are satisfied in the traces), and optionally their *confidence* (i.e., how often they are satisfied in traces in which their activation occurs), and *interest factor* (i.e., how often they are satisfied in traces in which both activation and target occur) [32,14]. Table 1 reports traces that satisfy or violate the aforementioned constraints. For instance, a trace like caacb satisfies RESPONSE(a,b) (increasing its mining measures) but violates ALTERNATERESPONSE(a,b) (decreasing it). Although a trace like bcc satisfies RESPONSE(a,b), it does not contribute to its confidence or interest factor, since the activation (a) does not occur. Support is thus an aggregate measure for compliance as those traces that do not comply with a constraint, decrease its value.

Semantic web technologies provide a comprehensive set of standards for the representation, decentralised linking, querying, reasoning about, and processing of semantically explicit information. We mainly base our approach on two elements of the semantic web technology stack: *(i)* the Resource Description Framework (RDF), which we use as a uniform model to express process information, declarative constraints, and validation reports, and *(ii)* the SHACL Shapes Constraint Language (SHACL), which provides a constraint specification and validation framework on top of RDF. In the following, we provide a brief overview of both of these standards.

[4] https://doi.org/10.1109/IEEESTD.2016.7740858

RDF is a data representation model published by the World Wide Web Consortium (W3C) as a set of recommendations and working group notes.[5] It provides a standard model for expressing information about *resources*, which in the context of the present paper represent traces, tasks, events, actors, etc. An RDF dataset consists of a set of statements about these resources, expressed in the form of triples $\langle s,p,o \rangle$ where s is a *subject*, p is a *predicate*, and o is an *object*; s and o represent the two resources being related whereas p represents the nature of their relationship. Formally, we define an RDF graph G as a set of RDF triples $(v_1,v_2,v_3) \in (U \cup B \cup L) \times U \times (U \cup B \cup L)$ where: U is an infinite set of constant values (called RDF *Uniform Resource Identifier (URI) references*); B is an infinite set of local identifiers without defined semantics (called *blank nodes*); and L is a set of values (called *literals*). For instance, a trace expressed in RDF may contain a statement such as `eventA xes:nextEvent eventB` to describe the temporal relationship between two particular events. A key advantage of RDF as a data model is its extensible nature, i.e., additional statements about `eventA` and `eventB` can be added at any time, adding concepts and predicates from various additional, potentially domain-specific vocabularies.

Furthermore, ontology specification languages such as the Web Ontology Language (OWL)[6] can be used to more closely describe the semantic characteristics of terms. Although beyond the scope of the present paper, this extensible semantic web technology stack opens up opportunities for the use of reasoners in compliance checking and in the interpretation of violations (e.g., generalisations, root cause analyses, etc.) [17,4]. Other benefits of RDF include interoperability, i.e., information expressed in RDF using shared vocabularies can be exchanged between applications without loss of meaning. Furthermore, it makes it possible to apply a wide range of general purpose RDF parsing, mapping, transformation, and query processing tools.

Finally, once transformed into RDF, information (such as process information and compliance checking rules) can be easily published online, interlinked, and shared between applications and organisations, which is particularly interesting in the context of collaborative processes.

SHACL is a W3C Recommendation and language for validating RDF graphs against a set of conditions.[7] These conditions are specified in the form of RDF *shape graphs* which can be applied to validate *data graphs*. SHACL thereby provides a flexible declarative mechanism to define arbitrary validity constraints on RDF graphs. Specifically, these constraints can be formulated using a range of constraint components defined in the standard, or as arbitrary constraints implemented in SPARQL Protocol and RDF Query Language (SPARQL).[8] Validation of a data graph against a shapes graph produces a validation report expressed in RDF as the union of results against all shapes in the shapes graph. Importantly, compliance checking in SHACL is performed gradually by validating each individual so-called *focus node* against the set of constraints that apply to it. Compliance checks therefore produce detailed validation reports that not only conclude whether a data graph conforms globally, but also reports on the specific violations encountered. In the context of

[5] http://www.w3.org/TR/rdf11-primer/
[6] http://www.w3.org/TR/owl2-primer/
[7] https://www.w3.org/TR/shacl/
[8] http://www.w3.org/TR/sparql11-query/

process compliance checking, this provides a foundation to not only determine whether a process trace conforms, but to also report on the specific declarative rules that are violated.

Semantic approaches to compliance checking. Several approaches have been developed to support organisations in their compliance efforts, each offering their own functional and operational capabilities [5,22]. The underlying frameworks, such as Process Compliance Language (PCL) [20], DECLARE [1], and BPMN-Q [3], offer different levels of expressiveness, and consequently, varying reasoning and modelling support for normative requirements [23]. Three main strategies for business process compliance checking have been proposed in the literature to date [22]: *(i) runtime monitoring* of process instances to detect or predict compliance violations (runtime compliance checking, online monitoring, compliance monitoring); *(ii) design-time* checks for compliant or non-compliant behaviour during the design of process models; *(iii) auditing* of log files in an offline manner, i.e., after the process instance execution has finished. Based on DECLARE, techniques such as the one described in [34] can discover declarative models, check process compliance to them, and apply the model constraints in online monitoring. Yet the result is at the granularity of a trace, without pointing at the cause. Remarkably, Maggi et al. [33] apply the outcome of that technique to repair BPMN [15] process models according to the constraints to which an event log is not compliant.

In our approach, we implement DECLARE constraints with semantic technologies. Related work on semantic approaches in process mining include the one of Thao Ly et al. [38], in which a vision towards life-time compliance is presented. The authors motivate the creation of global repositories to store reusable semantic constraints (e.g., for medical treatment). They advocate the importance of intelligible feedback on constraint violations. This includes the reasons for the violations, to help the user to devise strategies for conflict avoidance and compensation. Furthermore, the results of semantic process checks must be documented. In our paper, we meet those requirements. To reason about the data surrounding a task, semantic annotations of tasks are introduced in [21,19] as logical statements of the PCL language over process variables, to support business process design. Governatori et al. [18] show how semantic web technologies and languages like LegalRuleML (based on RuleML)[9] can be applied to model and link legal rules from industry to support humans in analysing process tasks over legal texts. They apply those semantic annotations in the context of a regularity compliance system. To that end, they extend their compliance checker Regorous [21] with semantics of LegalRuleML. An approach to include additional domain knowledge to the process perspective is introduced in [39]. They extract additional information from domain models and integrate this knowledge in the process compliance check. Pham et al. [36] introduce an ontology-based approach for business process compliance checking. They use a Business Process Ontology (BPO) and a Business Rules Ontology (BRO) including OWL axioms and SWRL[10] rules to model compliance constraints. They do not try to systematically translate compliance rules as in our approach, and mention that performance is a problem with standard reasoners. Our focus is different from the work presented so far in that our main aim is not to validate process models against rules, but verifying declarative specifications on existing process executions.

[9] http://wiki.ruleml.org/index.php/RuleML_Home
[10] https://www.w3.org/Submission/SWRL

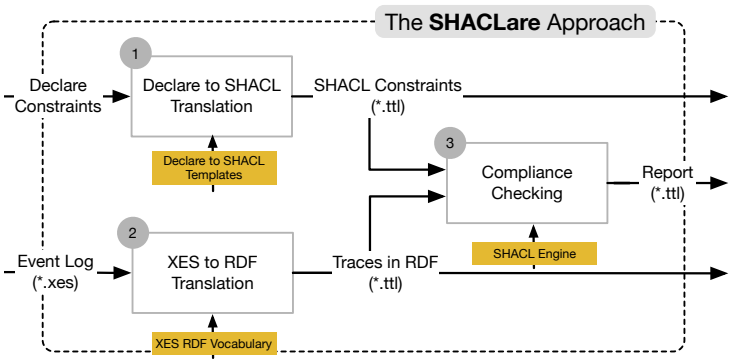


Fig. 1: SHACLARE components architecture

A Folder-Path enabled extension of SPARQL called FPSPARQL for process discovery is proposed in [6]. Similarly, Leida et al. [27] focus on the advantages of process representation in RDF and show the potential of SPARQL queries for process discovery. Our approach also builds on semantic representations of process data but explores how compliance checking can be conducted with semantic technologies.

3 Approach

Our approach applies a set of semantic technologies, including SHACL, for the compliance checking of DECLARE constraints on event logs. Because of that, we call it SHACLARE. As depicted in Fig. 1, the SHACLARE compliance checking workflow is composed as follows. ① We generate SHACL compliance constraints for a target process from a set of DECLARE constraints. In order to do that, we translate the DECLARE constraints through respective SHACL templates. ② We transform the event log into RDF, since SHACL constraints can only be validated against RDF graphs. This step takes an event log in XES format as input, and generates a traces file in RDF. ③ At this point, we have all the required input to run the SHACL compliance check, which results in a detailed report on each constraint violation.

① **DECLARE to SHACL.** In this first step, we generate SHACL compliance constraints for a target process. As input, we take a set of DECLARE constraints, which can be automatically discovered with tools such as MINERful [12]. Table 2 shows an excerpt of the declarative constraints retrieved by MINERful from the real-life Sepsis event log [35]. We then use the RDF Mapping language (RML)[11] to map each constraint (e.g., ALTERNATERESPONSE(Admission IC,CRP)) to the respective DECLARE RDF data model elements (available at `http://semantics.id/ns/declare`).

To enable SHACL validation on DECLARE constraints, we developed a complete set of SHACL templates to represent the DECLARE ones. Specifically, we analysed the DECLARE constraint semantics and derived a set of equivalent SHACL constraint templates,

[11] `http://rml.io/RMLmappingLanguage`

Table 2: Some DECLARE constraints discovered by MINERful [14] from the Sepsis log [35]

Template	Activation	Target	Support	Confidence	Interest factor
PARTICIPATION	ER Registration		1.000	1.000	1.000
NOTSUCCESSION	IV Antibiotics	ER Sepsis Triage	1.000	0.784	0.783
PRECEDENCE	CRP	Return ER	0.996	0.956	0.268
RESPONSE	ER Registration	ER Triage	0.994	0.994	0.994
ALTERNATERESPONSE	Admission IC	CRP	0.974	0.102	0.098

```
@prefix xes: <http://semantics.id/ns/xes#> .
@prefix dec:  <http://semantics.id/ns/declare#> .
@prefix decs: <http://semantics.id/ns/declare-shacl#> .
% ... deliberately left out rdf,rdfs,sh,owl, and xsd
<decs:param_URI>
    a sh:NodeShape ;
    sh:targetClass xes:Trace ;
    decs:baseConstraint <decs:param_declare> ;
    decs:baseConstraintClass dec:AlternateResponse ;
    sh:sparql [
        a sh:SPARQLConstraint ;
        sh:message "Each time 'decs:param_value1' occurs, then 'decs:param_value2' occurs afterwards,
      before 'decs:param_value1' recurs. Cause event: {?object}" ;
        sh:prefixes decs:namespace ;
        sh:select """
            SELECT $this ?object
            WHERE {
                $this xes:hasEventList/rdf:rest*/rdf:first ?object .
                {
                    # check if there is no "decs:param_value2" after "decs:param_value1"
                    ?object rdfs:label "decs:param_value1" .
                    FILTER NOT EXISTS {
                        ?object ^rdf:first/rdf:rest+/rdf:first ?object2 .
                        ?object2 rdfs:label "decs:param_value2" .                    }                }
                UNION                {
                    # check if there is no "decs:param_value2" between two "decs:param_value1"s
                    ?object rdfs:label "decs:param_value1" .
                    ?object ^rdf:first/rdf:rest+/rdf:first ?object2 .
                    ?object2 rdfs:label "decs:param_value1" .
                    FILTER NOT EXISTS {
                        ?object ^rdf:first/rdf:rest+/rdf:first ?item .
                        ?object2 ^rdf:first/^rdf:rest+/rdf:first ?item .
                        ?item rdfs:label "decs:param_value2" .
            }}}""" ;
    ];.
```

Listing 1: SHACL template exceprt for ALTERNATERESPONSE

similarly to what Schönig et al. [37] did with SQL. These SHACL templates are publicly available at `http://semantics.id/resource/shaclare` as documents written using the RDF Turtle notation.[12] Listing 1 shows an excerpt of SHACL template that represents ALTERNATERESPONSE in RDF Turtle notation. Lines 1-4 list the namespaces used in the templates. In addition to the standard `rdf`, `rdfs`, and `shacl` vocabularies defined by the W3C, we introduce the following vocabularies: *(i)* `desc:` provides a set of properties and parameters used in the template, which are specific to the SHACLARE approach, *(ii)* `dec:` defines the RDF classes and properties to describe the DECLARE language, and *(iii)* `xes:` provides the terms used to represents XES data in RDF. Line 5 contains a placeholder that will be replaced by the name of the particular DECLARE constraint instance during the compliance checking runtime. Lines 6-7 signify that the constraint will be validated against instances of class `xes:Trace`. Lines 8-9 specify that the SHACL constraint will

[12] `https://www.w3.org/TR/turtle/`

evaluate AlternateResponse, while line 12 provides a template for the violation constraint message. Lines 14-34 are the main part, representing the AlternateResponse Declare constraint template in its SHACL-SPARQL representation.

Subsequently, we translate the Declare constraints into SHACL constraints by injecting the constraint parameters `Activation` and `Target` in the SHACL template. All generated SHACL constraints are stored in memory for later execution, but also get persisted as a SHACL shape graph in the RDF Turtle `.ttl` format as a reference.

② **From XES to RDF.** The transformation takes as input an event log in XES format and translates it in RDF. To that end, we developed an XES RDF vocabulary (available at `http://semantics.id/ns/xes`), which provides the terms and concepts to represent XES data in RDF. We transform into main classes the basic XES objects: *(i)* `xes:Log` for the complete event log, *(ii)* `xes:Trace` to refer to individual traces, and *(iii)* `xes:Event` for single events. In addition to those classes, we define a set of data properties for the optional event attributes, such as `xes:timestamp` and `xes:transition`. Object properties (relation) bind together the elements, e.g., `xes:hasTrace` to relate `xes:Log` and `xes:Trace`, and `xes:hasEventList` to relate an `xes:Trace` and an `rdf:List` that contains event sequences.

③ **SHACL compliance checking.** Given the SHACL Declare constraints (①) and the RDF event log (②), we have all the required input to run the SHACL *Compliance Checking* component. All generated SHACL constraints are grouped according to their template (e.g., AlternateResponse) and subsequently checked one by one by a SHACL validation engine. The result of a SHACL validation is an RDF graph with one `ValidationReport` instance. In case the `conforms` attribute of a `ValidationReport` is false, a `result` instance provides further details for every violation, including: *(i)* `focusNode`, which points to the trace in which the violation occurred; *(ii)* `sourceShape`, linking the SHACL constraint instance which triggered the result; *(iii)* `resultMessage`, explaining the reason for the constraint violation with a natural-language sentence.

4 Evaluation

To demonstrate the efficiency and effectiveness of our compliance checking approach, we developed a proof-of-concept software tool, available for download at `gitlab.isis.tuwien.ac.at/shaclare/shaclare`. In this section, we report on its application on publicly available real-world event logs. In particular, we focus on the report produced on a specific event log to demonstrate the capability of SHACLare to provide detailed insights into compliance violations. The experiment environment, as well as the full set of input and output files are available on the aforementioned SHACLare GitLab project page.

The prototype. The prototype is implemented in Java, using a number of open source libraries,[13] including *(i)* Apache Jena, for RDF Graph manipulation and processing, *(ii)* OpenXES, for accessing and acquiring XES data into RDF Graph, *(iii)* caRML, for acquiring RDF representation of Declare constraints, and *(iv)* the TopBraid SHACL engine, for compliance checking. Based on the SHACLare approach description in Section 3, the following functions are available: ① translation of Declare to SHACL

[13] Apache Jena: `http://jena.apache.org/`; OpenXES: `http://code.deckfour.org/xes/`; caRML: `https://github.com/carml/carml/`; TopBraid: `https://github.com/topquadrant/shacl`.

Table 3: An excerpt of the SHACLARE validation report

Constraint	Trace	Message
dec:AlternateResponse_Admission+IC_CRP	xesi:trace/c677627d-079b-4585-87e4-8ea4ff11ff2a	Each time 'Admission IC' occurs, then 'CRP' occurs afterwards, before 'Admission IC' recurs. Cause event: xesi:event/A657AD0E-AD65-4E6F-B141-871A8C340B37
dec:Precedence_CRP_Return+ER	xesi:trace/4f1aff3f-b078-48e8-8e94-72c38c0ce6b7	If 'Return ER' occurs, 'CRP' must occur beforehand. Cause event: xesi:event/2BEFECDC-88C0-40E1-83D1-B41089C6A7C1

constraints, ② translation of XES event logs to its RDF Graph representations, and ③ compliance checking of event logs against defined constraints.

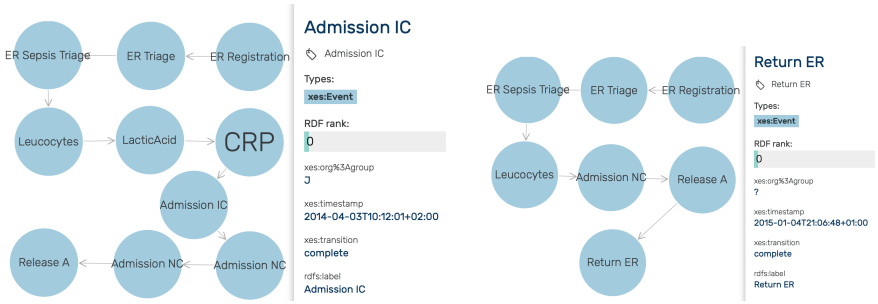
Insights into violations. The steps are illustrated on the example of the Sepsis treatment process event log [35]. That event log reports the trajectories of patients showing symptoms of sepsis in a Dutch hospital, from their registration in the emergency room to their discharge. Because of the reported flexibility of the healthcare process [35] and its knowledge-intensive nature, it is a suitable case for declarative process specification [13].

After running our prototype with the mined DECLARE constraints and the Sepsis event log as inputs, we receive as output the compliance report, the traces in RDF, and the set of checked constraints in SHACL (cf. Fig. 1). Those files form the basis of our analysis. We thus import the report and trace triples in GraphDB[14] to query and visualise the result data structures. An excerpt of non-compliant validation results is shown in Table 3.

Next, we explore why particular constraints are violated. For instance, the constraint ALTERNATERESPONSE(Admission IC,CRP) dictates that after every admission at the Intensive Care unit (Admission IC), a C-Reactive Protein test (CRP) must be performed later on, and the patient should not be admitted again at the IC before the CRP. As it can be noticed in Table 2, the support of the constraint is high (0.974), therefore we are interested in understanding what event(s) determined its violation. We thus extract the event `xesi:event/A657AD0E-AD65-4E6F-B141-871A8C340B37` signalled by the report and visualise it from the XES RDF representation. Figure 2(a) depicts a graph with the violating event and the connected events from its trace. We observe that the violation is due to the fact that the two events are swapped in the trace (CRP occurs before Admission IC). The right-hand side in the screenshot features a sample of metadata on the selected event (the violating activation, namely Admission IC). The tool allows us to scroll through all available properties and navigate through the trace, thus providing the user a powerful tool to inspect the violations in detail. Similarly, Fig. 2(b) shows that despite the high support of PRECEDENCE(CRP,Return ER) in the event log (0.996 as per Table 2), in one trace it was recorded that the patient was discharged and sent to the emergency room (Return ER), although no CRP took place before.

The analysis can be further extended through the integration of additional context information on the traces, which may be specified in domain-specific vocabularies. Furthermore, the by-products and results of our approach can be utilised for further analyses via semantic technologies. This can provide a rich interpretation context for the qualitative analysis of constraint violations. An important aspect is that it is possible to use all the metadata information directly in the SHACL constraints or later on in a SPARQL query. Therefore,

[14] http://graphdb.ontotext.com/



(a) ALT.RESPONSE(Admission IC,CRP) (b) PRECEDENCE(CRP,Return ER)

Fig. 2: Graphs illustrating the violations of constraints in the traces.

the analysis is readily extendible beyond the control flow perspective. The investigation of those extensions draws future plans for our research, outlined in the next section.

5 Conclusion and Future Work

In this vision paper, we proposed a novel approach for compliance checking leveraging semantic technologies, named SHACLARE. In particular, we provide a set of templates to translate DECLARE constraints into SHACL constraints, and subsequently validate them against a graph representation of XES process execution data. Thereby we overcome a typical limitation of existing compliance checking approaches, which lack granularity of information on violations and their context. The compliance reports provide links to involved traces, as well as to the events triggering constraint violations. Due to the semantic data structures, we can easily query and visualise connected events and inspect all metadata provided in the original log files. The preliminary results attained show how our approach could be used and extended further for auditability of event logs.

This new approach opens opportunities for future work in various directions. We aim at leveraging Linked Data frameworks so as to conduct compliance checking over multiple process data sources beyond single event logs, inspired by the seminal work of Calvanese et al. [8]. In that regard, the checking of richer constraints encompassing other perspectives than the sole control flow (i.e., data, time, resources, etc.) [7], draws our future research endeavours, driven by the capabilities readily made available to that extent by SHACL and SPARQL. This paper analysed the compliance checking setting, thus the analysis of constraints violations. If the goal is also to analyse the relevance of satisfactions, the sole verification of a formula is not sufficient and the vacuity problem must be taken into account [12]. We will investigate how to extend our approach to tackle this problem. Online compliance checking is another topic of interest. It would be worth exploring how SHACL constraints can be checked in a streaming fashion, potentially based on stream reasoning engines. In addition, we want to further (automatically) enrich the metadata extracted from the log data, using Natural Language Processing (NLP) and ontologies. From a formal analysis viewpoint, we will investigate the semantic equivalence of the SHACL constraints with respect to DECLARE LTL$_f$ formulas, and more in general the

expressive power of those languages. Finally, additional compliance checking tools could be developed to provide users with easy-to-use and feature-rich options to adopt this approach in their process management strategies.

Acknowledgements. This work was partially funded by the Austrian FFG grant 861213 (CitySPIN), the Austrian FWF / netidee SCIENCE grant P30437-N31 (SEPSES), the Austrian Federal Ministry for Digital, Business and Enterprise and the National Foundation for Research, Technology and Development.

References

1. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - Research and Development **23**(2), 99–113 (2009)
2. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
3. Awad, A., Decker, G., Weske, M.: Efficient compliance checking using BPMN-Q and temporal logic. In: BPM. pp. 326–341. Springer Berlin Heidelberg (2008)
4. Balduini, M., Celino, I., Dell'Aglio, D., Della Valle, E., Huang, Y., Lee, T., Kim, S.H., Tresp, V.: Reality mining on micropost streams. Semantic Web **5**(5), 341–356 (2014)
5. Becker, J., Delfmann, P., Eggert, M., Schwittay, S.: Generalizability and applicability of model-based business process compliance-checking approaches — a state-of-the-art analysis and research roadmap. Business Research **5**(2), 221–247 (2012)
6. Beheshti, S.M.R., Benatallah, B., Motahari-Nezhad, H.R., Sakr, S.: A query language for analyzing business processes execution. In: BPM. pp. 281–297. Springer Berlin Heidelberg (2011)
7. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016)
8. Calvanese, D., Kalayci, T.E., Montali, M., Santoso, A., van der Aalst, W.: Conceptual schema transformation in ontology-based data access. In: EKAW. pp. 50–67 (2018)
9. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. T. Petri Nets and Other Models of Concurrency **2**, 278–295 (2009)
10. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI. pp. 854–860 (2013)
11. Debois, S., Hildebrandt, T.T., Laursen, P.H., Ulrik, K.R.: Declarative process mining for DCR graphs. In: SAC. pp. 759–764. ACM (2017)
12. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: On the relevance of a business constraint to an event log. Information Systems **78**, 144–161 (2018)
13. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive Processes: Characteristics, requirements and analysis of contemporary approaches. J. Data Semantics **4**(1), 29–57 (2015)
14. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Manage. Inf. Syst. **5**(4), 24:1–24:37 (2015)
15. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
16. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE. pp. 411–420 (1999)
17. Francesconi, E.: Semantic model for legal resources: annotation and reasoning over normative provisions. Semantic Web **7**(3), 255–265 (2016)
18. Governatori, G., Hashmi, M., Lam, H.P., Villata, S., Palmirani, M.: Semantic business process regulatory compliance checking using LegalRuleML. In: EKAW. vol. 10024, pp. 746 – 761 (2016)

19. Governatori, G., Hoffmann, J., Sadiq, S., Weber, I.: Detecting regulatory compliance for business process models through semantic annotations. In: Business Process Management Workshops. pp. 5–17. Springer Berlin Heidelberg (2009)
20. Governatori, G., Rotolo, A.: A conceptually rich model of business process compliance. In: APCCM. pp. 3–12. Australian Computer Society, Inc. (2010)
21. Governatori, G., Shek, S.: Regorous: A business process compliance checker. In: ICAIL. pp. 245–246. ACM (2013)
22. Hashmi, M., Governatori, G.: Norms modeling constructs of business process compliance management frameworks: a conceptual evaluation. Artificial Intelligence and Law **26**(3), 251–305 (2018)
23. Hashmi, M., Governatori, G., Wynn, M.T.: Normative requirements for business process compliance. In: Service Research and Innovation. pp. 100–116. Springer International Publishing (2014)
24. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T.: Nested dynamic condition response graphs. In: FSEN. pp. 343–350. Lecture Notes in Computer Science, Springer (2011)
25. Kävrestad, J.: Fundamentals of Digital Forensics - Theory, Methods, and Real-Life Applications. Springer (2018)
26. Kuzuno, H., Karam, C.: Blockchain explorer: An analytical process and investigation environment for bitcoin. In: eCrime. pp. 9–16. IEEE (2017)
27. Leida, M., Majeed, B., Colombo, M., Chu, A.: A lightweight rdf data model for business process analysis. In: SIMPDA. pp. 1–23. Springer Berlin Heidelberg (2013)
28. de Leoni, M., Maggi, F.M., van der Aalst, W.M.: An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. Inf. Syst. **47**, 258–277 (2015)
29. Letia, I.A., Goron, A.: Model checking as support for inspecting compliance to rules in flexible processes. J. Vis. Lang. Comput. **28**, 100–121 (2015)
30. López, M.T.G., Parody, L., Gasca, R.M., Rinderle-Ma, S.: Prognosing the compliance of declarative business processes using event trace robustness. In: OTM. pp. 327–344. Springer (2014)
31. Maaradji, A., Dumas, M., Rosa, M.L., Ostovar, A.: Detecting sudden and gradual drifts in business processes from execution traces. IEEE Trans. Knowl. Data Eng. **29**(10), 2140–2154 (2017)
32. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: CAiSE. pp. 270–285. Springer (2012)
33. Maggi, F.M., Marrella, A., Capezzuto, G., Cervantes, A.A.: Explaining non-compliance of business process models through automated planning. In: ICSOC. pp. 181–197. Springer International Publishing (2018)
34. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: BPM. pp. 132–147. Springer Berlin Heidelberg (2011)
35. Mannhardt, F., Blinde, D.: Analyzing the trajectories of patients with sepsis using process mining. In: RADAR+EMISA. pp. 72–80. CEUR-ws.org (2017)
36. Pham, T.A., Le Thanh, N.: An ontology-based approach for business process compliance checking. In: IMCOM. pp. 1 – 6. ACM SIGAPP (2016)
37. Schönig, S., Rogge-Solti, A., Cabanillas, C., Jablonski, S., Mendling, J.: Efficient and customisable declarative process mining with SQL. In: CAiSE. pp. 290–305. Springer (2016)
38. Thao Ly, L., Göser, K., Rinderle-Ma, S., Dadam, P.: Compliance of semantic constraints - a requirements analysis for process management systems. In: GRCIS (2008)
39. Thao Ly, L., Rinderle-Ma, S., Dadam, P.: Design and verification of instantiable compliance rule graphs in process-aware information systems. In: CAiSE. pp. 9–23. Springer (2010)